Algebraic Algorithms in Combinatorial Optimization

CHEUNG, Ho Yee

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
September 2011

Thesis/Assessment Committee

Professor Shengyu Zhang (Chair)
Professor Lap Chi Lau (Thesis Supervisor)
Professor Andrej Bogdanov (Committee Member)
Professor Satoru Iwata (External Examiner)

Abstract

In this thesis we extend the recent algebraic approach to design fast algorithms for two problems in combinatorial optimization. First we study the linear matroid parity problem, a common generalization of graph matching and linear matroid intersection, that has applications in various areas. We show that Harvey's algorithm for linear matroid intersection can be easily generalized to linear matroid parity. This gives an algorithm that is faster and simpler than previous known algorithms. For some graph problems that can be reduced to linear matroid parity, we again show that Harvey's algorithm for graph matching can be generalized to these problems to give faster algorithms. While linear matroid parity and some of its applications are challenging generalizations, our results show that the algebraic algorithmic framework can be adapted nicely to give faster and simpler algorithms in more general settings.

Then we study the all pairs edge connectivity problem for directed graphs, where we would like to compute minimum s-t cut value between all pairs of vertices. Using a combinatorial approach it is not known how to solve this problem faster than computing the minimum s-t cut value for each pair of vertices separately. While a recent matrix formulation inspired by network coding directly implies a faster algorithm, we show that the matrix formulation can be computed more efficiently for graphs with good separators. As a consequence we obtain a faster algorithm for simple directed planar graphs, bounded genus graphs and fixed minor free graphs. Our algorithm takes amortized sublinear time to compute the edge connectivity between one pair of vertices.

摘要

在本論文我們用代數算法來解決組合優化中的兩個問題。首先，我們研究線性擬陣對等問題，我們證明了Harvey的線性擬陣交集算法可以很容易地推廣到線性擬陣對等問題。新算法比以前的更快和更簡單。對於一些可以歸結爲線性擬陣對等的圖論問題，我們再次證明Harvey的圖匹配算法可以推廣到這些問題中。雖然線性擬陣對等問題和它的應用都是一些具有挑戰性的問題，我們的結果表明代數算法框架可以很好地應用在更廣泛的問題中。

其次，我們研究在有向圖的邊連通度問題。在此問題中我們想知道每對頂點間的最小割數值。利用新的矩陣公式，在平面圖和它的推廣上我們可以更快速地解決此問題。攤分後我們的算法只需要次線性的時間來計算一對頂點之間的最小割數值。

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Lap Chi Lau for his enormous support. He is very patience with me, willing to spend hours discussing with me every week. He has enlightened me in both research and life. This thesis would not have been possible without his help.

I wish to thank my coauthor Kai Man Leung. Writing this thesis reminds me the time we worked hard, and also our last minute term paper submission in every semester.

I must also thank my 117-mate: Yuk Hei Chan, Siyao Guo, Tsz Chiu Kwok, Ming Lam Leung, Fan Li, Yan Kei Mak, Chung Hoi Wong and Zixi Zhang. We had interesting discussion and fun. They made my M.Phil. days enjoyable and delightful.

Finally, I thank my parents and sister for their endless love and support.

Contents

Chapter 1

Introduction

Combinatorial optimization is the study of optimization problems on discrete and combinatorial objects. This area includes many natural and important problems like shortest paths, maximum flow and graph matchings. Combinatorial optimization find its applications in real life problems such as resource allocation and network optimization. In a broader sense, combinatorial optimization also has applications in many fields like artificial intelligence, machine learning and computer vision. The study of combinatorial optimization gives a unified framework to characterize the optimal value through min-max formula and to design efficient algorithm to compute an optimal solution. Given broad applications of combinatorial optimization, we are interested in designing fast algorithms to solve these problems. Fast algorithms in combinatorial optimization are often based on the framework of finding augmenting paths and the use of advanced data structures.

On the other hand, there is another way to design fast algorithms using algebraic techniques. Using fast linear algebraic algorithms, such as computing matrix multiplication in $O(n^\omega)$ time where $\omega < 2.38$, we can solve problems in combinatorial optimization efficiently by characterizing these problems in matrix forms. This leads to fast algorithms for graph matching, all pairs shortest paths and counting triangles. Moreover, the determinant of a matrix, being a sum of factorial number of terms, can be computed in polynomial time gives us another direction to solve problems in combinatorial optimization efficiently. This fact allows us to

give the only known polynomial time algorithm to count the number of perfect matchings in planar graphs and the number of spanning trees in general graphs, and surprisingly the fastest known algorithms for graph matching, linear matroid intersection and determining whether a graph is Hamiltonian.

The connections between combinatorial optimization problems and matrix determinants are established by theorems that relate the optimal value of a problem to the rank of an appropriately defined matrix. An example is that the rank of the Tutte matrix of a graph is twice the size of a maximum matching. These results however do not directly imply fast algorithms to find the solution of the problems, such as the edges that belongs to a maximum matching. In a recent line of research, an elegant algorithmic framework has been developed for this algebraic approach. Mucha and Sankowski [48] showed how to use Gaussian elimination to construct a maximum matching in one matrix multiplication time, leading to an $O(n^\omega)$ time algorithm for the graph matching problem where n is the number of vertices. Harvey [30] developed a divide-and-conquer method to obtain the fastest algorithm for the linear matroid intersection problem, and also provide a simple $O(n^\omega)$ time algorithm for the graph matching problem. In addition, Mucha and Sankowski [49] showed that constructing a maximum matching in planar graphs only takes $O(n^{\omega/2})$ time, and Sankowski [62] also showed that there is an RNC algorithm for graph matching using only $O(n^\omega)$ processors. Furthermore, Sankowski [61] and Harvey [28] extended the algebraic approach to obtain faster pseudo-polynomial algorithms for the weighted bipartite matching problem and the weighted linear matroid intersection problem.

In this thesis we extend this approach to design fast algorithms for two problems in combinatorial optimization. First we study the linear matroid parity problem, a common generalization of graph matching and linear matroid intersection, that has applications in various areas. We show that Harvey's algorithm for linear matroid intersection can be easily generalized to linear matroid parity. This gives an algorithm that is faster and simpler than previous known algorithms. For some graph problems that can be reduced to linear matroid parity, we again show that Harvey's algorithm for graph matching can be generalized to these problems to give faster algorithms. While linear matroid parity and some of its applications

are challenging generalizations for the design of combinatorial algorithms, our results show that the algebraic algorithmic framework can be adapted nicely to give faster and simpler algorithms in more general settings.

Then we study the all pairs edge connectivity problem for directed graphs, where we would like to compute the minimum s-t cut value between all pairs of vertices. Using a combinatorial approach it is not known how to solve this problem faster than finding the minimum s-t cut value for each pair of vertices separately. However, using the idea of network coding, it was shown recently there is a matrix formulation for the all pairs edge connectivity problem. This implies an $O(m^\omega)$ algorithm to compute the edge connectivity for every pair, where m is the number of edges in the graph. We show that such a matrix formulation can be computed more efficiently for graphs with good separators. As a consequence we obtain a $O(d^{\omega-2}n^{\omega/2+1})$ time algorithm for the all pairs edge connectivity problem in simple directed planar graphs, bounded genus graphs and fixed minor free graphs, where d is the maximum degree of the graph. This implies it takes amortized sublinear time to compute edge connectivity between one pair of vertices, faster than the combinatorial method.

The results in this thesis are based on joint work with Lap Chi Lau and Kai Man Leung [11, 12].

Chapter 2

Background

In this chapter we review some backgrounds on matroids, matrices and algebraic algorithms. We first present some examples and applications of matroids and matrices. Then to see how we solve combinatorial optimization problems algebraically, we present some matrix formulations for these problems, which relate optimal values of these problems to ranks of these matrices. After that we will see some algebraic tools which help us to design fast algorithms. Finally we review some previous algebraic algorithms and techniques for the graph matching problem and the graph connectivity problem.

2.1 Matroids and Matrices

Matroid is a discrete structure that generalizes the concept of independence in linear algebra, and has many applications in combinatorial optimization. In this section we will introduce matroid and also describe some of its applications in graphs.

A *matroid* is a pair $M = (V, \mathcal{I})$ of a finite set V and a set $\mathcal{I}$ of subsets of V such that the following axioms are satisfied

1. $\emptyset \in \mathcal{I}$,
2. $I \subseteq J \in \mathcal{I} \implies I \in \mathcal{I}$,

3. $I, J \in \mathcal{I}, |I| < |J| \implies \exists v \in J \setminus I$ such that $I \cup \{v\} \in \mathcal{I}$.

We call V the *ground set* and $I \in \mathcal{I}$ an *independent set*. So $\mathcal{I}$ is the *family of independent sets*. *Bases* $\mathcal{B}$ of M are independent sets with maximum size. By the above axioms, all bases have the same size. For any $U \subseteq V$, the *rank* of U, denoted by $r_M(U)$, is defined as

$$r_M(U) = \max\{|I| \mid I \subseteq U,\ I \in \mathcal{I}\}.$$

Given a matrix M, the submatrix containing rows S and columns T is denoted by $M_{S,T}$. A submatrix containing all rows (or columns) is denoted by $M_{*,T}$ (or $M_{S,*}$), and an entry of M is denoted by $M_{i,j}$. Also let $\vec{e_i}$ to be a column vector with an 1 in the i-th position and 0 otherwise. We denote the matrix whose column vectors are v_1, v_2, $\cdots$, v_m as $(v_1|v_2|\cdots|v_m)$. Throughout this thesis, we work with matrices over a finite field $\mathbb{F}$, and we assume any field operation can be done in $O(1)$ time.

2.1.1 Examples

Here we give some examples of matroids. We will use the following matroids to model some graph problems later.

Linear Matroid: Let Z be a matrix over a field $\mathbb{F}$, and V be the set of the column vectors of Z. The linear independence among the column vectors of Z defines a matroid M on ground set V. A set $I \subseteq V$ is independent in M if and only if the column vectors indexed by I are linearly independent. The rank function r of M is simply defined as $r_M(I) = \text{rank}(Z_{*,I})$. A matroid that can be represented in this way is said to be *linearly representable over* $\mathbb{F}$.

Partition Matroid: Let $\{V_1, \cdots, V_k\}$ be a partition of ground set V, that is, $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. Then the family of the independent sets

$\mathcal{I}$ on the ground set V is given by

$$\mathcal{I} = \{I \subseteq V : |I \cap V_i| \leq 1 \ \ \forall i \in \{1, \cdots, k\}\}.$$

$M = (V, \mathcal{I})$ is called a *partition matroid*. Partition matroids are linearly representable. This can be done by representing each element $v \in V_i$ as a vector $\vec{e}_i$.

Graphic Matroid: Let $G = (V, E)$ be a graph with vertex set V and edge set E. A *graphic matroid* has ground set E. A set $I \subseteq E$ is independent if and only if I contains no cycles in G. The matroid is *linearly representable* by representing each edge $(u, v) \in E$ to a column vector $\vec{e}_u - \vec{e}_v$ in the linear matroid.

2.1.2 Constructions

Now we discuss some ways to construct a new matroid from an existing matroid, these constructions will be used later in our algorithms.

The *restriction* of a matroid M to $U \subseteq V$, denoted as $M|U$, is a matroid with ground set U so that $I \subseteq U$ is independent in $M|U$ if and only if I is independent in M. This is the same as saying $M|U$ is obtained by deleting the elements $V \setminus U$ in M. The rank function $r_{M|U}$ of $M|U$ is simply $r_{M|U}(I) = r_M(I)$ for $I \subseteq U$.

The *contraction* of $U \subseteq V$ from a matroid M, denoted by M/U, is a matroid on ground set $V \setminus U$ so that $I \subseteq V \setminus U$ is independent M/U if and only if $I \cup B$ is independent in M where B is a base of $M|U$. The rank function $r_{M/U}$ of M/U is given by

$$r_{M/U}(I) = r_M(I \cup U) - r_M(U), \qquad I \subseteq V \setminus U.$$

For any matrix Z and its corresponding linear matroid M, the matrix for $M/\{i\}$ can be obtained by Gaussian eliminations on Z as follows. First, using row operation and scaling we can transform the column indexed by i to a unit vector $\vec{e}_k$. Then the matrix obtained by removing i-th column and k-th row from M is the required matrix. It can be seen that $I \cup \{i\}$ is independent in M if and only if I is independent in $M/\{i\}$.

2.1.3 Matroid Intersection

Given two matroids $M_1 = (V, \mathcal{I}_1)$ and $M_2 = (V, \mathcal{I}_2)$ on the same ground set V, the *matroid intersection problem* is to find a maximum size common independent set of the two matroids. Edmonds [17] proved a min-max formula to characterize the maximum value, and showed that both the unweighted and weighted version of the problem can be solved in polynomial time [18]. The fastest known algorithm for linear matroid intersection is given by Harvey [30]. He gives an algebraic randomized algorithm with running time $O(mr^{\omega-1})$, where m is the size of the ground set and r is the rank of both matroids. The algorithm assumes both matroids are representable over the same field, but this already covers most of the applications of the matroid intersection problem.

Applications

Bipartite Matching A standard application of matroid intersection is bipartite matching. Consider a bipartite graph $G = (U, V, E)$. For each edge (u, v) where $u \in U$ and $v \in V$, associate vector $\vec{e}_u$ to M_1 and $\vec{e}_v$ to M_2. Now a set of edges forms a bipartite matching if and only if their corresponding vectors in M_1 are linearly independent (so that these edge do not share vertices in U) and their corresponding vectors in M_2 are linearly independent. Thus a maximum size common independent set of M_1 and M_2 corresponds to a maximum size bipartite matching.

Colorful spanning tree Another application of matroid intersection is the colorful spanning tree problem. In this problem we are given an undirected multigraph $G = (V, E)$ where each edge is colored by one of the $k \leq n$ colors, and the objective is to find a spanning tree T in G such that each edge in T has a distinct color. Note that finding an arborescence of a directed graph can be reduced to the colorful spanning tree problem.

The distinct color constraint can be modeled by a partition matroid just like in bipartite matching. In matroid M_1 an edge with color d is represented by a vector $\vec{e}_d$. So a set of edges are of distinct colors if their corresponding vectors for M_1 are

linearly independent. The tree constraint can be captured by a graphic matroid M_2, where an edge (u, v) is represented by a vector $\vec{e}_u - \vec{e}_v$. So a set of edges are acyclic if and only if their corresponding vectors for M_2 are linearly independent. Thus a maximum size common independent set of M_1 and M_2 gives a maximum size acyclic colorful subgraph of G, so it remains to check if the subgraph forms a spanning tree.

In Section 3.4.3 we will present a faster algorithm for the colorful spanning tree problem than directly applying a generic matroid intersection algorithm.

Other Applications The matroid intersection problem finds its applications in various area including routing [10], constrained minimum spanning tree [27, 32], graph connectivity [21, 38], graph orientation [63](Section 61.4a), network coding [31], mixed matrix theory [51] and electrical systems [51].

2.1.4 Matroid Parity

Given a matroid M whose elements in its ground set are given in pairs where each element is contained in exactly one pair, the *matroid parity problem* is to find a maximum cardinality collection of pairs so that the union of these pairs is an independent set of M. The general matroid parity problem is NP-hard on matroids with compact representations [42], and is intractable in the oracle model [35] where we are given an oracle which tells whether a set of elements is an independent set. For the important special case when the given matroid is a linear matroid, it was shown to be polynomial-time solvable by Lovász [42]. The linear matroid parity problem can be interpreted as given vectors in pairs $\{(b_1, c_1) \cdots (b_m, c_m)\}$, and the objective is to find a maximum cardinality collection of pairs so that the vectors chosen are linearly independent.

Applications

The linear matroid parity problem is a generalization of graph matching and linear matroid intersection, and has many applications of its own in different areas.

For example, the $\mathcal{S}$-path packing problem [46, 63], the minimum pinning set problem [42, 36], the maximum genus imbedding problem [20], the unique solution problem [44] in electric circuit, approximating minimum Steiner tree [59, 3] and approximating maximum planar subgraph [8]. In this section, we will highlight some of these applications.

Graph Matching For each edge (u, v) we construct a column pair $\vec{e}_u$ and $\vec{e}_v$. Then a set of edges is a matching if and only if their column pairs are linearly independent, because two columns can only be dependent if they represent the same vertex.

Linear Matroid Intersection Here we have to assume both matroid can be represented over the same field. Let M_1 and M_2 be the matrix representation of the two matroids. Let their dimension be $r_1 \times m$ and $r_2 \times m$ respectively. We create m columns pairs, where each column has size $r_1 + r_2$. To construct the i-th column pair, put column i of M_1 to the top r_1 entries of b_i, and put column i of M_2 to the last r_2 entries of c_i. By this construction, any b_i do not interfere with any c_j. For a column set C that $(M_1)_{*,C}$ and $(M_2)_{*,C}$ are both independent, the columns in $\{\{b_i, c_i\}|i \in C\}$ are also independent, and vice versa.

Minimum pinning set Let $G = (V, E)$ be a graph whose vertices are points in the Euclidean plane, and whose edges are rigid bars with flexible joints at the vertices. In the minimum pinning set problem [36], we want to pin down a minimum number of vertices, so that the graph become rigid. We say a graph is rigid if it has no non-trivial smooth motions.

It is known that the problem can be reduced to linear matroid parity [36]. We construct a linear matroid parity instance with $|V|$ column pairs where each column has size $|E|$. For each edge $e_i = (u, v)$, at row i we set

$$(b_u)_i = x_u - x_v, \quad (b_v)_i = x_v - x_u, \quad (c_u)_i = y_u - y_v, \quad (c_v)_i = y_v - y_u$$

where x_v and y_v are x and y coordinates of vertex v respectively.

Notice that non-zero entries are very sparse, and it might be possible to use this property to obtain more efficient algorithms than directly applying a generic algorithm for linear matroid parity.

Graphic Matroid Parity In this matroid parity problem the given matroid is a graphic matroid. Thus each column is in the form $\vec{e}_u - \vec{e}_v$, which represent an edge (u, v), and thus its name. Such columns are independent as long as the subgraph containing these edges are acyclic. Hence graphic matroid parity models the situation where edges are given in pairs, and we would like to pick the most number of edge pairs so that the union of these edges are acyclic.

In Section 3.4.2 we will present an efficient algorithm for this problem. An application of graphic matroid parity is the maximum planar subgraph problem, where we want to pick the maximum number of edges of a graph while remains planar. The problem is NP-complete but can be approximated using graphic matroid parity [8].

Approximating minimum Steiner tree Given an edge weighted graph $G = (V, E)$ and $K \subseteq V$, the minimum Steiner tree problem asks for the minimum weighted subgraph that spans K. The problem is NP-Hard. Prömel and Steger [59] showed that approximating the problem can be reduced to approximating the maximum weighted forest in a 3-uniform hypergraph. By splitting a hyperedge with three vertices to an edge pair in the form (u, v) and (v, w), the maximum weighted forest problem in a 3-uniform hypergraph can be reduced to a weighted graphic matroid parity problem, similar to approximating planar subgraph. This implies a 5/3-approximation algorithm for the minimum Steiner tree problem. In Section 3.5, we will give a pseudo-polynomial time algorithm and an FPTAS for the weighted linear matroid parity problem.

Mader's disjoint $\mathcal{S}$-path Given an undirected graph $G = (V, E)$ and let $S_1, \cdots, S_k$ be disjoint subsets of V. Let $T = S_1 \cup \cdots \cup S_k$. A path is called an $\mathcal{S}$-path if it starts and ends with vertices in S_i and S_j such that $S_i \neq S_j$, while all other internal vertices of the path are in $V \setminus T$. The disjoint $\mathcal{S}$-path problem is

to find a maximum cardinality collection of vertex disjoint $\mathcal{S}$-paths of the graph G. In the following we assume without loss of generality that the graph G is connected and each S_i is a stable set.

Here we will discuss how to reduce the $\mathcal{S}$-path problem to the linear matroid parity problem in detail, as we will use this reduction in our later algorithm for the $\mathcal{S}$-path problem. The reduction is first shown by Lovász [42], but here we follow the treatment of Schrijver ([63] page 1284).

The high level idea is to associate each edge to a 2-dimensional linear subspace, and show that the edges in a solution of the $\mathcal{S}$-path problem correspond to subspaces that are linearly independent in an appropriately defined quotient space $\mathbb{R}^{2n}/Q$, where two subspaces are linearly independent if their basis vectors are linearly independent.

Associate each edge $e = (u, w) \in E$ to a 2-dimensional linear subspace L_e of $(\mathbb{R}^2)^V$ such that

$$L_e = \left\{ x \in (\mathbb{R}^2)^V \mid x(v) = \mathbf{0} \text{ for each } v \in V \setminus \{u, w\} \text{ and } x(u) + x(w) = \mathbf{0} \right\}$$

where $x : V \to \mathbb{R}^2$ is a function that maps each vertex to a 2-dimensional vector. Let $r_1, \cdots r_k$ be k distinct 1-dimensional subspaces of $\mathbb{R}^2$. For each vertex $v \in V$, let $R_v = r_j$ if $v \in S_j$ for some j, and $R_v = \{\mathbf{0}\}$ otherwise. Define a linear subspace Q of $(\mathbb{R}^2)^V$ such that

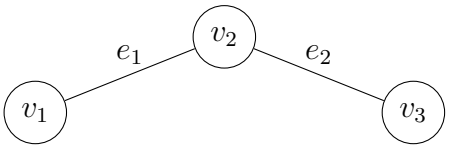
$$Q = \{x \in (\mathbb{R}^2)^V \mid x(v) \in R_v \text{ for all } v \in V\}.$$

Let $\mathcal{E}$ be the collection of subspaces L_e/Q for each $e \in E$ of $(\mathbb{R}^2)^V/Q$, where L_e/Q is the quotient space of L_e by Q. Note that $\dim(L_e/Q) = 2$ for all edges e, since it does not connect two vertices in the same S_i as we assume each S_i is a stable set. For any $F \subseteq E$, let $L_F = \{L_e/Q \mid e \in F\}$.

Lemma 2.1 ([63])**.** *L_F has dimension $2|F|$ if and only if F is a forest such that each component of (V, F) has at most two vertices in T, and at most one vertex in each S_i.*

Proof. Let X be the subspace spanned by the basis of e for all $e \in F$. Then we can

(a) Case 1: Both v_1 and v_2 belong to the same S_i.



(b) Case 2: A component containing three vertices from different S_i.

Figure 2.1: Two cases that $\dim(X \cap Q) > 0$.

check that X consists of all $x : V \to \mathbb{R}^2$ with $\sum_{v \in K} x(v) = \mathbf{0}$ for each component K of (V, F). So $\dim(X) = 2(|V| - k)$ where k is the number of components.

Also $\dim(X \cap Q) = 0$ if and only if each component has at most two vertices in common with T, and at most one with each S_i. To get an intuition why this is true, we will show that if a component has more than two vertices in T or more than one vertices in some S_i, then $\dim(X \cap Q) > 0$. To show $\dim(X \cap Q) > 0$, we assume every vertex belongs to some S_i, i.e. $V = T$. This is because for any vertex $v \notin T$, all $x \in Q$ have $x(v) = \mathbf{0}$, thus if $\dim(X \cap Q) > 0$ then v can only propagates vectors from one of its neighbors to other neighbors. Hence it remains to check the two cases in Figure 2.1. Assume v_i corresponds to the $2i - 1$ and $2i$ coordinates. For the first case, X is spanned by $(1\ 0\ \text{-}1\ 0)^T$ and $(0\ 1\ 0\ \text{-}1)^T$, whereas Q is spanned by $(a\ b\ 0\ 0)^T$ and $(0\ 0\ a\ b)^T$ for some a and b. And it is easy to check that $\dim(X \cap Q) > 0$. For the second case, basis of X and Q are

$$X : \begin{matrix} e_1 \\ \begin{pmatrix} 1 \\ 0 \\ \text{-}1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{matrix} \begin{matrix} e_1 \\ \begin{pmatrix} 0 \\ 1 \\ 0 \\ \text{-}1 \\ 0 \\ 0 \end{pmatrix} \end{matrix} \begin{matrix} e_2 \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \text{-}1 \\ 0 \end{pmatrix} \end{matrix} \begin{matrix} e_2 \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \text{-}1 \end{pmatrix} \end{matrix} \qquad Q : \begin{matrix} v_1 \\ \begin{pmatrix} a \\ b \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{matrix} \begin{matrix} v_2 \\ \begin{pmatrix} 0 \\ 0 \\ c \\ d \\ 0 \\ 0 \end{pmatrix} \end{matrix} \begin{matrix} v_3 \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ e \\ f \end{pmatrix} \end{matrix}.$$

The main idea is that after fixing the coordinates for the dimensions correspond to v_1 and v_3, X spans the subspace for v_2, thus $\dim(X \cap Q) > 0$.

Now consider

$$\dim(L_F) = \dim(X/Q) = \dim(X) - \dim(X \cap Q) \leq \dim(X) \leq 2|F|.$$

Vectors in L_F are independent if and only if $\dim(X) = 2|F|$ and $\dim(X \cap Q) = 0$. Combining with previous observations this proves the lemma. □

Theorem 2.2 ([63]). *The maximum number of disjoint $\mathcal{S}$-paths is given by $\nu(\mathcal{E}) - |V| + |T|$ where $\nu(\mathcal{E})$ is the size of a maximum collection of linearly independent 2-dimensional subspaces in $\mathcal{E}$.*

Proof. Let t be the maximum number of disjoint $\mathcal{S}$-paths, Π be a packing of t $\mathcal{S}$-paths, and F' be the set of edges contained in these paths. In F' we have $|T| - t$ components that contain some vertices in T. Now we add edges to F' so that we do not connect these $|T| - t$ components and obtain F. Since the original graph is connected, we can extend F' to F such that $|F| = |V| - (|T| - t)$. Now F satisfies the condition given in Lemma 2.1 and $|F| = t + |V| - |T|$. So $\nu(\mathcal{E}) \geq |F| = t + |V| - |T|$.

Conversely, let $\mathcal{F} \subseteq \mathcal{E}$ be a maximum collection that contains $\nu(\mathcal{E})$ linearly independent 2-dimensional subspaces. Then there exists a forest $F \subseteq E$ so that $L_F = \mathcal{F}$ and F satisfies the condition in Lemma 2.1. Let t be the number of components of (V, F) that intersect T twice. Then deleting t edges from F we obtain a forest such that each component intersects T at most once. So $|F| - t \leq |V| - |T|$ and hence $t \geq \nu(\mathcal{E}) - |V| + |T|$. □

From these proofs we see that from a linear matroid parity solution, we can construct the disjoint $\mathcal{S}$-paths simply by removing redundant edges from the linear matroid parity solution.

2.2 Matrix Formulations

In this section we present matrix formulations for graph matching and linear matroid parity. These formulations relate ranks of the appropriately defined

matrices to the optimal values of the corresponding problems. This allows us to use algebraic tools to tackle these problems.

2.2.1 Graph Matching

In this section we will show the matrix formulation for bipartite matching and general graph matching. Since the matrix formulation for the linear matroid parity problem generalize these results, we will just give a brief idea why do these matrix formulations capture the size of a maximum matching.

For a bipartite graph $G = (U, V, E)$, the Edmonds matrix A of the graph is a $|U| \times |V|$ matrix constructed as follows. For an edge $(u_i, v_j) \in E$, put $A_{i,j} = t_{i,j}$ where $t_{i,j}$ are distinct indeterminates.

Theorem 2.3 ([47]). *The rank of the Edmonds matrix of a bipartite graph G equals the size of a maximum matching in G.*

For the special case when $|U| = |V|$, it is easy to see that there is a perfect matching in G if and only if A is of full rank. Let $n = |U| = |V|$, and consider the determinant of A,

$$\det A = \sum_{\sigma \in \mathcal{S}^n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where σ runs through all permutation of $\{1, \cdots, n\}$, and $\operatorname{sgn}(\sigma)$ is $+1$ if σ has even number of inversions and -1 otherwise. Consider a non-zero term $\prod_{i=1}^{n} A_{i,\sigma(i)}$ in $\det A$. Since i and $\sigma(i)$ covers 1 to n, the product consists of exactly one entry from each row (and each column) of A. As each entry in A corresponds to an edge in G, the product corresponds to a set of edges that cover each vertex in U and V. Thus a non-zero term in $\det A$ represents a perfect matching.

For a general graph $G = (V, E)$, the Tutte matrix T of the graph is an $n \times n$ matrix constructed as follows. For an edge $(v_i, v_j) \in E$ where $i < j$, put $T_{i,j} = t_{i,j}$ and $T_{j,i} = -t_{i,j}$ where $t_{i,j}$ are distinct indeterminates.

Theorem 2.4 ([69]). *The rank of the Tutte matrix of a graph G is twice the size of a maximum matching in G.*

To see why this theorem is true we need some knowledge on skew-symmetric matrices. The Tutte matrix T is a *skew-symmetric* matrix, which means that $T_{ij} = -T_{ji}$ for all i and j. Given a skew-symmetric matrix B of even dimension $p = 2n$, for each perfect matching $P = \{\{i_1, j_1\}, \ldots, \{i_n, j_n\}\}$ of K_{2n}, let

$$b_P = \text{sgn}(i_1, j_1, \ldots, i_n, j_n) b_{i_1,j_1} \cdots b_{i_n,j_n}$$

where the sign function sgn has the same definition as in determinant. Note that b_P does not depend either on the order of the two vertices in an edge or the order of the edges. The *Pfaffian* of a skew-symmetric matrix B is defined by

$$\text{pf}(B) = \sum_P b_P.$$

Thus pf T of a graph can be interpret as a "sum" of all its perfect matching. An important property of the Pfaffian is that

$$\det(B) = (\text{pf}(B))^2,$$

which we will prove later in Section 2.2.2.

Now we go back to Theorem 2.4 and see why there is a perfect matching in G if and only if T is full rank. To see this, observe that if T is full rank, then $\det(T)$ and thus $\text{pf}(T)$ are both non-zero polynomials. Hence there is a non-zero term in $\text{pf}(T)$, which implies there is a perfect matching. To see the other direction, observe that different terms for different perfect matching in the Pfaffian cannot cancel each other because they consists of different indeterminates $t_{i,j}$.

2.2.2 Skew-Symmetric Matrix

Since most of the matrix formulations we use in this thesis are skew-symmetric, this section we introduce some properties of skew-symmetric matrices and also give the proof that the determinant of a matrix is equal to the square of the Pfaffian.

A is a *skew-symmetric* matrix if $A_{ij} = -A_{ji}$ for all i, j. If A is a skew-symmetric

matrix of odd dimension, then

$$\det(A) = \det(A^t) = \det(-A) = (-1)^n \det(A) = -\det(A),$$

and hence $\det(A) = 0$ if A is of odd dimension.

Theorem 2.5 ([29]). *For any skew-symmetric matrix, its inverse (if exists) is also skew-symmetric.*

Proof. Let M be a skew-symmetric matrix, we have $M^T = -M$. Therefore

$$(M^{-1})_{i,j} = ((M^{-1})^T)_{j,i} = ((M^T)^{-1})_{j,i} = ((-M)^{-1})_{j,i} = (-M^{-1})_{j,i} = -(M^{-1})_{j,i}.$$

This implies $(M^{-1})^T = -M^{-1}$, hence M^{-1} is skew-symmetric. □

Theorem 2.6 ([51]). *For a rank r skew-symmetric matrix M with $\operatorname{rank}(M_{R,C}) = r$ where $|R| = |C| = r$, then $\operatorname{rank}(M_{R,R}) = r$.*

Proof. Since $M_{R,C}$ has the same rank of M, rows of $M_{R,*}$ spans all rows of M. In particular, rows of $M_{R,R}$ spans $M_{*,R}$. Therefore $\operatorname{rank}(M_{R,R}) = \operatorname{rank}(M_{*,R}) = \operatorname{rank}(M_{R,*}) = r$ □

Let B be a skew-symmetric matrix of even dimension $p = 2n$. For each unique perfect matching $P = \{\{i_1, j_1\}, \ldots, \{i_n, j_n\}\}$ of K_{2n} form the expression,

$$b_P = \operatorname{sgn}(i_1, j_1, \ldots, i_n, j_n) b_{i_1,j_1} \cdots b_{i_n,j_n}$$

where the sign function sgn has the same definition as in determinant. Note that b_P does not depend either on the order of the two vertices in an edge or the order of the edges. The *Pfaffian* of a skew-symmetric matrix B is defined by

$$\operatorname{pf}(B) = \sum_P b_P.$$

An important property of the Pfaffian is that

$$\det(B) = (\operatorname{pf}(B))^2$$

or alternatively,

$$\sum_{\sigma \in \mathcal{S}^n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)} = \left(\sum_P b_P \right) \left(\sum_P b_P \right).$$

In the remaining of this section we will give a combinatorial proof of this theorem, following the treatment of Godsil [26]. Define the even-cycle cover of a graph to be a disjoint union of even cycles that covers all the vertices of the graph. The idea of the proof is that each term of the determinant corresponds to an even-cycle cover of K_{2n}. Such an even-cycle cover can be decomposed into two perfect matchings of K_{2n}, whereas a perfect matching is presented by a term in Pfaffian. Hence each term in $\det(B)$ can be factored as a product of two terms in $\operatorname{pf}(B)$.

Given a permutation σ, we can construct a cycle cover over K_{2n}. For each i in 1 to $2n$ add a directed edge $(i, \sigma(i))$. Since each vertex has exactly one incoming edge and one outgoing edge, we obtain a cycle cover. Now we claim that for a permutation σ, if we swap two adjacent vertices in a cycle represented by σ, then the sign of σ remains unchanged. Let σ' be the new permutation obtained. Also let C and C' be the cycles represented by σ and σ' respectively. Assume by swapping vertices b and c in C we obtain C'.

$$C : \cdots a \to b \to c \to d \cdots \qquad C' : \cdots a \to c \to b \to d \cdots$$

Then σ and σ' has the following form:

$$\begin{array}{rccccc} i: & \cdots & a & b & c & \cdots \\ \sigma(i): & \cdots & b & c & d & \cdots \end{array} \qquad \begin{array}{rccccc} i: & \cdots & a & b & c & \cdots \\ \sigma'(i): & \cdots & c & d & b & \cdots \end{array}.$$

We can see that σ' is obtained from σ by making two swaps. Since each swap in a permutation changes its sign, we have $\operatorname{sgn}(\sigma) = \operatorname{sgn}(\sigma')$.

Odd cycles vanish: Then we claim that terms in the determinant that contains odd cycles vanished. Hence $\det(B)$ is a sum of even-cycle covers. Firstly, since B is skew symmetric, we have $B_{i,i} = 0$, thus permutations that contains self loop ($\sigma(i) = i$) will vanish. Now consider a permutation σ that contains some

odd cycle, among all odd cycles pick the cycle that contains the vertex with the smallest index. Reverse that cycle and call the new permutation σ'. First we can use the previous claim to show $\text{sgn}(\sigma) = \text{sgn}(\sigma')$. This is because we must be able to swap adjacent vertices in the reversed cycle multiple times to obtain σ' from σ. In addition, we have

$$\prod_{i=1}^{2n} B_{i,\sigma(i)} = -\prod_{i=1}^{2n} B_{i,\sigma'(i)}$$

because odd number of $B_{i,\sigma(i)}$ get its sign changed (when we reverse an edge (i,k), $B_{i,\sigma(i)} = -B_{\sigma(i),i} = -B_{k,\sigma'(k)}$). So the terms in $\det(B)$ for σ and σ' get cancelled. In addition $(\sigma')' = \sigma$ so all permutations that contain odd cycles pair up and cancel each other.

Bijection between even-cycle covers and pairs of perfect matchings: Now we claim there is a bijection between an even-cycle covers and pairs of perfect matchings. Given a cycle

$$i_1 \to i_2 \to \cdots i_{2n} \to i_1,$$

we can construct two matchings π_1 and π_2:

$$\pi_1 = \{\{i_1, i_2\}, \{i_3, i_4\}, \cdots \{i_{2n-1}, i_{2n}\}\}$$
$$\pi_2 = \{\{i_2, i_3\}, \{i_4, i_5\}, \cdots \{i_{2n}, i_1\}\}.$$

Note that a cycle and its reversed cycle have π_1 and π_2 swapped, so any even-cycle cover maps to different pairs of matching (note the matching pair (π_1, π_2) is considered different from (π_2, π_1)). For the case that an even-cycle cover contains more than one cycle, we can construct matchings π_1 and π_2 by applying the above construction to each cycle. Also each pair of matchings are mapped to an even-cycle cover because the union of two perfect matchings becomes an even-cycle cover. Thus there is a bijection between even-cycle covers and pairs of matchings.

Signs of terms match: Now we have

$$\left|\operatorname{sgn}(\sigma)\prod_{i=1}^{2n}B_{i,\sigma(i)}\right| = \left|\left(\operatorname{sgn}(\pi_1)\prod_{i=1}^{n}B_{\pi_1(2i-1),\pi_1(2i)}\right)\cdot\left(\operatorname{sgn}(\pi_2)\prod_{i=1}^{n}B_{\pi_2(2i-1),\pi_2(2i)}\right)\right|.$$

The terms in the product match because of the way we construct π_1 and π_2 from σ, and it remains to show

$$\operatorname{sgn}(\sigma) = \operatorname{sgn}(\pi_1)\cdot\operatorname{sgn}(\pi_2).$$

First assume σ only contains one cycle, and further assume the cycle is in the form $1 \to 2 \to \cdots 2n \to 1$. Then according to the bijection we have

$$\sigma = (2, 3, 4, \cdots, 2n, 1)$$

and

$$\pi_1 = (1, 2, 3, 4, \cdots, 2n-1, 2n) \qquad \pi_2 = (2, 3, 4, 5, \cdots, 2n, 1),$$

so the signs match. Now if we swap two adjacent element in the cycle, and we obtain σ', π_1' and π_2'. As we have shown before $\operatorname{sgn}(\sigma) = \operatorname{sgn}(\sigma')$. And because π_1' and π_2' are obtained by making one swap to π_1 and π_2 respectively, we have $\operatorname{sgn}(\pi_1) = -\operatorname{sgn}(\pi_1')$ and $\operatorname{sgn}(\pi_2) = -\operatorname{sgn}(\pi_2')$. So the signs still match after swapping two adjacent element in the cycle, thus the signs match if there is only one cycle.

Then we consider the case when there are two cycles. Assume each cycle contains vertices with consecutive indices, i.e. there are in the form $1 \to 2 \to \cdots \to k \to 1$ and $k+1 \to k+2 \to \cdots \to 2n \to k+1$ where k is even. Then

$$\begin{aligned}
\sigma &= (2, 3, \cdots, k, 1, k+1, k+2, \cdots, 2n, k+1)\\
\pi_1 &= (1, 2, \cdots, k-1, k, k+1, k+2, \cdots, 2n-1, 2n)\\
\pi_2 &= (2, 3, \cdots, k, 1, k+2, k+3, \cdots, 2n, k+1)
\end{aligned}$$

and it is easy to see their signs match. If we swap vertices i and j from different cycles, it suffices to swap $\sigma(i)$ with $\sigma(j)$ and $\sigma(\sigma^{-1}(i))$ with $\sigma(\sigma^{-1}(j))$, and do

one swap each in π_1 and π_2. Hence the signs still match after swapping vertex in different cycles.

By starting with even cycle cover of the form

$$(1 \to 2 \to \cdots j \to 1)$$
$$(j+1 \to j+2 \to \cdots k \to j+1)$$
$$(k+1 \to k+2 \to \cdots 2n \to k+1),$$

it is easy to check the signs match. By swapping adjacent elements in the same cycle and swapping elements in different cycles, we can obtain any even-cycle cover and the signs still match. So we complete the proof of the following theorem.

Theorem 2.7 ([26]). *If B is a skew-symmetric matrix, then* $\det(B) = (\mathrm{pf}(B))^2$.

2.2.3 Linear Matroid Parity

There are two equivalent matrix formulations for the linear matroid parity problem. We will first show the compact formulation given by Lovász [41].

Here we are given m vector pairs $\{\{b_1, c_1\}, \ldots, \{b_m, c_m\}\}$ where each vector is of dimension r. We use ν_M to denote the optimal value of the linear matroid parity problem, which is the maximum number of pairs we can choose so that the chosen vectors are linearly independent. We call an optimal solution a *parity basis* if $\nu_M = r/2$, which is the best possible. Define the *wedge product* $b \wedge c$ of two column vectors b and c is defined as $bc^T - cb^T$.

To prove the matrix formulation of Lovász we first prove a special case. We will show that if we have $r/2$ pairs, then these pairs form a parity basis if and only if a certain matrix is of full rank.

Lemma 2.8 ([9]). *Given $m = r/2$ pairs $\{b_i, c_i\}$, and let $M = (b_1|c_1|\cdots|b_m|c_m)$ be an $r \times r$ matrix. Then*

$$\mathrm{pf}\left(\sum_{i=1}^{m} x_i(b_i \wedge c_i)\right) = \pm \det(M) \prod_{i=1}^{m} x_i$$

where x_i are indeterminates.

Proof. The left hand side of the equation can be written as

$$\mathrm{pf}\left(\sum_{i=1}^{m} x_i(b_i \wedge c_i)\right) = \mathrm{pf}\left(\sum_{i=1}^{m} x_i(M\vec{e}_{2j-1} \wedge M\vec{e}_{2j})\right).$$

Note for any matrix M and vectors u and v,

$$Mu \wedge Mv = Muv^T M^T - Mvu^T M^T = M(uv^T - vu^T)M^T = M(u \wedge v)M^T,$$

so the expression for the Pfaffian can be written as

$$\mathrm{pf}\left(\sum_{i=1}^{m} x_i M(\vec{e}_{2j-1} \wedge \vec{e}_{2j})M^T\right) = \mathrm{pf}\left(M(\sum_{i=1}^{m} x_i(\vec{e}_{2j-1} \wedge \vec{e}_{2j}))M^T\right).$$

Let $A = \sum_{i=1}^{m} x_i(\vec{e}_{2j-1} \wedge \vec{e}_{2j})$, then

$$\mathrm{pf}(MAM^T) = \pm\sqrt{\det(MAM^T)} = \pm\sqrt{\det(M)^2 \det(A)} = \pm \det(M)\,\mathrm{pf}(A),$$

and

$$\mathrm{pf}(A) = \prod_{i=1}^{m} x_i.$$

Combining the above two expressions we complete the proof. □

Theorem 2.9 (Lovász [41])**.** *Given m column pairs $\{(b_i, c_i)\}$ for $1 \le i \le m$ and $b_i, c_i \in \mathbb{R}^r$. Let*

$$Y = \sum_{i=1}^{m} x_i(b_i \wedge c_i),$$

where x_i are indeterminates. Then $2\nu_M = \mathrm{rank}(Y)$.

Proof. We first show the case that there is parity basis if and only if Y is of full rank. Assume r is even, otherwise there is no parity basis. By the definition of Pfaffian, all the terms in $\mathrm{pf}(Y)$ are of degree $r/2$ in x_i (another way to see this: $\det(Y)$ is of degree r, and $\mathrm{pf}(Y)^2 = \det(Y)$). To see the coefficient of a term

$x_{i_1} \cdots x_{i_{r/2}}$, we can ignore other x_j, thus set $x_j = 0$ for $j \notin \{i_1, \cdots, i_{r/2}\}$. So it suffices to consider

$$\mathrm{pf}\left(\sum_{j \in \{i_1 \cdots i_{r/2}\}} x_j (b_j \wedge c_j)\right),$$

which is

$$\pm \det(b_{i_1}|c_{i_1}| \cdots |b_{i_{r/2}}|c_{i_{r/2}}) \prod_{j \in \{i_1 \cdots i_{r/2}\}} x_j$$

by Lemma 2.8. Now we can observe that the determinant is non-zero if only if $\{(b_1, c_1) \cdots (b_{r/2}, c_{r/2})\}$ forms a parity basis. Hence a term only survives if its corresponding columns form a parity basis. So we have

$$\mathrm{pf}(Y) = \sum_{\text{Parity Basis } J} \pm \det(b_{j_1}|c_{j_1}| \cdots |b_{j_{r/2}}|c_{j_{r/2}}) \prod_{j \in J} x_j. \tag{2.2.1}$$

Thus we can conclude there is a parity basis if and only if Y is of full rank.

Now we use these ideas to prove the general cases. We first prove $2\nu_M \leq \mathrm{rank}(Y)$. Let R be the rows span by the ν_M pairs. If we just consider these rows, then by the construction of Y we see $Y_{R,R}$ is of full rank. To see $\mathrm{rank}(Y) \leq 2\nu_M$, first note that by Theorem 2.6, there is a principal submatrix $Y_{R,R}$ that is of full rank and $\mathrm{rank}(Y_{R,R}) = \mathrm{rank}(Y)$. Note that $Y_{R,R}$ corresponds to the matrix formulation of the parity problem with column pairs restricted to the rows R. Thus $\mathrm{rank}(Y) = \mathrm{rank}(Y_{R,R}) \leq 2\nu_M^* \leq 2\nu_M$, where is ν_M^* is the maximum number of pairs we can pick if we restrict to rows R. □

Another formulation is a sparse matrix formulation given by Geelen and Iwata [25]. Let T be a matrix with size $2m \times 2m$, so that indeterminate t_i appears in $T_{2i-1,2i}$ and $-t_i$ appears in $T_{2i,2i-1}$ for $i \in [1, m]$ while all other entries of T are zero. Also let M be an $r \times 2m$ matrix for the linear matroid parity problem.

Theorem 2.10 (Geelen and Iwata [25])**.** *Let*

$$Z = \begin{pmatrix} 0 & M \\ -M^T & T \end{pmatrix}$$

where $M = (b_1|c_1|b_2|c_2| \cdots |b_m|c_m)$*. Then* $2\nu_M = \mathrm{rank}(Z) - 2m$*.*

Proof. We will use Theorem 2.9 to prove this theorem. Consider the matrix Z. Using T to eliminate M, we have

$$Z' = \begin{pmatrix} MT^{-1}M^T & O \\ -T^{-1}M^T & I_{2m} \end{pmatrix}.$$

So $\operatorname{rank}(Z) = \operatorname{rank}(Z') = \operatorname{rank}(MT^{-1}M^T) + 2m$. To prove the theorem it suffices to show

$$2\nu_M = \operatorname{rank}(Z) - 2m = \operatorname{rank}(MT^{-1}M^T).$$

Therefore it remains to show $\operatorname{rank}(MT^{-1}M^T) = \operatorname{rank}(Y)$ where Y is the matrix formulation defined in Theorem 2.9, and the result follows. Let $P_i = \begin{pmatrix} b_i & c_i \end{pmatrix}$ be a $2r \times 2$ matrix, $Q_i = \begin{pmatrix} 0 & t_i \\ -t_i & 0 \end{pmatrix}$. Then

$$\begin{aligned}
MT^{-1}M^T &= \begin{pmatrix} P_1 & P_2 & \cdots P_m \end{pmatrix} \begin{pmatrix} Q_1^{-1} & O & \cdots & O \\ O & Q_2^{-1} & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & Q_n^{-1} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ \vdots \\ P_m \end{pmatrix} \\
&= \begin{pmatrix} P_1Q_1^{-1} & P_2Q_2^{-1} & \cdots P_mQ_m^{-1} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ \vdots \\ P_m \end{pmatrix} \\
&= \sum_{i=1}^{m} P_iQ_i^{-1}P_i^T \\
&= \sum_{i=1}^{m} \begin{pmatrix} b_i & c_i \end{pmatrix} \begin{pmatrix} 0 & t_i \\ -t_i & 0 \end{pmatrix}^{-1} \begin{pmatrix} b_i^T \\ c_i^T \end{pmatrix} \\
&= \sum_{i=1}^{m} \frac{1}{t_i} \begin{pmatrix} c_i & -b_i \end{pmatrix} \begin{pmatrix} b_i^T \\ c_i^T \end{pmatrix} \\
&= \sum_{i=1}^{m} \frac{1}{t_i} (c_ib_i^T - b_ic_i^T)
\end{aligned}$$

$$= \sum_{i=1}^{m} x_i(b_i \wedge c_i)$$
$$= Y$$

where $x_i = -1/t_i$. □

2.2.4 Weighted Problems

The algebraic approach can be extended to weight problems. The general idea is to introduce a variable y, and put the weight w to the exponent of y. Take the bipartite matching problem as an example, for an edge (u_i, v_j) with weight $w_{i,j}$, put $t_{i,j}y^{w_{i,j}}$ in the (i,j)-th entry to the matrix A. Then the determinant of A is in the form

$$\sum_{\sigma \in \mathcal{S}^n} \operatorname{sgn}(\sigma) \left(\prod_{i=1}^{n} t_{i,\sigma(i)} \right) \left(\prod_{i=1}^{n} y^{w_{i,\sigma(i)}} \right).$$

Thus the total weight of all edges in a perfect matching sums up in the exponent of y. And it suffices to compute $\det A$ and look at the highest degree of y in $\det A$ to determine the weight of a maximum weighted perfect matching.

We now do the same for the weight version of linear matroid parity, where each column pair is assigned a weight. The matrix formulation is almost the same as the formulation for the unweighted case in Theorem 2.9. The only exception is that all indeterminates x_i are now replaced by $x_i y^{w_i}$.

Theorem 2.11 (Camerini, Galbiati, Maffioli [9])**.** *Let*

$$Y^* = \sum_{i=1}^{m} x_i(b_i \wedge c_i)y^{w_i},$$

where the pairs $\{(b_i, c_i)\}$ *compose* M*,* x_i *and* y *are indeterminates. Then the degree of* y *in* $\det Y^*$ *is twice the weight of a maximum weighted parity basis.*

Proof. Recall Equation 2.2.1 in the proof of Theorem 2.9.

$$\operatorname{pf}(Y) = \sum_{\text{Parity Basis } J} \det(b_{j_1}|c_{j_1}|\cdots|b_{j_{r/2}}|c_{j_{r/2}}) \prod_{j \in J} x_j.$$

Thus for the weight formulation Y^* we have

$$\mathrm{pf}(Y^*) = \sum_{\text{Parity Basis } J} \det(b_{j_1}|c_{j_1}|\cdots|b_{j_{r/2}}|c_{j_{r/2}}) \left(\prod_{j\in J} x_j\right)\left(\prod_{j\in J} y^{w_j}\right).$$

So the highest degree of y in $\mathrm{pf}(Y^*)$ gives the weight of a maximum weighted parity basis. Then the theorem follows from the fact $\mathrm{pf}(Y^*)^2 = \det(Y^*)$. □

2.3 Algebraic Tools

To use matrix formulations in the previous section to design fast algorithms, we need efficient ways to manipulate a matrix and also to compute its determinant and its rank. In this section we will introduce the algebraic tools that we will use.

2.3.1 Matrix Algorithms

One of the most powerful tools we have in linear algebra is to compute product of two $n \times n$ matrices in $O(n^\omega)$ time where $\omega < 2.38$ [14]. It is known that this implies we can compute LUP decomposition of an $n \times n$ matrix in $O(n^\omega)$ time [7, 29]. Thus for an $n \times n$ matrix M, all the of following can be computed in $O(n^\omega)$ time:

- The determinant of M
- The rank of M
- The inverse of M
- A maximum rank submatrix of M, which is a non-singular submatrix $M_{R,C}$ of M so that $|R| = |C| = \operatorname{rank} M$.

For multiplication between non-square matrices, one can divide each matrix to blocks of square matrix. Then we can apply fast matrix multiplication on these

square matrices to obtain the product of the original matrices. Similar technique can also be applied to other matrix operations as below. These will help us to design fast algorithms in later chapters.

Theorem 2.12. *Given an $m \times n$ matrix M where $m \geq n$, the rank and a maximum rank submatrix of M can be founded in $O(mn^{\omega-1})$ time.*

Proof. Partition rows of M to m/n groups $R_1, R_2, \cdots, R_{m/n}$ so that each group contains n rows. We will use row set R to keep the set of rows that span M. We begin with empty set R. For each R_i we compute a maximum rank submatrix $M_{S,T}$ of the matrix $M_{R \cup R_i,*}$, and assign S to R. Then the final row set R is the rows that span M, and $|R|$ gives the rank of M. To see the time complexity of this algorithm, notice that we have the invariant $|R| \leq n$ because $\text{rank}(M) \leq n$. Thus $M_{R \cup R_i,*}$ is an $O(n) \times n$ matrix, by treating it as a square matrix its maximum rank submatrix can be computed time $O(n^\omega)$ time. Hence the time complexity to find the rank of M is $O(n^\omega \cdot m/n) = O(mn^{\omega-1})$. To find a maximum rank submatrix of M it suffices to a maximum rank submatrix of $M_{R,*}$, which can be done in the same time above. □

Theorem 2.13. *Given a size $n \times n$ matrix M that has rank $r \leq k$, it can be written as $M = UV^T$ where U and V are $n \times r$ matrix in $O(n^2 k^{\omega-2})$ time.*

Proof. First we show that a maximum rank submatrix of M can be founded in $O(n^2 k^{\omega-2})$ time. Then we will show that we can use the maximum rank submatrix to decompose M to UV^T.

To find a maximum rank submatrix of M, we use a similar approach as in Theorem 2.12. We partition rows of M to n/k groups $R_1, R_2, \cdots, R_{n/k}$ so that each group contains k rows. Again we use R to keep the set of rows that span M, beginning with R as an empty set. For each R_i we compute a maximum rank submatrix $M_{S,T}$ of $M_{R \cup R_i,*}$ (size $O(k) \times n$) in $O(nk^{\omega-1})$ time using Theorem 2.12, then we assign S to R. After these we obtain a row set R that spans M where $|R| \leq k$. Hence a maximum rank submatrix of $M_{R,*}$ is a maximum rank submatrix of M, and by Theorem 2.12 this can be computed in $O(nk^{\omega-1})$ time. Finally the time taken to compute a maximum rank submatrix of M is $O(nk^{\omega-1} \cdot n/k) = O(n^2 k^{\omega-2})$.

Now we use the maximum rank submatrix of $M_{R,C}$ to decompose M to UV^T. To do this it suffices to put $U = M_{*,C}$ which contains columns that span M, and it remains to express other columns of M as linear combinations of columns in U. This can be easily achieved by writing $V = (M_{R,C})^{-1}M_{R,*}$, which takes $O(k^\omega \cdot n/k) = O(nk^{\omega-1})$ to compute. □

2.3.2 Computing Matrix Inverse

The inverse of a matrix plays a key role in designing fast algebraic algorithms. For example, a single entry of the inverse tells us whether a particular edge should be included in a perfect matching. Thus we need efficient ways to compute and update the inverse of a matrix. In this section we introduce three tools to help us to do so.

Given a matrix M, considering its Schur complement gives us a way to compute its inverse. Let

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix},$$

where A and D are square matrices. If A is nonsingular, then $S = D - CA^{-1}B$ is called the Schur complement of A.

Theorem 2.14 (Schur's formula [71] (Theorem 1.1))**.** *Let M and A, B, C, D be matrices as defined above. If A is non-singular then* $\det(M) = \det(A) \times \det(S)$. *Furthermore if A and S are non-singular then*

$$M^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}.$$

Proof. The Schur's formula can be derived by doing block Gaussian elimination. First we make the top-left corner to be identity by multiplying M by an elementary block matrix:

$$\begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ C & D \end{pmatrix}.$$

Then we eliminate C by another elementary block matrix:

$$\begin{pmatrix} I & 0 \\ -C & I \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ C & D \end{pmatrix} = \begin{pmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{pmatrix}.$$

Therefore,

$$\begin{pmatrix} I & 0 \\ -C & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & D - CA^{-1}B \end{pmatrix}$$

By taking determinant on both sides, we have $\det(A^{-1}) \cdot \det(M) = \det(D - CA^{-1}B)$, which implies $\det(M) = \det(A) \times \det(S)$.

We now continue to transform M into the identity matrix. Since $S = D - CA^{-1}B$ is non-singular, we can eliminate the bottom-right block by multiplying an elementary block matrix:

$$\begin{pmatrix} I & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & S \end{pmatrix} = \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}.$$

Finally we eliminate $A^{-1}B$ by multiplying another elementary block matrix:

$$\begin{pmatrix} I & -A^{-1}B \\ 0 & I \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}.$$

Therefore we have:

$$\begin{pmatrix} I & -A^{-1}B \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -C & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}.$$

Multiplying the first four matrices gives us M^{-1} as in the statement. □

Suppose we have a matrix M and its inverse M^{-1}. If we perform a small rank update on M, the following formula [70] shows how to update M^{-1} efficiently.

Theorem 2.15 (Sherman-Morrison-Woodbury [70])**.** *Let M be an $n \times n$ matrix, U be an $n \times k$ matrix, and V be an $n \times k$ matrix. Suppose that M is non-singular. Then*

1. *$M + UV^T$ is non-singular if and only if $I + V^T M^{-1} U$ is non-singular.*

2. *If $M + UV^T$ is non-singular, then $(M + UV^T)^{-1} = M^{-1} - M^{-1}U(I + V^T M^{-1} U)^{-1} V^T M^{-1}$.*

Proof. The matrix $M + UV^T$ arises when we perform Gaussian elimination:

$$\begin{pmatrix} I & 0 \\ U & I \end{pmatrix} \begin{pmatrix} I & V^T \\ -U & M \end{pmatrix} = \begin{pmatrix} I & V^T \\ 0 & M + UV^T \end{pmatrix},$$

and so the original matrix is non-singular if and only if $M + UV^T$ is non-singular. By doing Gaussian elimination in another way, we obtain:

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

Combining we have

$$\begin{pmatrix} I & 0 \\ U & I \end{pmatrix} \begin{pmatrix} I & V^T M^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} I + V^T M^{-1} U & 0 \\ -U & M \end{pmatrix} = \begin{pmatrix} I & V^T \\ 0 & M + UV^T \end{pmatrix}. \tag{2.3.2}$$

Taking determinant on both sides, we have

$$\det(I + V^T M^{-1} U) \cdot \det(M) = \det(M + UV^T)$$

As M is non-singular, $\det(M) \neq 0$ and the first statement follows. To prove the second statement, we observe $(M + UV^T)^{-1}$ must appear in the bottom-right corner of the inverse of the right hand side of (2.3.2). Hence $(M + UV^T)^{-1}$ is equal to the bottom-right corner of

$$\begin{pmatrix} I + V^T M^{-1} U & 0 \\ -U & M \end{pmatrix}^{-1} \begin{pmatrix} I & V^T M^{-1} \\ 0 & I \end{pmatrix}^{-1} \begin{pmatrix} I & 0 \\ U & I \end{pmatrix}^{-1} \tag{2.3.3}$$

The product of the last two term of (2.3.3) is equal to

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

The first term of (2.3.3) can be written as a product of elementary block matrices:

$$\begin{aligned}\begin{pmatrix} I + V^T M^{-1} U & 0 \\ -U & M \end{pmatrix}^{-1} &= \begin{pmatrix} I & 0 \\ M^{-1}U & I \end{pmatrix} \begin{pmatrix} (I + V^T M^{-1} U)^{-1} & 0 \\ 0 & M^{-1} \end{pmatrix} \\ &= \begin{pmatrix} (I + V^T M^{-1} U)^{-1} & 0 \\ M^{-1}U(I + V^T M^{-1} U)^{-1} & M^{-1} \end{pmatrix}\end{aligned}$$

Now $(M+UV^T)^{-1}$ is equal to the bottom-right corner of the product of the above two matrices, which is equal to

$$-M^{-1}U(I + V^T M^{-1} U)^{-1} V^T M^{-1} + M^{-1}. \qquad \square$$

Alternatively, if we update $M_{S,S}$ for small $|S|$, then Harvey [30] showed that the Sherman-Morrison-Woodbury formula can be used to compute the values in ${M^{-1}}_{T,T}$ quickly for small $|T|$.

Theorem 2.16 (Harvey [30])**.** *Let M be a non-singular matrix and let $N = M^{-1}$. Let $\tilde{M}$ be a matrix which is identical to M except $\tilde{M}_{S,S} \neq M_{S,S}$ and let $\Delta = \tilde{M} - M$.*

1. *$\tilde{M}$ is non-singular if and only if $\det(I + \Delta_{S,S} N_{S,S}) \neq 0$.*
2. *If $\tilde{M}$ is non-singular then $\tilde{M}^{-1} = N - N_{*,S}(I + \Delta_{S,S} N_{S,S})^{-1} \Delta_{S,S} N_{S,*}$.*
3. *Restricting $\tilde{M}^{-1}$ to a subset T, we have*

$${\tilde{M}^{-1}}_{T,T} = N_{T,T} - N_{T,S}(I + \Delta_{S,S} N_{S,S})^{-1} \Delta_{S,S} N_{S,T},$$

which can be computed in $O(|T|^\omega)$ time for $|T| \geq |S|$.

Proof. In this proof we shall make use of Theorem 2.15 by putting $\tilde{M} = M + UV^T$. Let $\Delta_{S,S} = \tilde{M}_{S,S} - M_{S,S}$. Then, we can substitute $\tilde{M}$, M, U and V as follows:

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

$$U = \begin{array}{c} \\ S \\ \overline{S} \end{array} \begin{array}{c} S \\ \begin{pmatrix} I \\ 0 \end{pmatrix} \end{array} \qquad V = \begin{array}{c} S \quad \overline{S} \\ S \begin{pmatrix} \Delta_{S,S} & 0 \end{pmatrix} \end{array}$$

Then, with this construction and using Theorem 2.15, $\tilde{M}$ is non-singular if and only if $I + V^T M^{-1} U$ is non-singular. Using the definition of $M^{-1} = N$, U and V, we have

$$I + V^T M^{-1} U = I + V^T N U = I + V^T N_{*,S} = I + \Delta_{S,S} N_{S,S}$$

Therefore $\det(I + \Delta_{S,S} N_{S,S}) \neq 0$, and we prove the first part of the theorem.

The second part of the theorem also follows from Theorem 2.15, if $\tilde{M} = M + UV^T$ is non-singular, then

$$\begin{aligned} \tilde{M}^{-1} &= (M + UV^T)^{-1} \\ &= M^{-1} - M^{-1}U(I + V^T M^{-1} U)^{-1} V^T M^{-1} \\ &= N - NU(I + V^T NU)^{-1} V^T N \\ &= N - N_{*,S}(I + \Delta_{S,S} N_{S,S})^{-1} \Delta_{S,S} N_{S,*} \end{aligned}$$

The last part of the theorem follows from the second part. $I + \Delta_{S,S} N_{S,S}$ has size $|S| \times |S|$ and its inverse can be computed in $O(|S|^\omega)$ time. The remaining computations involve matrix with size no more than $|T| \times |T|$, hence can be done in $O(|T|^\omega)$ time. □

2.3.3 Matrix of Indeterminates

All the matrix formulations we introduced involve indeterminates. However, it is hard to deal with such matrices. The approach to tackle this problem suggested by Lovász [41] is to substitute these indeterminates with random values. In this section we analyze the probability that the rank of a matrix is altered by the substitution.

Let $\mathbb{F}$ be a field, and let $\mathbb{F}(x_1, \ldots, x_m)$ be the field of rational function over $\mathbb{F}$ with

indeterminates $\{x_1, x_2, \ldots, x_m\}$. A matrix with entries in $\mathbb{F}(x_1, \ldots, x_m)$ is called a matrix of indeterminates. A matrix M of indeterminates is non-singular if and only if its determinant is not the zero function. To check if an $n \times n$ matrix M with indeterminates is non-singular, one can substitute each x_i with a random value in $\mathbb{F}$ and call the resulting matrix M'. Schwartz-Zippel Lemma bounds the probability that M is non-singular but M' becomes singular.

Theorem 2.17 (Schwartz-Zippel Lemma [47])**.** *Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Let S be a finite subset of $\mathbb{F}$ and let $r_1, r_2, \ldots, r_n$ be selected randomly from S. Then*

$$\Pr[P(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Proof. We will prove by induction on the number of variables. For the base case $n = 1$, $P(x)$ is a degree d polynomial with at most d roots. So there are only d ways to pick r to make $P(r)$ zero. Hence $Pr[P(r) = 0] \leq \frac{d}{|S|}$.

Now assume the theorem holds for all polynomials with $n - 1$ variables where $n > 1$. Now we group terms of P by its power of x_1, so we can write P as

$$P(x_1, ..., x_n) = \sum_{i=0}^{d} x_1^i P_i(x_2, ..., x_n)$$

where P_i is of degree at most $d - i$. If $P \not\equiv 0$ then there exists i such that $P_i(x_2, ..., x_n) \not\equiv 0$. Consider the largest such i. Randomly pick $r_2, ..., r_n \in S$. By the induction hypothesis, $Pr[P_i(r_2, ..., r_n) = 0] \leq \frac{d-i}{|S|}$ since P_i has degree $d - i$. If $P_i(r_2, ..., r_n) \neq 0$, then $P(x_1, r_2, ..., r_n)$ is now of degree i with one variable. Hence $Pr[P(r_1, r_2, ..., r_n) = 0 | P_i(r_2, ..., r_n) \neq 0] \leq \frac{i}{|S|}$. Finally,

$$\begin{aligned}
&Pr[P(r_1, r_2, ..., r_n) = 0] \\
&= Pr[P(r_1, ..., r_n) = 0 | P_i(r_2, ..., r_n) = 0] + Pr[P(r_1, ..., r_n) = 0 | P_i(r_2, ..., r_n) \neq 0] \\
&\leq Pr[P_i(r_2, ..., r_n) = 0] + Pr[P(r_1, r_2, ..., r_n) = 0 | P_i(r_2, ..., r_n) \neq 0] \\
&\leq \frac{d-i}{|S|} + \frac{i}{|S|} = \frac{d}{|S|}
\end{aligned}$$

□

Since $\det(M)$ is a degree n polynomial, by the lemma, if M is non-singular then M' is singular with probability at most $n/|\mathbb{F}|$. Hence, by setting $q = n^c$ for a large constant c, this gives a randomized algorithm with running time $O(n^\omega)$ to test if M is non-singular with high probability.

2.3.4 Mixed Skew-symmetric Matrix

Given two skew-symmetric matrix Q and T, where entries in Q are values from a field $\mathbb{F}$ and entries in T are distinct indeterminates except $T_{i,j} = -T_{j,i}$, then the matrix $A = Q + T$ is called a mixed skew-symmetric matrix. In addition we require if $Q_{i,j}$ is non-zero then $T_{i,j}$ is zero. Some of the matrix formulations we will use later in this thesis are mixed skew-symmetric matrix, so we discuss some of its properties here.

Lemma 2.18. *A mixed skew-symmetric matrix $A = Q + T$ is nonsingular if and only if both $Q_{I,I}$ and $T_{V-I,V-I}$ are nonsingular for some $I \subseteq V$.*

Proof. First we show $\mathrm{pf}(A)$ can be written as

$$\mathrm{pf}(A) = \sum_{I \subseteq V} \pm \mathrm{pf}(Q_{I,I}) \cdot \mathrm{pf}(T_{V-I,V-I}). \tag{2.3.4}$$

Recall that each term of the Pfaffian corresponds to a perfect matching. Then $\mathrm{pf}(A)$ corresponds to all perfect matchings that use some edges(entries) from Q and some edges from T. Now consider a vertex set $I \subseteq V$, $\mathrm{pf}(Q_{I,I}) \cdot \mathrm{pf}(T_{V-I,V-I})$ corresponds to all perfect matching on V such that edges represented by Q (entries in Q) match all vertices in I and edges represented by T match all vertices in $V - I$. Summing over all possible I, all possible partitions of perfect matchings are considered. Thus we prove Equation 2.3.4.

Using the equation, $\mathrm{pf}(T_{V-I,V-I})$ from different $I \subseteq V$ cannot cancel each other because each contains a different set of indeterminates from T. Hence $\mathrm{pf}(A) \neq 0$ if and only if both $\mathrm{pf}\, Q_{I,I}$ and $\mathrm{pf}\, T_{V-I,V-I}$ are non-zero for some I. □

Lemma 2.19 ([51] (Theorem 7.3.22))**.** *For a mixed skew-symmetric matrix A, we have*

$$\operatorname{rank}(A) = \max_{I \subseteq V} \{\operatorname{rank}(Q_{I,I}) + \operatorname{rank}(T_{V-I,V-I})\}$$

Proof. Denote a maximum rank submatrix of A to be $A_{S,S}$. Now we apply Lemma 2.18 to $A_{S,S}$, there exists index set I so that

$$\operatorname{rank}(A_{S,S}) = \operatorname{rank}(Q_{I,I}) + \operatorname{rank}(T_{S-I,S-I}).$$

Hence

$$\begin{aligned} \operatorname{rank}(A) &= \operatorname{rank}(A_{S,S}) \\ &= \operatorname{rank}(Q_{I,I}) + \operatorname{rank}(T_{S-I,S-I}) \\ &\leq \operatorname{rank}(Q_{I,I}) + \operatorname{rank}(T_{V-I,V-I}) \leq \operatorname{rank}(A). \end{aligned}$$

So there exists an I such that

$$\operatorname{rank}(Q_{I,I}) + \operatorname{rank}(T_{V-I,V-I}) = \operatorname{rank}(A).$$

□

2.4 Algebraic Algorithms for Graph Matching

In this section we briefly describe the history and the key ideas for algebraic algorithms for graph matchings.

Recall the Tutte matrix T from Theorem 2.4. The rank of the $n \times n$ matrix gives the size of the maximum matching. This formulation does not have direct algorithmic implications because the rank of a matrix with indeterminates is hard to compute. To determine the size of a maximum matching, Lovász [41] first observed that by substituting indeterminates in T by random values, we have a good chance that the rank of T remains the same. This is because the determinant of T is a degree n polynomial, and thus we can apply the Schwartz-Zippel Lemma. As a result, we have an $O(n^\omega)$ time algorithm to determine the

size of the maximum matching.

The question to ask next is whether we can construct a maximum matching in $O(n^\omega)$ time. The above algorithm implies a straightforward $O(m \cdot n^\omega) = O(n^{\omega+2})$ time algorithm to find a maximum matching. We try to match each edge, throw away adjacent edges and then check if the rank of T remains unchanged. If the rank does not drop then we should add such an edge in the maximum matching.

In the next two section we will see the ideas used to develop an $O(n^\omega)$ time algorithm for constructing a perfect matching. Notice that we can reduce the problem of finding a maximum matching to finding a perfect matching as follows. By computing $k = \text{rank}(T)$ we have the size of the maximum matching, then we add $n - 2k$ dummy vertices and dummy edges that connects these dummy vertices to all original vertices. And it is easy to see that a perfect matching in the new graph corresponds to a maximum matching in the original graph.

2.4.1 Matching in $O(n^{\omega+1})$ time

For the bipartite matching problem one can do better than $O(n^{\omega+2})$ by observing that $(A^{-1})_{i,j} = C_{j,i}/\det(A)$ where $C_{j,i}$ denotes the (j,i)-th cofactor of A. Also note that $C_{j,i} = \pm\det(A_{V-j,V-i})$ which is non-zero if and only if there is a perfect matching after removing vertices u_j and v_i. Therefore, to check whether u_j and v_i is contained in a perfect matching, we just need to check whether $(A^{-1})_{i,j}$ is non-zero. So we can do one matrix inverse to identify all the edges which are contained in some perfect matching. By including one such edge into the matching, removing other edges that are adjacent to the matched edges, and then recompute the matrix inverse we get an $O(n \cdot n^\omega)$ time algorithm to construct a perfect matching in bipartite graphs.

Rabin and Vazirani [60] observed that one can do the same for general matching. The difference is that we need to compute $\det(T_{V-\{i,j\},V-\{i,j\}})$ in order to determine whether there is a perfect matching after removing vertices v_i and v_j. Rabin and Vazirani showed that checking this condition is equivalent to check whether $(T^{-1})_{i,j} \neq 0$ and thus the information can be computed by doing one matrix inverse. We now follow Harvey's argument to prove Rabin-Vazirani's claim. First

we need the Jacobi Theorem [33].

Lemma 2.20 (Jacobi Theorem ([33] page 21))**.** *Let M be a non-singular matrix. For any equal-sized sets $I, J \subseteq V$,*

$$\det(M_{I,J}) = \pm \det(M) \cdot \det\left((M^{-1})_{V-J,V-I}\right).$$

Proof. Since permuting rows and columns of matrix only changes the sign of its determinants, we assume $M_{I,J}$ is at the bottom-right corner. Let

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad \text{and} \quad M^{-1} = \begin{pmatrix} A^* & B^* \\ C^* & D^* \end{pmatrix}.$$

Notice that

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} A^* & 0 \\ C^* & I \end{pmatrix} = \begin{pmatrix} I & B \\ 0 & D \end{pmatrix}.$$

Therefore we have $\det(M) \cdot \det(A^*) = \det(D)$. Substitute $A^* = {M^{-1}}_{V-J,V-I}$ and $D = M_{I,J}$. Then the lemma follows. □

By the theorem we have

$$\begin{aligned} \det(T_{V-\{i,j\},V-\{i,j\}}) &= \pm \det(T) \cdot \det\left((T^{-1})_{\{i,j\},\{i,j\}}\right) \\ &= \pm \det(T) \cdot \left((T^{-1})_{i,j}\right)^2, \end{aligned}$$

where the last equality follows from the fact that the inverse of a skew-symmetric matrix is also skew-symmetric (Theorem 2.5). Hence the edge (v_i, v_j) is contained in a perfect matching if and only if $(T^{-1})_{i,j} \neq 0$.

2.4.2 Matching in $O(n^3)$ time

To obtain a $O(n^3)$ algorithm, observe that each time when we decide to match an edge, we remove all the adjacent edges, which only affect two rows and two columns of T. This is a rank two update to T. Using Woodbury's formula

(Theorem 2.15), after a rank two update to T, we can update T^{-1} in $O(n^2)$ time instead of recomputing it from scratch. Since we will match n edges, and each time after an edge is matched we spend $O(n^2)$ time to update the inverse. Then the inverse allows us to decide if an edge should be matched in $O(1)$ time. Thus it takes $O(n^3)$ time overall.

2.4.3 Matching in $O(n^\omega)$ time

We see that to decide if an edge should be included into a matching, it suffices to look at an entry in the inverse. So instead of recomputing the whole matrix inverse as in Rabin-Vazirani, we can only compute the inverse entries required in the future few steps. Mucha and Sankowski [48] were the first to make this observation and gave an algorithm to construct a maximum matching in $O(n^\omega)$ time. But their approach is quite complicated. We will follow Harvey's [30] approach instead, which is mainly based on Woodbury's formula, which says that after a low-rank update the inverse can be updated quickly.

The approach here is to remove all redundant edges so that the remaining edges form a perfect matching. We remove an edge, and then check if the Tutte matrix remains full rank. Notice that removing an edge only changes two entries in the matrix. Harvey derived Theorem 2.16 from Woodbury's formula, which allows us to determine if an edge is removable quickly, and also to recompute portion of the inverse quickly after modifying a small number of entries.

Here we describe the algorithm in depth since our algorithms on graphs later in Chapter 3 are based on this matching algorithm. Define function $\textsc{Remove}(U, W)$ to remove all redundant edges between vertex set U and W, so that the remaining edges belong to a perfect matching. To achieve this, we divide U into two halves U_1 and U_2, and also divide W into two halves W_1 and W_2. So we can solve $\textsc{Remove}(U, W)$ by solving four subproblems $\textsc{Remove}(U_i, W_j)$ for $i, j \in \{1, 2\}$. After involving $\textsc{Remove}(U_1, W_1)$, the removability of other edges is affected. Let $S = U \cup W$. To prepare for the next subproblem $\textsc{Remove}(U_1, W_2)$, we update $(T^{-1})_{S,S}$ using Theorem 2.16, which takes $O(|S|^\omega)$. After $\textsc{Remove}(U_1, W_2)$, we update $(T^{-1})_{S,S}$ and proceed to $\textsc{Remove}(U_2, W_1)$, and so on. For the base case

where $|U| = |W| = 1$, we have only one edge to examine, and the removability of the edge can be determined by looking at one entry in the inverse. To remove all redundant edges in $G = (V, E)$ we can simply start with $\textsc{Remove}(V, V)$. Now we analyze the running time of this algorithm, let $f(n)$ be the time needed by $\textsc{Remove}(U, W)$ where $n = |U| = |W|$. Then we have the recurrence relation

$$f(n) = 4 \cdot f(n/2) + O(n^\omega).$$

Thus by the master theorem [15] this algorithm takes $O(n^\omega)$ time.

2.4.4 Weighted Algorithms

The basic idea for the weighted algorithms is to add a variable x and store the weight of an edge on the degree of x, i.e. put $t_{ij}x^{w_{ij}}$ in the Tutte matrix if there is an edge between vertices i and j. Now the determinant of the matrix is a polynomial in x, and its maximum degree is twice the weight of a maximum weighted perfect matching. If we use Gaussian elimination to compute the determinant, the intermediate degree in the entries may go up exponentially. Another approach is to interpolate the determinant by substituting x with different values. Let W be the maximum weight of an edge. Since the degree of the determinant is at most nW, we have to evaluate the determinant nW times. Storjohann showed a way to compute this much more efficiently.

Theorem 2.21 (Storjohann [65])**.** *Let $A \in \mathbb{F}[x]^{n\times n}$ be a polynomial matrix of degree d and $b \in \mathbb{F}[x]^{n\times 1}$ be a polynomial vector of the same degree, then*

- *determinant of A,*
- *rational system solution $A^{-1}b$,*

can be computed in $\tilde{O}(dn^\omega)$ operations in $\mathbb{F}$ with high probability.

Storjohann's result immediately implies a $\tilde{O}(Wn^\omega)$ time algorithm to compute the weight of an optimal matching. Sankowski [61] showed how to construct an optimal bipartite matching in the same time. Later Harvey [28] generalized

this to linear matroid intersection in almost the same time. Their algorithms are based on clever ways to use Storjohann's result to compute optimal dual solutions. To compute the optimal dual solutions, they compute the weights of many problems at the same time, and an important technical detail is that they manage to retrieve all the information in one column of the inverse matrix, as Storjohann's result does not allow us to compute the whole inverse in the same amount of time (in fact it is impossible as pointed out by Sankowski [61]). It is still not known how to generalize these results to construct a maximum weighted matching in a general graph.

2.4.5 Parallel Algorithms

The algebraic approach also allows us to obtain parallel algorithms for combinatorial optimization problems. Although it is known how to compute the determinant and the inverse of a matrix in parallel [37], an issue in parallel computing is how to coordinate all the computers to search for the same maximum matching. Mulmuley, Vazirani and Vazirani [50] proved an Isolating Lemma to get around this problem.

Theorem 2.22 ([50])**.** *Given a set system with n elements, if each element is assigned a uniformly random weight from $[1, 2n]$, then the probability of having a unique minimum weighted set is at least $1/2$.*

To solve the minimum weighted perfect matching problem in bipartite graphs, we first assign a uniform random weight from $[1, 2m]$ to each edge. And in the matrix they put $2^{w_{ij}}$ in the ij-th entry. By the Isolating Lemma we have a unique minimum weighted matching with probability at least one half. And the minimum weight w is now the largest number for which 2^w divides $\det A$. Let $\tilde{A}_{ij}$ be the matrix obtained by removing i-th row and j-th column of A. To let each parallel process to determine if an edge belongs to the minimum weight matching, Mulmuley et al. showed that an edge $u_i v_j$ belongs to the unique minimum weighted perfect matching if and only if $(\det \tilde{A}_{ij} \cdot 2^{w_{ij}})/2^w$ is odd. To see this first notice that $\det \tilde{A}_{ij} \cdot 2^{w_{ij}}$ is the sum of the terms for matchings that use the edge (u_i, v_j). If (u_i, v_j) is in the minimum matching, all other permutations

have value zero or a higher power of two, and thus that term is odd. Otherwise, all permutations have value zero or a power of two higher than 2^w and thus that term is even.

To adapt this method to the weighted problem one just need to assign the weight $2mnw_e + r_e$ to each edge where r_e is from $[1, 2m]$.

2.5 Algebraic Algorithms for Graph Connectivity

The edge connectivity from vertex s to vertex t is defined as the size of a minimum s-t cut, or equivalently the maximum number of edge disjoint paths from s to t. In this section we review some algebraic algorithms for the edge connectivity problem. We will focus on simple directed graphs.

We begin with some notation and definitions for graphs. In a directed graph $G = (V, E)$, an edge $e = (u, v)$ is directed from u to v, and we say that u is the tail of e and v is the head of e. For any vertex $v \in V$, we define $\delta^{in}(v) = \{uv \mid uv \in E\}$ as the set of incoming edges of v and $d^{in}(v) = |\delta^{in}(v)|$; similarly we define $\delta^{out}(v) = \{vw \mid vw \in E\}$ as the set of outgoing edges of v and $d^{out}(v) = |\delta^{out}(v)|$. For a subset $S \subseteq V$, we define $\delta^{in}(S) = \{uv \mid u \notin S, v \in S, uv \in E\}$ and $d^{in}(S) = |\delta^{in}(S)|$.

2.5.1 Previous Approach

The previously known way to solve s-t edge connectivity algebraically is as follows. First reduce the edge connectivity problem to a vertex connectivity problem by constructing the directed line graph $L(G) = (V_L, E_L)$. Add a supersource s' in $L(G)$, and add edges from s' to vertices in $L(G)$ that represent edges incident from s. Also add edges from vertices that represent edges going to t, to a supersink t'. Now the s'-t' vertex connectivity in $L(G)$ is equal to the s-t edge connectivity in G.

Next we reduce vertex connectivity to bipartite matching. First let S be the set

of vertices that have incoming edges from s', and let T be the set of vertices that have outgoing edges to t'. Here we assume there is no intersection between S and T, otherwise we can safely remove each of these vertices and this will decrease the vertex connectivity exactly by one. We construct the bipartite graph $G' = (U', V', E')$ as follows. Split each vertex v in $L(G)$ to two vertices v_{out} and v_{in}, and place them in U' and V' respectively. For each vertex v of $L(G)$ that is not in either S or T, add an edge (v_{out}, v_{in}). For each edge (u, v) from $V_L - T$ to $V_L - S$ in $L(G)$, add an edge (u_{out}, v_{in}). Using only the edges of the form (v_{out}, v_{in}), there is a bipartite matching of size $|V_L| - |S| - |T|$. Observe that each vertex disjoint path from s' to t' increases the size of this bipartite matching in G' by one. Thus there are k vertex disjoint s'-t' paths in $L(G)$ if and only if there is a bipartite matching of size $|V_L| - |S| - |T| + k$ in G'. Since $L(G)$ has m vertices and solving the bipartite matching problem on an m vertex graph takes $O(m^\omega)$ time using algebraic algorithms, we have an $O(m^\omega)$ time algorithm for computing the s-t edge connectivity.

2.5.2 Matrix Formulation Using Network Coding

Here we present a recent result [12] that formulates edge connectivities algebraically using the ideas developed in network coding.

Given a directed graph $G = (V, E)$ and a specified source vertex s, we are interested in computing the edge connectivities from s to other vertices. Let $E = \{e_1, e_2, \ldots, e_m\}$, $d = d^{out}(s)$ and $\delta^{out}(s) = \{e_1, \ldots, e_d\}$. Also let $\mathbb{F}$ be a finite field. For each edge $e \in E$, we associate a *global encoding vector* $f_e \in \mathbb{F}^d$ of dimension d where each entry is in $\mathbb{F}$. We say a pair of edges e' and e are *adjacent* if the head of e' is the same as the tail of e, i.e. $e' = (u, v)$ and $e = (v, w)$ for some $v \in V$. For each pair of adjacent edges e' and e, we associate a *local encoding coefficient* $k_{e',e} \in \mathbb{F}$. Given the local encoding coefficients for all pairs of adjacent edges in G, we say that the global encoding vectors are a *network coding solution* if the following two sets of equations are satisfied:

1. For each edge $e_i \in \delta^{out}(s)$, we have $f_{e_i} = \sum_{e' \in \delta^{in}(s)} k_{e',e_i} \cdot f_{e'} + \vec{e}_i$, where $\vec{e}_i$ is the i-th vector in the standard basis.

2. For each edge $e = (v, w)$ with $v \neq s$, we have $f_e = \sum_{e' \in \delta^{in}(v)} k_{e',e} \cdot f_{e'}$.

Theorem 2.23 ([12]). *If we choose each local encoding coefficient independently and uniformly at random from a field $\mathbb{F}$, then with probability at least $1 - O(m^3/|\mathbb{F}|)$:*

1. *There is a unique network coding solution for the global encoding vectors f_e.*

2. *For $t \in V - s$, let $\delta^{in}(t) = \{a_1, \ldots, a_l\}$, the edge connectivity from s to t is equal to the rank of the matrix $(f_{a_1}|f_{a_2}|\ldots|f_{a_l})$.*

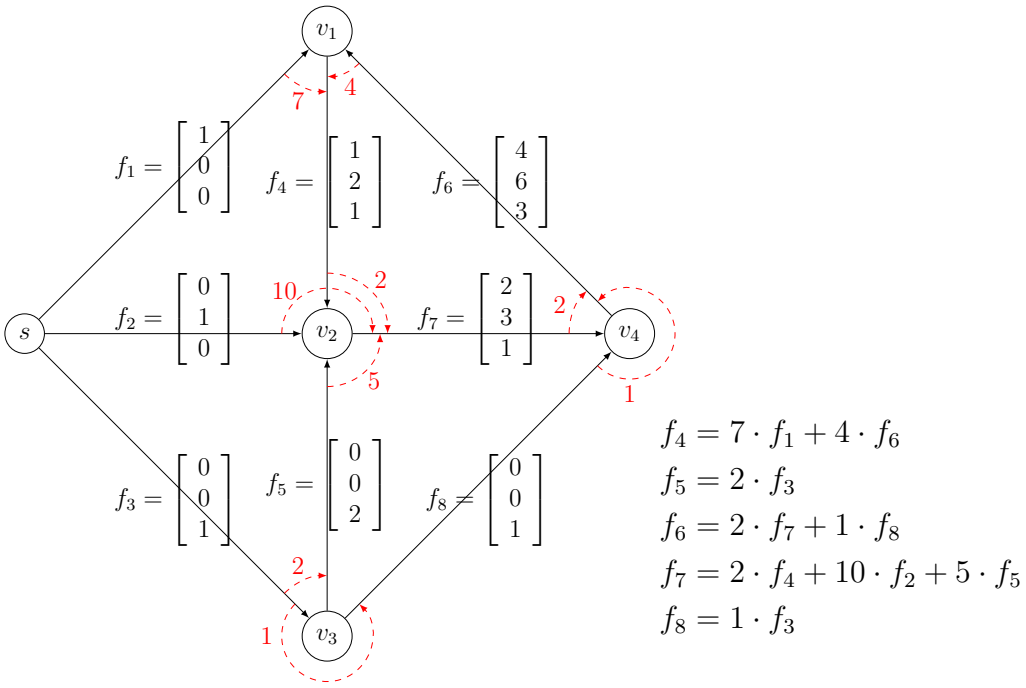


Figure 2.2: In this example three independent vectors f_1, f_2 and f_3 are sent from the source s. Other vectors are a linear combination of the incoming vectors, according to the random coefficients on the dotted lines. All operations are done in the field of size 11. To compute the edge connectivity from s to v_2, for instance, we compute the rank of $(f_2|f_4|f_5)$ which is 3 in this example.

See Figure 2.2 for an example of the theorem, and we will now prove it. First we

rewrite the requirements of a network coding solution in a matrix form.

$$\begin{pmatrix} & | & \\ \cdots & f_{e_j} & \cdots \\ & | & \end{pmatrix} = \begin{pmatrix} & | & \\ \cdots & f_{e_j} & \cdots \\ & | & \end{pmatrix} \begin{pmatrix} & k_{e_1,e_j} & \\ & \vdots & \\ \cdots & k_{e_i,e_j} & \cdots \\ & \vdots & \\ & k_{e_m,e_j} & \end{pmatrix} + \begin{pmatrix} | & & | & | & & | \\ \vec{e}_1 & \cdots & \vec{e}_d & \vec{0} & \cdots & \vec{0} \\ | & & | & | & & | \end{pmatrix}$$

Let F be the $d \times m$ matrix $(f_{e_1}|\ldots|f_{e_m})$. Let K be the $m \times m$ matrix where $K_{i,j} = k_{e_i,e_j}$ when e_i and e_j are adjacent edges, and 0 otherwise. Let H_s be the $d \times m$ matrix $(\vec{e}_1|\ldots|\vec{e}_d|\vec{0}|\vec{0}|\ldots|\vec{0})$ where $\vec{0}$ denotes the all zero vector of dimension d. Then the network coding equations are equal to the following matrix equation:

$$F = FK + H_s. \tag{2.5.5}$$

To prove the first part of Theorem 2.23, we will prove in the following lemma that $(I - K)$ is non-singular with high probability. Then the above equation can be rewritten as $F = H_s(I-K)^{-1}$, which implies that the global encoding vectors are uniquely determined.

Lemma 2.24 ([12])**.** *Given the conditions in Theorem 2.23, the matrix* $(I - K)$ *is non-singular with probability at least* $1 - O(m/|\mathbb{F}|)$.

Proof. Since the diagonal entries of K are zero, the diagonal entries of $I - K$ are all one. By treating each entry of K as an indeterminate $K_{i,j}$, it follows that $\det(I - K) = 1 + p(\cdots, K_{i,j}, \cdots)$ where $p(\cdots, K_{i,j}, \cdots)$ is a polynomial of the indeterminates with total degree at most m. Note that $\det(I - K)$ is not a zero polynomial since there is a constant term. Hence, by the Schwartz-Zippel Lemma, if each $K_{i,j}$ is a random element in $\mathbb{F}$, then $\det(I - K) = 0$ with probability at most $O(m/|\mathbb{F}|)$, proving the lemma. □

After we obtained the global encoding vectors, we would like to show that the edge connectivities can be determined from the ranks of these vectors. Consider a vertex $t \in V - s$. Let $\delta^{in}(t) = \{a_1, \ldots, a_l\}$ and let M_t be the $d \times l$ matrix

$(f_{a_1}|f_{a_2}|\ldots|f_{a_l})$. Let the edge connectivity from s to t be $\lambda_{s,t}$. We prove in the following lemma that $\text{rank}(M_t) = \lambda_{s,t}$ with high probability.

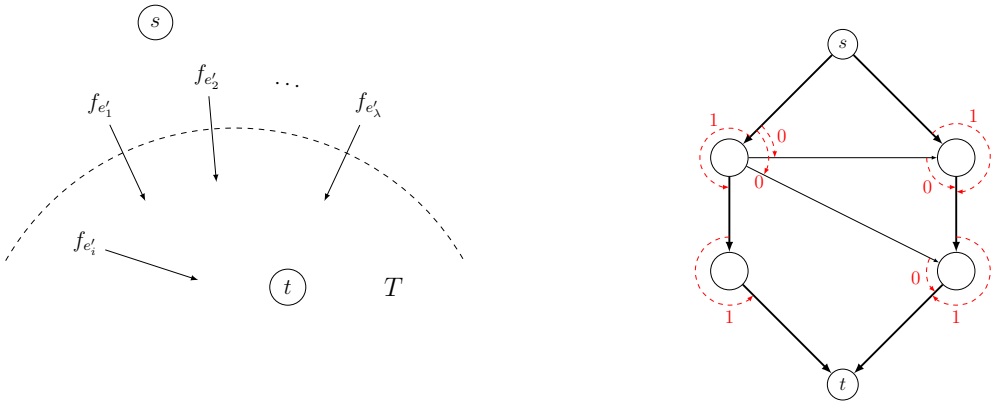
Lemma 2.25 ([12]). *Given the conditions of Theorem 2.23, we have* $\text{rank}(M_t) = \lambda_{s,t}$ *with probability at least* $1 - O(m^2/|\mathbb{F}|)$.

Proof. First we prove that $\text{rank}(M_t) \leq \lambda_{s,t}$ with high probability. The plan is to show that the global encoding vector on each incoming edge of t is a linear combination of the global encoding vectors in a minimum s-t cut with high probability. Consider a minimum s-t cut $\delta^{in}(T)$ where $T \subset V$ with $s \notin T$ and $t \in T$ and $d^{in}(T) = \lambda_{s,t}$. Let $E' = \{e'_1, \ldots, e'_{m'}\}$ be the set of edges in E with their heads in T. Let $\lambda = \lambda_{s,t}$ and assume $\delta^{in}(T) = \{e'_1, \ldots, e'_\lambda\}$. See Figure 2.3a for an illustration. Let F' be the $d \times m'$ matrix $(f_{e'_1}|\ldots|f_{e'_{m'}})$. Let K' be the $m' \times m'$ submatrix of K restricted to the edges in E'. Let H' be the $d \times m'$ matrix $(f_{e'_1}|\ldots|f_{e'_\lambda}|\vec{0}|\ldots|\vec{0})$. Then, by the network coding requirements, the matrices satisfy the equation

$$F' = F'K' + H'.$$

By the same argument as in Lemma 2.24, the matrix $(I - K')$ is nonsingular with probability at least $1 - O(m/|\mathbb{F}|)$. So the above matrix equation can be rewritten as $F' = H'(I - K')^{-1}$. This implies that every global encoding vector in F' is a linear combination of the global encoding vectors in H', which are the global encoding vectors in the cut $\delta^{in}(T)$. Therefore the $\text{rank}(M_t) \leq d^{in}(T) = \lambda_{s,t}$.

Now we prove that $\text{rank}(M_t) \geq \lambda_{s,t}$ with high probability. The plan is to show that there is a $\lambda \times \lambda$ submatrix M'_t of M_t such that $\det(M'_t)$ is a non-zero polynomial of the local encoding coefficients with small total degree. First we use the edge disjoint paths from s to t to define M'_t. Let $\lambda = \lambda_{s,t}$ and $P_1, \ldots, P_\lambda$ be a set of λ edge disjoint paths from s to t. Set $k_{e',e} = 1$ for every pair of adjacent edges $e', e \in P_i$ for every i, and set all other local encoding coefficients to be zero. See Figure 2.3b for an illustration. Then each path sends a distinct unit vector of the standard basis to t, and thus M_t contains a $\lambda \times \lambda$ identity matrix as a submatrix. Call this $\lambda \times \lambda$ submatrix M'_t. Next we show that the $\det(M'_t)$ is a non-zero polynomial of the local encoding coefficients with small total degree. Recall that $F = H(I - K)^{-1}$. By considering the adjoint matrix of $I - K$, each



(a) Upper bound by min-cut.

(b) Lower bound by paths.

Figure 2.3: Rank of vectors received related to connectivity.

entry of $(I-K)^{-1}$ is a polynomial of the local encoding coefficients with total degree m divided by $\det(I-K)$, and thus the same is true for each entry of F. Hence $\det(M'_t)$ is a degree λm polynomial of the local encoding coefficients divided by $(\det(I-K))^\lambda$. By using the edge disjoint paths $P_1, \ldots, P_\lambda$, we have shown that there is a choice of the local encoding coefficients so that $\text{rank}(M'_t) = \lambda$. Thus $\det(M'_t)$ is a non-zero polynomial of the local encoding coefficients. Conditioned on the event that $\det(I-K)$ is nonzero, the probability that $\det(M'_t)$ is nonzero is at most $O(\lambda m/|\mathbb{F}|) \leq O(m^2/|\mathbb{F}|)$ by the Schwartz-Zippel lemma. This implies that M'_t is a full rank submatrix with high probability, and thus $\text{rank}(M_t) \geq rank(M'_t) = \lambda$. We conclude that $\text{rank}(M_t) = \lambda_{s,t}$ with probability at least $1 - O(m^2/|\mathbb{F}|)$. □

Thus the probability that $\text{rank}(M_t) \neq \lambda_{s,t}$ for some $t \in V - s$ is at most $n \cdot O(m^2/|\mathbb{F}|) \leq O(m^3/|\mathbb{F}|)$ by union bound on t, and this proves the second part of Theorem 2.23. Therefore, we only need to pick a large enough field $\mathbb{F}$ so that $|\mathbb{F}| = \Omega(m^4)$ to guarantee a high probability result.

Now we show that the matrix $(I-K)^{-1}$ actually contains the information of connectivities between all vertices.

Theorem 2.26 ([12]). *The rank of the submatrix* $(I-K)^{-1}_{\delta^{out}(s),\delta^{in}(t)}$ *gives the s-t edge connectivity.*

Proof. To solve $F = FK + H_s$, we compute $(I - K)^{-1}$ and get $F = H_s(I - K)^{-1}$. In our setup, H_s is a $d^{out}(s) \times m$ matrix with a $d^{out}(s) \times d^{out}(s)$ identity matrix in the columns corresponding to $\delta^{out}(s)$. So F is just equal to $((I - K)^{-1})_{\delta^{out}(s),*}$ up to permuting rows. To compute the edge connectivity from s to t, by Theorem 2.23 we compute the rank of $F_{*,\delta^{in}(t)}$ and this is just equal to the rank of $((I - K)^{-1})_{\delta^{out}(s),\delta^{in}(t)}$. □

Since $I - K$ is an $m \times m$ matrices, its inverse can be computed in $O(m^\omega)$ time. Given vertices s and t, computing the rank of $((I - K)^{-1})_{\delta^{out}(s),\delta^{in}(t)}$ takes $O(d^{out}(t)(d^{out}(s))^{\omega-1})$ time. So the total time to compute the ranks of the required submatrices for all s and t is

$$\sum_{s \in V} \sum_{t \in V} O(d^{out}(t)(d^{out}(s))^{\omega-1}) = \sum_{s \in V} O(m \cdot (d^{out}(s))^{\omega-1}) = O(m^\omega).$$

Thus the total time to compute edge connectivities between all s and t is $O(m^\omega)$ time. In Chapter 4 we will show that $(I - K)^{-1}$ can be computed more efficiently in graphs with good separators.

Chapter 3

Linear Matroid Parity

The results presented in this chapter are based on joint work with Lap Chi Lau and Kai Man Leung [11].

3.1 Introduction

The graph matching problem and the matroid intersection problem are two fundamental polynomial-time solvable problems in combinatorial optimization. Several efforts have been made to obtain an elegant common generalization of these two problems, e.g. the matroid parity problem by Lawler [39] (equivalent to the matchoid problem by Edmonds [34] and the matroid matching problem by Lovász [42]), the optimal path-matching problem by Cunningham and Geelen [16], and the membership problem for jump system by Bouchet and Cunningham [5, 43].

So far the matroid parity problem is the most-studied and the most fruitful problem among these generalizations. Although it is shown to be intractable in the oracle model [35] and is NP-hard for matroids with compact representations [42], Lovász [42] proved an exact min-max formula and obtained a polynomial time algorithm for the *linear* matroid parity problem.

This provides a polynomial-time solvable common generalization of the graph

matching problem and the linear matroid intersection problem. Moreover, the linear matroid parity problem has many applications of its own in various areas, including the path packing problem [46, 63] in combinatorial optimization, the minimum pinning set problem [42, 36] in combinatorial rigidity, the maximum genus imbedding problem [20] in topological graph theory, the graphic matroid parity problem [42, 22] used in approximating minimum Steiner tree [59, 3] and approximating maximum planar subgraph [8], and the unique solution problem [44] in electric circuit.

Given its generality and applicability, it is thus of interest to obtain fast algorithms for the linear matroid parity problem. In this paper we will present faster and simpler algorithms for the linear matroid parity problem, and also improved algorithms for specific graph problems of interest. The algorithms are based on the algebraic algorithmic framework developed by Mucha and Sankowski [48], Harvey [30, 28], and Sankowski [61].

3.1.1 Problem Formulation and Previous Work

The linear matroid parity problem can be formulated as follows without using terminology from matroid theory: given an $r \times 2m$ matrix whose columns are partitioned into m pairs, find a maximum cardinality collection of pairs so that the union of the columns of these pairs are linearly independent. For instance, to formulate the graph matching problem as a linear matroid parity problem, we construct an $n \times 2m$ matrix where the rows are indexed by the vertices and the pairs are indexed by the edges, where an edge ij is represented by two columns where one column has an 1 in the i-th entry and 0 otherwise and the other column has an 1 in the j-th entry and 0 otherwise.

There are several deterministic combinatorial algorithms for the linear matroid parity problem. The first polynomial time algorithm is obtained by Lovász with a running time of $O(m^{17})$ which can be implemented to run in $O(m^{10})$ time [42, 44]. The fastest known algorithm is an augmenting path algorithm obtained by Gabow and Stallmann [23] with running time $O(mr^{\omega})$ [63], where $\omega \approx 2.376$ is the exponent on the running time of the fastest known matrix multiplication

algorithm [14]. Orlin and Vande Vate [55] presented an algorithm with running time $O(mr^{\omega+1})$ [63] by reducing it to a sequence of matroid intersection problems. Recently Orlin [54] presented a simpler algorithm with running time $O(mr^3)$. While these algorithms are all deterministic and reveal substantial structural insights into the problem, even the simplest algorithm by Orlin is quite complex and probably too difficult to implement in practice.

On the other hand, Lovász [41] proposed an algebraic approach to the linear matroid parity problem. First, he constructed an appropriate matrix with indeterminates (variables) where the matrix is of full rank if and only if there are $r/2$ linearly independent pairs (see Section 2.9). Then he showed that determining whether the matrix is of full rank can be done efficiently with high probability, by substituting the variables with independent random values from a large enough field, and then computing the determinant of the resulting matrix [41]. This approach can be easily modified to determine the optimal value of the linear matroid parity problem in one matrix multiplication time, and one can also construct a solution in m matrix multiplications time. Note that this already gives a randomized $O(mr^{\omega})$-time algorithm for the linear matroid parity problem, and this algebraic approach also leads to an efficient parallel algorithm for the linear matroid parity problem [52].

In a recent line of research an elegant algorithmic framework has been developed for this algebraic approach. Mucha and Sankowski [48] showed how to use Gaussian eliminations to construct a maximum matching in one matrix multiplication time, leading to an $O(n^{\omega})$ time algorithm for the graph matching problem where n is the number of vertices. Harvey [30] used a divide-and-conquer method to obtain an algebraic algorithm for the linear matroid intersection problem with running time $O(mr^{\omega-1})$ where m is the number of columns, and a simple $O(n^{\omega})$ time algorithm for the graph matching problem. Furthermore, Sankowski [61] and Harvey [28] extended the algebraic approach to obtain faster pseudo-polynomial algorithms for the weighted bipartite matching problem and the weighted linear matroid intersection problem.

Besides matching and linear matroid intersection, other special cases of the linear matroid parity problem have also been studied. One special case of interest

is the graphic matroid parity problem [22, 24, 66, 67], which has applications in designing approximation algorithms [8, 59, 3]. For this problem the fastest known algorithm is by Gabow and Stallmann [22] which runs in $O(mn \lg^6 n)$ time. Another special problem of considerable interest is Mader's $\mathcal{S}$-path packing problem [46, 42, 64, 13, 56, 57, 58, 2] which is a generalization of the graph matching problem and the s-t vertex disjoint path problem. Lovász [42] showed that this problem can be reduced to the linear matroid parity problem. Chudnovsky, Cunningham and Geelen [13] obtained an $O(n^6)$ time direct combinatorial algorithm for the problem, and Pap [58, 57] obtained a simpler direct combinatorial algorithm for the problem and also for the more general capacitated setting.

3.1.2 Our Results

We obtain fast and simple algebraic algorithms for the linear matroid parity problem and also for some specific graph problems of interest. All algorithms are best possible in the sense that either they match the running time in well-known special cases or they are optimal in terms of some parameters.

Linear Matroid Parity

There are two algebraic formulations for the linear matroid parity problem, one is a "compact" formulation by Lovász in Theorem 2.9 and another is a "sparse" formulation by Geelen and Iwata in Theorem 2.10. Using the compact formulation and the Sherman-Morrison-Woodbury formula, we present a very simple algorithm for the linear matroid parity problem.

Theorem 3.1. *There is an $O(mr^2)$-time randomized algorithm for the linear matroid parity problem.*

One feature of this algorithm is that it does not use fast matrix multiplication and is very easy to be implemented in practice. Note that it is already faster than the Gabow-Stallmann $O(mr^\omega)$ time algorithm, and actually if fast matrix multiplication is not used then the best known algorithms run in $O(mr^3)$ time [23, 54]. Using the divide-and-conquer method of Harvey [30] on the sparse formulation

and fast matrix multiplications, we can improve the running time further to match the running time of the linear matroid intersection problem, answering a question of Harvey [30].

Theorem 3.2. *There is an $O(mr^{\omega-1})$-time randomized algorithm for the linear matroid parity problem.*

It is still open whether there is a polynomial time algorithm for the *weighted* linear matroid parity problem, and even a deterministic PTAS is not known yet [40]. We present a faster pseudo-polynomial randomized algorithm for the weighted matroid parity problem, which also implies a faster randomized FPTAS for the weighted linear matroid parity problem using standard scaling technique [59].

Graph Algorithms

For graph problems that can be reduced to the linear matroid parity problem, we show that the additional structure can be exploited in the compact formulation to obtain faster algorithms than that follow from Theorem 3.2. We illustrate this with some well-known problems.

Mader's Disjoint $\mathcal{S}$-Path In this problem we are given an undirected graph $G = (V, E)$ and $\mathcal{S}$ is a collection of disjoint subsets of V. The goal is to find a maximum collection of vertex disjoint $\mathcal{S}$-paths, where an $\mathcal{S}$-path is a path that connects two different sets in $\mathcal{S}$ and has no internal vertex in $\mathcal{S}$. This problem generalizes the graph matching problem and the vertex disjoint s-t path problem, and is of considerable interest [46, 42, 64, 13, 56, 57, 58, 2]. Obtaining a direct combinatorial algorithm is quite nontrivial [13, 58]. The best known running time is still the $O(mn^{\omega})$-time bound implied by the Gabow-Stallmann algorithm, where m is the number of edges and n is the number of vertices. The algorithm in Theorem 3.2 implies an $O(mn^{\omega-1})$-time algorithm. By using the compact formulation, we further improve the running time to match the algebraic algorithms for the graph matching problem. The algorithm would be quite simple if fast matrix multiplication is not used, and its running time would be $\tilde{O}(n^3)$ which is still faster than the existing algorithms.

Theorem 3.3. *There is an $O(n^\omega)$-time randomized algorithm for Mader's $\mathcal{S}$-path problem.*

Graphic Matroid Parity In this problem we are given an undirected graph and some edge pairs, and the problem is to find a maximum collection of edge pairs such that the union of these edges forms a forest. One special case of interest [44, 66] is when each pair has a common vertex (i.e. $\{ij, ik\}$ for some vertex i). This has applications in approximating minimum Steiner tree [59, 3] and approximating maximum planar subgraph [8]. In the general problem the input could have up to $\Omega(n^4)$ edge pairs where n is the number of vertices, and in the special problem the number of edge pairs could be $\Omega(n^3)$. The following algorithms achieve optimal running time in terms of n for both problems.

Theorem 3.4. *There is an $O(n^4)$-time randomized algorithm for the graphic matroid parity problem, and an $O(n^3)$-time randomized algorithm when each edge pair has a common vertex.*

The fastest algorithm on graphic matroid parity is obtained by Gabow and Stallmann [22] with running time $O(mn \lg^6 n)$ where m is the number of edge pairs, and so our algorithm is faster if there are $\Omega(n^3)$ edge pairs in the general problem and if there are $\Omega(n^2)$ edge pairs in the special problem. We remark that the same statement holds even if we use a cubic algorithm for matrix multiplication, and the resulting algorithm is much simpler than that of Gabow and Stallmann.

Colorful Spanning Tree In this problem we are given an undirected multigraph $G = (V, E)$ where each edge has one color, and the objective is to determine whether there is a spanning tree in which every edge has a distinct color. This is a generalization of the arborescence problem and the connected detachment problem [53, 63], and is a special case of the linear matroid intersection problem. Note that the input graph could have $\Omega(n^3)$ edges where n is the number of vertices, since each pair of vertices could have $\Omega(n)$ edges in between, each of which has a distinct colors. So the following algorithm has optimal running time in terms of n.

Theorem 3.5. *There is an $O(n^3)$-time randomized algorithm for the colorful spanning tree problem.*

3.1.3 Techniques

Our results show that both the algebraic algorithms for graph matching and linear matroid intersection can be generalized to linear matroid parity. The $O(mr^{\omega-1})$-time algorithm for linear matroid parity is a straightforward generalization of Harvey's linear matroid intersection algorithm, and the algorithm for weighted linear matroid parity follows from the techniques used by Sankowski [61]. The main new technical contribution is the use of the compact formulation to design new algebraic algorithms. For graph problems, the basic observation is that the column vectors have at most a constant number of nonzeros, and this allows us to extend Harvey's matching algorithm to obtain faster algorithms using the compact formulation. The $O(n^\omega)$ algorithm for the $\mathcal{S}$-path problem is based on a good matrix formulation of the problem, while the $O(n^3)$ algorithms for graphic matroid parity and colorful spanning tree are based on different recursions used in the divide-and-conquer method. We remark that this approach on the compact formulation implies some new results for linear matroid intersection problems as well, e.g. colorful spanning tree, graphic matroid intersection, simple $O(mr^2)$ algorithm.

While linear matroid parity and Mader's disjoint $\mathcal{S}$-path are challenging generalizations for the design of combinatorial algorithms, our results show that the algebraic algorithmic framework can be adapted nicely to give faster and simpler algorithms in more general settings. Our algorithms are still faster than the existing algorithms even if fast matrix multiplications are not used, and these simpler algorithms could be implemented easily in practice using MATLAB (see e.g. [29]).

Organization In Section 3.3, we will give the proof of Theorem 3.1. Then we will present graph algorithms for Mader's Disjoint $\mathcal{S}$-path problem, the graphic matroid parity problem and the colorful spanning tree problem in Section 3.4.

The results on weighted linear matroid parity will be presented in Section 3.5, and the proof of Theorem 3.2 will be presented in Section 3.6.

3.2 Preliminaries

When a set of integers S are partitioned into k subsets, the set S is partitioned into k equal size subsets S_1, S_2, $\cdots S_k$. In addition, S_1 contains the smallest $|S|/k$ elements of S, S_2 contains the next $|S|/k$ smallest elements of S, and S_k contains the largest $|S|/k$ elements of S.

We assume the number of pairs m and the number of rows r in the linear matroid parity problem will be powers of two. This assumption can be easily satisfied by adding redundant pairs and rows.

3.3 A Simple Algebraic Algorithm for Linear Matroid Parity

In this section we will present a matrix formulation and a simple $O(mr^2)$ time algorithm for the linear matroid parity problem.

Given an $r \times 2m$ matrix M where the columns are partitioned into m pairs $\{\{b_1, c_1\}, \ldots, \{b_m, c_m\}\}$, the linear matroid parity problem is to find a maximum collection of pairs $J \subseteq [m]$ so that the vectors in $\bigcup_{i \in J}\{b_i, c_i\}$ are linearly independent. We use ν_M to denote the optimal value, and call an optimal solution a *parity basis* if $\nu_M = r/2$. We also call a set *parity set* if every column pair is either contained in it or disjoint from it.

3.3.1 An $O(mr^2)$ Algorithm

In this subsection we present a very simple $O(mr^2)$-time algorithm for the linear matroid parity problem. Here we consider the case where we find a parity basis

if one exists or report that no parity basis exists. We will show how to reduce the general problem to this case in Section 3.7.

A pseudocode of the algorithm is presented in Algorithm 3.1. First we construct the matrix Y with indeterminates using the compact formulation in Theorem 2.9.

Theorem 2.9. *Given m column pairs $\{(b_i, c_i)\}$ for $1 \leq i \leq m$ and $b_i, c_i \in \mathbb{R}^r$. Let*

$$Y = \sum_{i=1}^{m} x_i(b_i \wedge c_i),$$

where x_i are indeterminates. Then $2\nu_M = \operatorname{rank}(Y)$.

By Theorem 2.9 we have $\nu_M = r/2$ if and only if Y is of full rank. As stated in Section 3.2, we can test whether Y is of full rank in $O(r^3)$ time with high probability, by substituting the indeterminates with random values and then checking whether the resulting matrix has nonzero determinant. If Y is not of full rank, then we report that no parity basis exists, otherwise we construct the matrix Y^{-1} in $O(r^3)$ time.

Then, for each column pair (b_i, c_i), the algorithm checks whether this pair can be removed while keeping the resulting matrix full rank. If so this pair is removed from the problem since there is still an optimal solution surviving, otherwise this pair is kept since it is in every parity basis with high probability. In the end the algorithm returns the pairs that were not removed.

Next we show how to check whether a pair can be removed efficiently. First we recall Theorem 2.15.

Theorem 2.15 (Sherman-Morrison-Woodbury)**.** *Let M be an $n \times n$ matrix, U be an $n \times k$ matrix, and V be an $n \times k$ matrix. Suppose that M is non-singular. Then*

1. *$M + UV^T$ is non-singular if and only if $I + V^T M^{-1} U$ is non-singular.*
2. *If $M + UV^T$ is non-singular, then $(M + UV^T)^{-1} = M^{-1} - M^{-1}U(I + V^T M^{-1} U)^{-1} V^T M^{-1}$.*

Removing the i-th column pair from M is equivalent to assign x_i to zero. Let Y' be the new matrix with $x_i = 0$, then

$$Y' = Y - x_i(b_i c_i^T - c_i b_i^T) = Y - x_i \begin{pmatrix} b_i & c_i \end{pmatrix} \begin{pmatrix} c_i & -b_i \end{pmatrix}^T$$

Observe that this is just a rank-2 update. By setting $U = x_i \begin{pmatrix} b_i & c_i \end{pmatrix}$ and $V = \begin{pmatrix} -c_i & b_i \end{pmatrix}$ and using Theorem 2.15(1), Y' is of full rank if and only if $I + V^T Y^{-1} U$ is of full rank. Since both U and V are of size $r \times 2$, we can check whether a pair can be removed in $O(r^2)$ time. If so, we apply Theorem 2.15(2) to compute the inverse of Y' by the formula $Y^{-1} - Y^{-1}U(I + V^T Y^{-1} U)^{-1} V^T Y^{-1}$, this can be computed in $O(r^2)$ time since $I + V^T Y^{-1} U$ is of size 2×2. Applying this procedure iteratively, the whole algorithm can be implemented in $O(mr^2)$ time.

Finally the algorithm fails only if a matrix is of full rank but the determinant is zero after the random substitutions. As stated in Section 3.2, this happens with probability at most r/q where q is the field size. Since we only check the rank at most m times, the failure probability is at most mr/q by the union bound, and so by choosing $q = mr/\epsilon$ this probability is at most ϵ.

Algorithm 3.1 A simple algebraic algorithm for linear matroid parity

```
SimpleParity(M)
  Construct Y using the compact formulation and assign random values to
  indeterminates x_i
  if det(Y) = 0 return "there is no parity basis"
  Compute Y^{-1}
  Set I = {b_1, c_1, ··· b_m, c_m}
  for i = 1 to m do
    Set Y' := Y − x_i (b_i  c_i) (c_i  -b_i)^T
    if det(Y') ≠ 0 then
      Y := Y'
      Update Y^{-1} by the Sherman-Morrison-Woodbury formula
      I := I − {b_i, c_i}
  return I
```

3.4 Graph Algorithms

In most applications of linear matroid parity, not only is the given matroid a linear matroid, but also each column vector of the matroid has few nonzero entries. For example, each column vector of a graphic matroid has only two nonzero entries. In this section, we will show how we can exploit such special structure to obtain faster algorithms for some graph problems of interest.

For Mader's $\mathcal{S}$-path problem in Section 3.4.1, we will translate the reduction into a good matrix formulation, so that the recursive approach for graph matching problem can be extended to solve this problem. Also, we will give different recursive algorithms to solve the graphic matroid parity problem in Section 3.4.2 and the colorful spanning tree problem in Section 3.4.3.

Our algorithms below assume the matroid parity instance contains a parity basis. If not we can use the approach to be described in Section 3.7 to reduce to this case: Suppose the given linear matroid M has r rows. Consider the matrix formulation Y in Theorem 2.9. The maximum rank submatrix $Y_{S,S}$ can be found in $O(r^\omega)$ time, and then we only need to focus on $Y_{S,S}$. At any time our algorithm considers a submatrix $Y_{R,C}$, we shall consider $Y_{R\cap S,C\cap S}$ instead.

3.4.1 Mader's $\mathcal{S}$-Path

Given an undirected graph $G = (V, E)$ and let $S_1, \cdots, S_k$ be disjoint subsets of V. A path is called an $\mathcal{S}$-path if it starts and ends with vertices in S_i and S_j such that $S_i \neq S_j$, while all other internal vertices of the path are in $V\backslash(S_1 \cup S_2 \cup \cdots \cup S_k)$. The disjoint $\mathcal{S}$-path problem is to find a maximum cardinality collection of vertex disjoint $\mathcal{S}$-Paths of the graph G. In the following we assume without loss of generality that each S_i is a stable set.

Lovász [42] showed that the $\mathcal{S}$-path problem can be reduced to the linear matroid parity problem, but it is not immediately clear how his reduction can be translated into a matrix formulation of the problem. Instead, we will follow the reduction by Schrijver ([63] page 1284), and show that it can be translated into a good matrix formulation.

Reduction to Linear Matroid Parity

Here we first recall the reduction from Section 2.1.4. The high level idea is to associate each edge to a 2-dimensional linear subspace, and show that the edges in a solution of the $\mathcal{S}$-path problem correspond to subspaces that are linearly independent in an appropriately defined quotient space $\mathbb{R}^{2n}/Q$, where two subspaces are linearly independent if their basis vectors are linearly independent.

Associate each edge $e = (u, w) \in E$ to a 2-dimensional linear subspace L_e of $(\mathbb{R}^2)^V$ such that

$$L_e = \left\{ x \in (\mathbb{R}^2)^V \mid x(v) = \mathbf{0} \text{ for each } v \in V \backslash \{u, w\} \text{ and } x(u) + x(w) = \mathbf{0} \right\}$$

where $x : V \to \mathbb{R}^2$ is a function that maps each vertex to a 2-dimensional vector. Let $r_1, \cdots r_k$ be k distinct 1-dimensional subspaces of $\mathbb{R}^2$. For each vertex $v \in V$, let $R_v = r_j$ if $v \in S_j$ for some j, and $R_v = \{\mathbf{0}\}$ otherwise. Define a linear subspace Q of $(\mathbb{R}^2)^V$ such that

$$Q = \{x \in (\mathbb{R}^2)^V \mid x(v) \in R_v \text{ for all } v \in V\}.$$

Let $\mathcal{E}$ be the collection of subspaces L_e/Q for each $e \in E$ of $(\mathbb{R}^2)^V/Q$, where L_e/Q is the quotient space of L_e by Q. Note that $\dim(L_e/Q) = 2$ for all edges e, since it does not connect two vertices in the same S_i as we assume each S_i is a stable set. In Theorem 2.2 we proved the following theorem which shows the reduction to the linear matroid parity problem.

Theorem 2.2. *If G is connected, then the maximum number of disjoint $\mathcal{S}$-paths is equal to $\nu(\mathcal{E}) - |V| + |T|$, where $T = \bigcup_{i=1}^{k} S_i$ and $\nu(\mathcal{E})$ is the size of a maximum collection of linearly independent 2-dimensional subspaces in $\mathcal{E}$.*

Matrix Formulation

To translate the above reduction into a matrix formulation, we need to associate each edge e to a column pair (b'_e, c'_e), such that for $F \subseteq E$ the subspaces in $\mathcal{L}_F = \{L_e/Q \mid e \in F\}$ are linearly independent if and only if the vectors in

$\bigcup_{e\in F}\{b'_e, c'_e\}$ are linearly independent.

Let $\vec{e}_k$ be the k-th unit vector. For each edge $e = (u, v) \in E$, construct an orthogonal basis b_e and c_e of L_e such that

$$b_e = \vec{e}_{2u-1} - \vec{e}_{2v-1} \quad \text{and} \quad c_e = \vec{e}_{2u} - \vec{e}_{2v},$$

where we abuse notation to also use u and v as indices of the vertices u and v. For $v \in V$ we define:

$$q_v = \begin{cases} \vec{e}_{2v-1} + i\vec{e}_{2v} & \text{if } v \in S_i \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Note that the collection of non-zero q_v forms an orthogonal basis of Q. To obtain the vectors for L_e/Q, we just need to write $b_e = b'_e + b_Q$ and $c_e = c'_e + c_Q$ where $b'_e, c'_e \in Q^\perp$ and $b_Q, c_Q \in Q$. Then, for any subset $F \subseteq E$, the vectors in $\bigcup_{e\in F}\{b_e, c_e\}$ are linearly independent in $\mathbb{R}^{2n}/Q$ if and only if the vectors in $\bigcup_{e\in F}\{b'_e, c'_e\}$ are linearly independent in $\mathbb{R}^{2n}$.

We can use a procedure similar to the Gram-Schmidt process to compute (b'_e, c'_e) from (b_e, c_e). Recall that the collection of non-zero q_v forms an orthogonal basis of Q. Define

$$b'_e = b_e - \sum_{v\in V: q_v\neq 0} \frac{b_e^T q_v}{q_v^T q_v} q_v \qquad c'_e = c_e - \sum_{v\in V: q_v\neq 0} \frac{c_e^T q_v}{q_v^T q_v} q_v.$$

By subtracting the projection of b_e onto q_v for all v from b_e, the resulting vector b'_e is orthogonal to the subspace Q. Thus, by the above discussion, we have that for each $F \subseteq E$, the subspaces in $\mathcal{L}_F = \{L_e/Q \mid e \in F\}$ are linearly independent if and only if the vectors in $\bigcup_{e\in F}\{b'_e, c'_e\}$ are linearly independent in $\mathbb{R}^{2n}$.

Therefore, by solving the linear matroid parity problem on the set of column pairs $\{(b'_e, c'_e)\}$ for all $e \in E$, we can find the maximum number of disjoint $\mathcal{S}$-paths in G, using Theorem 2.2. Also, from the solution of the linear matroid parity problem, one can easily construct the solution for the $\mathcal{S}$-path problem, (see Section 2.1.4).

Observe that for any $e = (u, v)$, after the Gram-Schmidt process, b'_e and c'_e are of

the form:

$$b'_e = \frac{i^2}{1+i^2}\vec{e}_{2u-1} - \frac{i}{1+i^2}\vec{e}_{2u} - \frac{j^2}{1+j^2}\vec{e}_{2v-1} + \frac{j}{1+j^2}\vec{e}_{2v}$$
$$c'_e = -\frac{i}{1+i^2}\vec{e}_{2u-1} + \frac{1}{1+i^2}\vec{e}_{2u} + \frac{j}{1+j^2}\vec{e}_{2v-1} - \frac{1}{1+j^2}\vec{e}_{2v}$$

where $u \in S_i$ and $v \in S_j$ for some i and j. If u or v are not in any S_i, then the corresponding entries in b'_e and c'_e remain the same as in b_e and c_e. Therefore, M contains at most four non-zero entries in each column. Now we can apply Theorem 2.9 to construct the described matrix Y for the linear matroid parity problem, which is given by $Y = \sum_{e \in E} x_e(b'_e \wedge c'_e)$.

Let $m = |E|$ and $n = |V|$. Then Y is a $2n \times 2n$ matrix. For each wedge product, there are at most four 2×2 non-zero blocks, and so for each edge e there are at most 16 entries of x_e in Y. Further observe that for any 2×2 non-zero block at the two rows occupied by u and two columns occupied by v of Y, the same block (but negated) appears at the two rows occupied by v and two columns occupied by u of Y. Hence the appearance of 2×2 blocks (as well as the indeterminates x_i) are always symmetric.

Recursive Algorithm

Our algorithm is a direct generalization of Harvey's graph matching algorithm discussed in Section 2.4.3. Here is the high-level idea of the recursive algorithm to construct a parity basis of M. Similar to the $O(mr^2)$-time algorithm in Section 3.3, the algorithm checks for each edge e whether some parity basis survives after the column pair (b'_e, c'_e) is removed. Removing a column pair (b'_e, c'_e) is equivalent to setting the corresponding x_e to zero. The observation is that each edge e has at most 16 entries of x_e in Y, and so the small area update formula of Harvey can be applied. Recall Theorem 2.16:

Theorem 2.16 (Harvey)**.** *Let M be a non-singular matrix and let $N = M^{-1}$. Let $\tilde{M}$ be a matrix which is identical to M except $\tilde{M}_{S,S} \neq M_{S,S}$ and let $\Delta = \tilde{M} - M$.*

1. *$\tilde{M}$ is non-singular if and only if* $\det(I + \Delta_{S,S}N_{S,S}) \neq 0$.

2. *If $\tilde{M}$ is non-singular then $\tilde{M}^{-1} = N - N_{*,S}(I + \Delta_{S,S}N_{S,S})^{-1}\Delta_{S,S}N_{S,*}$.*

3. *Restricting $\tilde{M}^{-1}$ to a subset T, we have*

$$\tilde{M}^{-1}{}_{T,T} = N_{T,T} - N_{T,S}(I + \Delta_{S,S}N_{S,S})^{-1}\Delta_{S,S}N_{S,T},$$

which can be computed in $O(|T|^\omega)$ time for $|T| \geq |S|$.

Suppose we already have Y and Y^{-1}, this implies that checking whether e can be removed can be done in constant time by Theorem 2.16(1). Note that we also need to update Y^{-1} for future queries, and therefore we use a recursive procedure so that edges within a subset are removed consecutively, so that the relevant entries in the inverse can be computed more efficiently using Theorem 2.16(3).

The algorithm is shown in Algorithm 3.2. Let R and C be the indices of a subset of rows and a subset of columns of Y, and $S = R \cup C$. For each edge $e = uv$, the corresponding x_e appears only in Y_{T_e,T_e} where $T_e = \{2u-1, 2u, 2v-1, 2v\}$. Procedure Remove$(R, C)$ will try to remove all edges $e = uv$ with $T_e \subseteq S$. In the base case when $|R| = |C| = 2$, we can determine whether x_e can be eliminated or not by Theorem 2.16(1) in constant time. Otherwise, when $|R| = |C| > 2$, we partition R and C into R_1, R_2 and C_1, C_2, such that first(second) half of R goes to $R_1(R_2)$, and C is also partitioned in the same way. And then we recursively call Remove(R_i, C_j) for $i, j \in \{1, 2\}$. Note that before entering into any smaller area during the recursion, we need to update Y^{-1}, but only updating $Y^{-1}{}_{S,S}$ is enough for the checkings in Remove(R_i, C_j) by Theorem 2.16(1), and this can be done in $O(|S|^\omega)$ time using Theorem 2.16(3).

Correctness: The algorithm is correct because every pair is checked, and when a pair is checked the relevant entries in the inverse are always updated. Consider an instance of Remove on rows R and columns C and let $S = R \cup C$. We keep the invariant $N_{S,S} = Y^{-1}{}_{S,S}$. After each recursive call Remove(R_i, C_j) for $i, j \in \{1, 2\}$, only the entries in $Y_{S,S}$ have been changed, denoted by $\Delta_{S,S}$. By Theorem 2.16(3), $N_{S,S}$ can be updated by $N_{S,S} - N_{S,S}(I + \Delta_{S,S}N_{S,S})^{-1}\Delta_{S,S}N_{S,S}$, which can be done in $O(|S|^\omega)$ time. When a base case is reached, by Theorem 2.16(1), an indeterminate x can be removed if and only if $\det(I + \Delta_{S,S}N_{S,S}) \neq 0$, which can

be checked in constant time since $|S| = 4$. The analysis of the failure probability is the same as in Section 3.3.1 and we omit it here.

Time Complexity: Let $f(n)$ be the time required by REMOVE, where $n = |R| = |C|$. From Algorithm 3.2 we have $f(n) = 4f\big(n/2\big) + O(n^\omega)$. Hence we have $f(n) = O(n^\omega)$ by the master theorem [15]. The initialization also takes time $O(n^\omega)$, and so the overall time complexity is $O(n^\omega)$.

Algorithm 3.2 An algebraic algorithm for disjoint $\mathcal{S}$-paths

```
SPATH(M)
  Construct Y and assign random values to each indeterminate x_e for e ∈ E
  Compute N := Y^{-1} by a fast inverse algorithm
  REMOVE({1..2n}, {1..2n})
  return all remaining pairs

REMOVE(R, C)
  Let S = R ∪ C
  Invariant: N_{S,S} = Y^{-1}_{S,S}
  if |R| = |C| = 2 then
    Let e = uv be the edge (if exists) with S = {2u − 1, 2u, 2v − 1, 2v}
    Let x_e and b'_e, c'_e be the indeterminate and the vectors associated with e
    Set Y' = Y − x_e(b'_e ∧ c'_e)
    Check if Y' is non-singular by the small area update formula (Theorem 2.16(1))
    if Y' is non-singular then
      Remove e and set Y = Y'
  else
    Partition R and C into two equal-size subsets
    for all pairs i, j ∈ {1, 2} do
      REMOVE(R_i, C_j)
      Compute N_{S,S} = Y^{-1}_{S,S} by the small area update formula (Theorem 2.16(3))
```

3.4.2 Graphic Matroid Parity

In this problem we are given an undirected graph and some edge pairs, and the problem is to find a maximum collection of edge pairs such that the union of these edges forms a forest. In some applications for graphic matroid parity, each of the

given edge pair has a common vertex. We will first show an $O(n^3)$ time algorithm for this special case, followed by an $O(n^4)$ time algorithm for the general case.

Construct the matrix Y using the compact formulation in Theorem 2.9. Since the matroid is graphic, there are only two nonzero entries in each b_i and c_i. Let each b_i and c_i be written in the form $\vec{e}_j - \vec{e}_k$ and $\vec{e}_u - \vec{e}_v$ where jk is one edge and uv is another edge. It is easy to see that each pair of elements affects at most 8 entries in Y, and thus the small area update formula can be used. Similar to previous sections, we use a recursive approach to enumerate each edge pair. For each pair our algorithm checks if some parity basis survives after removal of such pair. Recall that a parity basis exists if and only if its corresponding matrix formulation Y is of full rank. Removing a pair is done by assigning corresponding x_i to zero. Since x_i affects at most 8 entries, this can be checked in constant time by Theorem 2.16(1) using Y^{-1}. If Y remains full rank after setting x_i to zero, we remove such pair. When the algorithm terminates, the remaining pairs forms a parity basis.

We first consider the special case where each edge pair has a common vertex, where we can obtain a speedup over the general graphic matroid parity problem. The algorithm is shown in Algorithm 3.3. Define procedure $\textsc{Remove}(P, R, C)$ to check all edge pairs (i, j), (i, k) that have $i \in P$, $j \in R$ and $k \in C$. Consider the base case where $|P| = |R| = |C| = 1$. We need to determine whether pair (i, j),(i, k) ($i \in P$, $j \in R$, $k \in C$) can be removed. Since removal of such pair will only affect entries in $Y_{S,S}$ where $S = P \cup R \cup C$, decision can be made using Theorem 2.16(1) in constant time using ${Y^{-1}}_{S,S}$.

The algorithm start with $\textsc{Remove}(V, V, V)$, $V = \{1..n\}$, which will check all edge pairs. The procedure simply calls recursions when it does not reach its base cases yet. For any set T, define its first (second) half by T_1 (T_2). Then the procedure can be implemented by recursive call to $\textsc{Remove}(P_x, R_y, C_z)$ for all $x, y, z \in \{1, 2\}$. Since inverse of Y is required to decide if a pair can be removed, ${Y^{-1}}_{S,S}$ ($S = P \cup R \cup C$) is recomputed before each recursive call using Theorem 2.16(3), as in the algorithm for the $\mathcal{S}$-path problem.

Now we analyze the time complexity of this algorithm. Any changes done by $\textsc{Remove}(P, R, C)$ is made to $Y_{S,S}$ where $S = P \cup R \cup C$. So, similar to that

in the $\mathcal{S}$-path problem, updating ${Y^{-1}}_{S,S}$ using Theorem 2.16(3) takes $O(|S|^\omega)$ time. Let $f(n)$ be time required by REMOVE where $n = |P| = |R| = |C|$. We have $f(n) = 8f(n/2) + O(n^\omega)$. By the master theorem [15], if fast matrix multiplication is used, this algorithm has overall time complexity $O(n^3)$, otherwise its time complexity is $O(n^3 \log n)$ time. The analysis of the failure probability is the same as in Section 3.3.1 and we omit it here.

For the general case where edge pairs are in the form (i, k) and (j, l). Our algorithm is very similar but the procedure is now defined as REMOVE(P, Q, R, C) which checks all pairs in the form $i \in P$, $j \in Q$, $k \in R$ and $l \in C$. Hence we now require 16 recursion calls of REMOVE(P_w, Q_x, R_y, C_z) where $w, x, y, z \in \{1, 2\}$. This gives an $O(n^4)$ time algorithm by the master theorem.

3.4.3 Colorful Spanning Tree

Given an connected undirected multigraph $G = (V, E)$ where each edge is colored by one of the $k \leq n$ colors. The colorful spanning tree problem [63] is to determine if there is a spanning tree T in G such that each edge in T has a distinct color. Let $n = |V|$ and $m = |E|$.

The tree constraint can be captured by a graphic matroid M_1, where an edge (u, v) is represented by a column vector $\vec{e}_u - \vec{e}_v$. So a set of edges are acyclic if and only if their vectors for M_1 are independent. The distinct color constraint can be modeled by a partition matroid M_2, where an edge with color d is represented by a column vector $\vec{e}_d$. So a set of edges have distinct colors if and only if their vectors for M_2 are independent.

Thus a maximum cardinality common independent set of M_1 and M_2 gives a maximum size acyclic colorful subgraph of G. In particular when $k = n - 1$ and G is connected, a common basis of the two matroids is a colorful spanning tree of G. We show how to reduce the problem to the linear matroid parity problem. For each edge $e = (u, v)$ with color d, construct a column pair as follows. Let b_e and c_e have size $2n \times 1$. Set the first half of b_e to be $\vec{e}_u - \vec{e}_v$, the second half of c_e to be $\vec{e}_d$, and $\vec{0}$ for remaining half of b_e and c_e. Now each b_i is independent with any c_j. Hence a parity set is independent if and only if the corresponding edges

Algorithm 3.3 An algebraic algorithm for graphic matroid parity, when each edge pair has a common vertex.

GraphicParity(M)
  Construct Y and assign random values to indeterminates x_i
  $N := Y^{-1}$
  Remove($\{1..n\}, \{1..n\}, \{1..n\}$)
  return all remaining pairs

Remove(P, R, C)
  Let $S = P \cup R \cup C$
  Invariant: $N_{S,S} = {Y^{-1}}_{S,S}$
  if $|P| = |R| = |C| = 1$ **then**
    Let $i \in P$, $j \in R$, $k \in C$
    Let x, b, c be the indeterminate and the vectors associated with edge pair (i, j) and (i, k) (if exists)
    $Y' = Y - x(b \wedge c)$
    Check if Y' is non-singular by the small area update formula (Theorem 2.16(1))
    if Y' is non-singular **then**
      Remove this edge pair and set $Y = Y'$
  else
    Partition P, R and C into two equal-size subsets
    for all tuples $i, j, k \in \{1, 2\}$ **do**
      Remove(P_i, R_j, C_k)
      Compute $N_{S,S} = {Y^{-1}}_{S,S}$ using the small area update formula (Theorem 2.16(3))

form a colorful subgraph.

The idea of the algorithm is to examine each edge e one by one, and see if any parity basis (that is a colorful spanning tree) remains after removal of this edge. We construct Y as described in Theorem 2.9. Then each x_i will only appear in four entries because $b_i c_i^T$ has only two non-zero entries. Let Y' be the new matrix with x_i assigned to zero, which is equivalent to remove edge e_i. Let $S = \{u, v, d+n\}$, Y' is identical to Y except $Y'_{S,S}$. Recall that we can remove edge e_i if the rank of Y' remains the same. If so we simply remove that edge and update Y^{-1}. After checking all edges a parity basis remains. If the size of the parity basis is $n-1$, then it is a colorful spanning tree.

One technical point is that we require Y to have full rank before the checking starts. In our problem the originally constructed matrix Y is never of full rank. So we need another matrix that gives the same result as Y while having full rank. We will describe in Section 3.7 how to find such a matrix in Section 3.7. Henceforth we assume that Y is of full rank.

The algorithm is shown in Algorithm 3.4, which is similar to that for the graphic matroid parity problem. Let R be subset of rows of Y, C and C' be subset of columns of Y. Define procedure $\textsc{Remove}(R, C, C')$, which tries to remove edges connecting u and v having color d that have $(d+n) \in R$, $u \in C$, $v \in C'$. $\textsc{Remove}(R, C, C')$ has $|R| = |C| = |C'| = 1$ as base case, where we have to determine whether a particular edge (u, v) having color d can be removed ($(d+n) \in R$, $u \in C$, $v \in C'$). This can be done in constant time using Theorem 2.16(1) because removing such edge only affect four entries in Y. In other cases, R, C and C' are partitioned into R_1, R_2, C_1, C_2 and C'_1, C'_2. All eight smaller cases $\textsc{Remove}(R_i, C_j, C'_k)$ will be called, where $i, j, k \in \{1, 2\}$. After any recursive call Y^{-1} is updated using Theorem 2.16(3). Let $S = R \cup C \cup C'$, any instance of $\textsc{Remove}(R, C, C')$ triggers updates to $Y_{S,S}$. The updating process takes only $O(|S|^\omega)$ time.

Time Complexity: Let $f(n)$ be the time required by $\textsc{Remove}$, where $n = |R| = |C| = |C'|$. We have $f(n) = 8f(n/2) + O(n^\omega)$. Hence $f(n) = O(n^3)$ by the master theorem. As a result, the algorithm has time complexity $O(n^3)$. If fast matrix multiplication is not used, then the algorithm has time complexity

$O(n^3 \log n)$ again by the master theorem.

Algorithm 3.4 An algorithm to compute colorful spanning tree

```
ColorfulSpanningTree(G)
  Construct Y and assign random values to indeterminates x_i
  Compute N := Y^{-1} by fast inverse
  Remove({1..2n}, {1..2n}, {1..2n})
  return all remaining edges

Remove(R, C, C')
  Let S = R ∪ C ∪ C'
  Invariant: N_{S,S} = Y^{-1}_{S,S}
  if |R| = |C| = |C'| = 1 then
    Let (d + n) ∈ R, u ∈ C, v ∈ C'
    Let x, b, c be the indeterminate and the vectors associated with edge (u, v)
    with color d (if exists)
    Y' = Y − x(b ∧ c)
    Check if Y' is non-singular by the small area update formula (Theo-
    rem 2.16(1))
    if Y' is non-singular then
      Remove this edge and set Y = Y'
  else
    Partition R, C and C' into two equal-size subsets
    for all tuples i, j, k ∈ {1, 2} do
      Remove(R_i, C_j, C'_k)
      Compute N_{S,S} = Y^{-1}_{S,S} using the small area update formula (Theo-
      rem 2.16(3))
```

3.5 Weighted Linear Matroid Parity

In the weighted matroid parity problem, each pair i is assigned a non-negative weight w_i, and the objective is to find a parity basis with maximum weight. In Theorem 2.11 we saw Camerini et al. gave a compact matrix formulation for this problem. Consider the matrix $Y^* = \sum_{i=1}^{m} x_i(b_i \wedge c_i)y^{w_i}$. The theorem says each term of $\operatorname{pf} Y^*$ corresponds to a parity basis, and the degree of y in that term corresponds to the weight to the parity basis.

This gives an algorithm for the weighted problem similar to Algorithm 3.1, but we need to make sure that some parity basis with maximum weight is preserved. If the maximum weighted parity basis have weight p, we want to calculate $\mathrm{pf}\, Y^*$ to see if a term containing y^p still remains. It can be achieved by finding $\det(Y^*)$ and look for a term that contains y^{2p} because $\det(Y^*) = \mathrm{pf}(Y^*)^2$ However calculating its determinant may not be easy. Define $W = \max_i\{w_i\}$. Camerini et al. [9] proposed an $\tilde{O}(m^2r^2 + Wmr^4)$ algorithm to find the parity basis with maximum weight.

Now we show we can apply Theorem 2.21 by Storjohann to give a faster algorithm. The theorem says that given a $n \times n$ polynomial matrix of degree d then its determinant can be computed in $\tilde{O}(n^\omega d)$ time. By choosing a large enough field $\mathbb{F}$, we can check if removing each pair i (by assigning $x_i = 0$) would affect the parity basis with maximum weight, with high probability. If the degree of $\det(Y^*)$ does not drop after removal of a pair, then it can be removed. And removal of a pair can be simply done by an update to Y^* in $O(r^2)$ time. Each time the checking can be done in $\tilde{O}(Wr^\omega)$ time by Theorem 2.21, which dominates the updating time. Calculating Y^* at the beginning takes $O(mr^2)$ time. Hence we can find a parity basis with the maximum weight in $\tilde{O}(Wmr^\omega)$ time. The pseudocode of the algorithm can be found in Algorithm 3.5.

Algorithm 3.5 An algebraic algorithm for weighted linear matroid parity

```
MaxWeightParity(M)
  Construct Y* and assign random values to indeterminates x_i
  J := {b_1, c_1, ··· b_m, c_m}
  for i = 1 to m do
    Y := Y* − x_i(b_i ∧ c_i)y^{w_i}
    if degree of det(Y) equals that of det(Y*) then
      Y* := Y
      J := J − {b_i, c_i}
  return J
```

Finally we describe how to obtain a randomized fully polynomial-time approximation scheme using the pseudo-polynomial algorithm by standard scaling technique [59]. Here we assume every column pair is contained in some parity basis. This assumption can be satisfied by checking the rank of corresponding $Z(J)$ as

in Theorem 3.7. If $Z(J)$ is not full rank we discard the corresponding column pair.

The idea is to scale the weight of each pair down and solve the new instance using Algorithm 3.5. Given $\epsilon > 0$, let $K = \epsilon W/r$. For each pair i scale its weight to $w_i^* = \lfloor w_i/K \rfloor$. Solve the new instance using Algorithm 3.5. This takes $\tilde{O}(\lfloor W/K \rfloor mr^\omega) = \tilde{O}(mr^{\omega+1}/\epsilon)$ time.

We will show the result of the scaled instance is an $(1-\epsilon)$-approximation of the original instance. Let J and O be pairs return by the above algorithm and the optimal pairs respectively. Also denote original (scaled) weight of a set S by $w(S)$ ($w^*(S)$). We have $w(O) - Kw^*(O) \leq rK$ because for each pair in O at most weight with value K is lost, and there are at most $r/2$ pairs chosen. Then

$$w(J) \geq K \cdot w^*(J) \geq K \cdot w^*(O) \geq w(O) - rK = w(O) - \epsilon W \geq (1-\epsilon) \cdot w(O).$$

3.6 A Faster Linear Matroid Parity Algorithm

In this section we present an $O(mr^{\omega-1})$-time randomized algorithm for the linear matroid parity problem. We first consider the problem of determining whether M has a parity basis, and show how to reduce the general problem into it in Section 3.7. The algorithm is very similar to the algebraic algorithm for linear matroid intersection by Harvey [30]. The general idea is to build a parity basis incrementally. A subset of pairs is called *growable* if it is a subset of some parity basis. Starting from the empty solution, at any step of the algorithm we try to add a pair to the current solution so that the resulting subset is still growable, and the algorithm stops when a parity basis is found.

3.6.1 Matrix Formulation

We use the matrix formulation of Geelen and Iwata in Theorem 2.10.

$$Z := \begin{pmatrix} 0 & M \\ -M^T & T \end{pmatrix}$$

Then we have $\nu_M = r/2$ if and only if Z is of full rank. To determine whether a subset J of pairs is growable, we define $Z(J)$ to be the matrix that have $t_i = 0$ for all pair i in J. We define $\nu_{M/J}$ to be the optimal value of the linear matroid parity problem of M/J, which is the contraction of M by J as stated in Section 2.1. Informally the linear matroid parity problem of M/J corresponds to the linear matroid parity problem of M when the pairs in J are picked. In the following we will show that, following from the Geelen-Iwata formula, that J is growable if and only if $Z(J)$ is of full rank.

Theorem 3.6. *For any independent parity set J,* $\operatorname{rank}(Z(J)) = 2\nu_{M/J} + 2m + |J|$.

Proof. In the following let R be the set of rows of M and V be the set of columns of M (i.e. $|V| = 2m$). Note that $Z(J)$ is in the following form.

$$\underbrace{\begin{array}{c} \\ R \\ J \\ V \setminus J \end{array}\begin{array}{c} \begin{array}{ccc} R & J & V \setminus J \end{array} \\ \begin{pmatrix} & M_{R,J} & M_{R,V\setminus J} \\ (-M^T)_{J,R} & & \\ (-M^T)_{V\setminus J,R} & & T_{V\setminus J,V\setminus J} \end{pmatrix} \end{array}}_{Z(J)}$$

$$= \underbrace{\begin{array}{c} \\ R \\ J \\ V \setminus J \end{array}\begin{array}{c} \begin{array}{ccc} R & J & V \setminus J \end{array} \\ \begin{pmatrix} & M_{R,J} & M_{R,V\setminus J} \\ (-M^T)_{J,R} & & \\ (-M^T)_{V\setminus J,R} & & \end{pmatrix} \end{array}}_{Q} + \underbrace{\begin{array}{c} \\ R \\ J \\ V \setminus J \end{array}\begin{array}{c} \begin{array}{ccc} R & J & V \setminus J \end{array} \\ \begin{pmatrix} & & \\ & & \\ & & T_{V\setminus J,V\setminus J} \end{pmatrix} \end{array}}_{\tilde{T}}.$$

By Lemma 2.19, we have

$$\operatorname{rank}(Z(J)) = \max_{A \subseteq S} \{\operatorname{rank}(Q_{A,A}) + \operatorname{rank}(\tilde{T}_{S\setminus A, S\setminus A})\} \tag{3.6.1}$$

where $S = R \cup V$ is the column set and row set for $Z(J)$. Consider a set A that maximize $\operatorname{rank}(Z(J))$, then A must be in the form $R \cup A'$ where $J \subseteq A' \subseteq V$.

Recall that a set is a parity set if every pair is either contained in it or disjoint from it. We can assume that A' is a parity set. If A' is not, consider parity set B' such that $A' \subseteq B'$ and B' has smallest size. Let $B = R \cup B'$, we have $\text{rank}(Q_{A,A}) \leq \text{rank}(Q_{B,B})$ and $\text{rank}(\tilde{T}_{S\setminus A,S\setminus A}) = \text{rank}(\tilde{T}_{S\setminus B,S\setminus B})$ where the equality follows from the structure of T.

Since M and $-M^T$ occupy disjoint rows and columns of Q, and M is skew-symmetric,

$$\begin{aligned}\text{rank}(Q_{A,A}) &= \text{rank}(M_{R,A'}) + \text{rank}((-M^T)_{A',R}) \\ &= 2\,\text{rank}(M_{R,A'}).\end{aligned}$$

We also have $\text{rank}(\tilde{T}_{S\setminus A,S\setminus A}) = |S \setminus A| = |V \setminus A'| = |V| - |A'|$. Putting these to (3.6.1) we have

$$\text{rank}(Z(J)) = \max_{A'\subseteq V}\{2\,\text{rank}(M_{R,A'}) + |V| - |A'|\}.$$

Write $A' = I \cup J$ where $I \cap J = \emptyset$ and I is a parity set. By the rank function $r_{M/J}$ of the matroid M/J, we have $r_{M/J}(A' \setminus J) = r_M(A') - r_M(J)$. Hence

$$\begin{aligned}\text{rank}(Z(J)) &= \max_{I\subseteq V\setminus J}\{2(r_{M/J}(I) + |J|) + |V| - (|I| + |J|)\} \\ &= \max_{I\subseteq V\setminus J}\{2r_{M/J}(I) - |I|\} + |V| + |J| \qquad (3.6.2)\end{aligned}$$

Observe that a maximizer I' of (3.6.2) must be an independent parity set (so $r_{M/J}(I') = |I'|$); otherwise an independent set $K \subset I'$ such that $r_{M/J}(K) = r_{M/J}(I')$ and $|K| < |I'|$ gives a larger value for (3.6.2). So a maximizer I' of (3.6.2) would maximize $r_{M/J}(I)$, which implies that I' is indeed a maximum cardinality parity set of M/J. The result follows since $2\nu_{M/J} = |I'| = r_{M/J}(I')$. □

Theorem 3.7. *For any independent parity set J, $Z(J)$ is non-singular if and only if J is growable.*

Proof. If $Z(J)$ is non-singular, by Theorem 3.6 we have

$$\begin{aligned} \operatorname{rank}(Z(J)) &= 2\nu_{M/J} + |V| + |J| \\ 2m + r &= 2\nu_{M/J} + 2m + |J| \\ r &= 2\nu_{M/J} + |J| \end{aligned}$$

Hence J is growable. □

3.6.2 An $O(m^\omega)$ Algorithm

The algorithm here maintains a growable set J, starting with $J = \emptyset$. To check whether a pair i can be added to J to form a growable set, we test whether $Z(J \cup \{2i-1, 2i\})$ is of full rank. Observe that $Z(J \cup \{2i-1, 2i\})$ is obtained from $Z(J)$ by a small area update, and so Theorem 2.16 can be used to check whether $Z(J \cup \{2i-1, 2i\})$ is of full rank more efficiently. Pseudocode of the algorithm is shown in Algorithm 3.6.

First we show how to check whether a pair can be included to J to form a larger growable set.

Claim 3.8. *Let $N = Z(J)^{-1}$, $n_i = N_{2i-1,2i}$ and $J' = J \cup \{2i-1, 2i\}$. Then J' is a growable set if and only if $t_i n_i + 1 \neq 0$.*

Proof. By Theorem 3.7, J' is growable if and only if $Z(J')$ is non-singular. By Theorem 2.16(1), this is true if and only if the following expression is non-zero.

$$\det\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & t_i \\ -t_i & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & n_i \\ -n_i & 0 \end{pmatrix}\right) = \det\begin{pmatrix} t_i n_i + 1 & 0 \\ 0 & t_i n_i + 1 \end{pmatrix}$$

Thus $Z(J')$ is non singular if and only if $(t_i n_i + 1)^2 \neq 0$, which is equivalent to $t_i n_i + 1 \neq 0$. □

Correctness: At the time MatroidParity call BuildParity, the invariant $N = Z(J)^{-1}$ obviously holds, and so as the first recursive call to BuildParity. Regardless the changes made in the first recursive call, $Z(J \cup J_1)^{-1}$ is recomputed

Algorithm 3.6 An $O(m^\omega)$-time algebraic algorithm for linear matroid parity

```
MatroidParity(M)
  Construct Z and assign random values to indeterminates t_i
  Compute N := Z^{-1} by fast matrix inverse
  return BuildParity(S, N, ∅)

BuildParity(S,N,J)
  Invariant 1: J is a growable set
  Invariant 2: N = Z(J)^{-1}_{S,S}
  if |S| = 2 then
    Let S = {2i − 1, 2i}
    if 1 + t_i N_{2i−1,2i} ≠ 0 then
      return {2i − 1, 2i}
    else
      return ∅
  else
    Partition S into two equal-size subsets
    J_1 := BuildParity(S_1, N_{S_1,S_1}, J)
    Compute M := Z(J ∪ J_1)^{-1}_{S_2,S_2} using Claim 3.9
    J_2 := BuildParity(S_2, M, J ∪ J_1)
    return J_1 ∪ J_2
```

so the invariant is also satisfied with the second recursive call. Note S is partitioned in such a way that its first half goes to S_1 and the remaining goes to S_2, so both S_1 and S_2 must be parity set.

In the algorithm every element of M is considered. By Claim 3.8, $Z(J)$ is always non-singular. Hence, by Theorem 3.7, this implies that J is always a growable set.

Time complexity: In the following claim we show how to compute $M := Z(J \cup J_1)^{-1}{}_{S_2,S_2}$ efficiently.

Claim 3.9. *Let $J' = J \cup J_1$, $J_1 \subseteq S$. Using $N = Z(J)^{-1}{}_{S,S}$, computing $Z(J')^{-1}{}_{S,S}$ can be done in $O(|S|^\omega)$ time.*

Proof. Since $Z(J')$ is identical to $Z(J)$ except $Z(J')_{J_1,J_1} = 0$, since $Z(J)^{-1}{}_{R,C} = N_{R,C}$, by Theorem 2.16(3),

$$\begin{aligned} Z(J')^{-1}{}_{S,S} &= N_{S,S} - N_{S,J_1}(I + (-Z(J)_{J_1,J_1})N_{J_1,J_1})^{-1}(-Z(J)_{J_1,J_1})N_{J_1,S} \\ &= N_{S,S} + N_{S,J_1}(I - Z_{J_1,J_1}N_{J_1,J_1})^{-1}Z_{J_1,J_1}N_{J_1,S} \end{aligned}$$

Note the last equality holds because $Z(J)_{J_1,J_1} = Z_{J_1,J_1}$. At any time during the computation, matrices involved have size at most $|S| \times |S|$. Hence computing $Z(J')^{-1}{}_{S,S}$ takes $O(|S|^\omega)$ time. □

Since Z has dimension $(2m + r) \times (2m + r)$, initial computation of $Z^{-1}{}_{S,S}$ takes $O((2m + r)^\omega) = O(m^\omega)$ time. Let $f(m)$ be the time required by BuildParity with $|S| = 2m$, then

$$f(m) = 2 \cdot f(m/2) + O(m^\omega)$$

which implies $f(n) = O(m^\omega)$ by the master theorem.

3.6.3 An $O(mr^{\omega-1})$ Algorithm

The previous algorithm works for matroids with large rank. In this section we present an algorithm with better time complexity when rank is small. The idea

behind is to break the ground set S into a number of smaller pieces. In this way, inverse of these matrices to be computed will have smaller size.

Algorithm 3.7 An $O(mr^{\omega-1})$-time algebraic algorithm for linear matroid parity

```
MatroidParity(M)
  Construct Z and assign random values to indeterminates t_i
  Compute Y := MT^{-1}M^T using Claim 3.10
  Parition S into m/r subsets each with size r
  J := ∅
  for i = 1 to m/r do
    Compute N := Z(J)^{-1}_{S_i,S_i} using Claim 3.12
    J' := BuildParity(S_i, N, J)
    J := J ∪ J'
  return J
```

To achieve this we make use of Schur's formula (Theorem 2.14). Denote $Y = MT^{-1}M^T$ as the Schur complement of T in Z. By Theorem 2.14,

$$Z^{-1} = \begin{pmatrix} Y^{-1} & -Y^{-1}MT^{-1} \\ T^{-1}M^TY^{-1} & T^{-1} - T^{-1}M^TY^{-1}MT^{-1} \end{pmatrix} \tag{3.6.3}$$

We will assume Y^{-1} exists; otherwise, if Y has no inverse, then we can conclude that Z has no inverse by Theorem 2.14 and thus there is no parity basis. In the following we will show how to compute Y efficiently, and then show how to compute $Z(J)^{-1}{}_{S_i,S_i}$ efficiently.

Claim 3.10. *Let $Y = MT^{-1}M^T$, Y^{-1} can be computed in $O(mr^{\omega-1})$ time.*

Proof. First we show that $MT^{-1}{}_{R,C}$ can be computed in $O(RC)$ time. Recall that T is a skew-symmetric matrix, having exactly one entry in each row and column. Moreover, the positions of the non-zero entries in T are just one row above or below the diagonal of T. It is thus easy to compute T^{-1}. If $T_{i,j}$ is zero, then $T^{-1}{}_{i,j}$ is also zero. Otherwise $T^{-1}{}_{i,j} = -1/T_{i,j}$. As a result T^{-1} is also skew-symmetric and shares the same special structure of T. Therefore any entry of MT^{-1} can be computed in $O(1)$ time. Hence $MT^{-1}{}_{R,C}$ takes $O(RC)$ to compute.

Now we are going to show $MT^{-1}M^T$ can be computed in $O(mr^{\omega-1})$ time. Since M has size $r \times m$, MT^{-1} can be computed in $O(mr)$ time. To compute product of MT^{-1} (size $r \times m$) and M^T (size $m \times r$), we can break each of them into m/r matrices each of size $r \times r$, so computation of their products takes $O(mr^{\omega-1})$.

Finally Y is of size $r \times r$ and its inverse can be computed in $O(r^\omega)$ time. □

Next we show how to compute ${Z(J)^{-1}}_{S_i,S_i}$ efficiently using Y^{-1}.

Claim 3.11. *Given the matrix Y^{-1}, for any $A, B \subseteq S$ with $|A|, |B| \leq r$, ${Z^{-1}}_{A,B}$ can be computed in $O(r^\omega)$ time.*

Proof. By Equation 3.6.3,

$$ {Z^{-1}}_{S,S} = T^{-1} - T^{-1}M^T Y^{-1} M T^{-1} $$

Hence for any $A, B \subseteq S$,

$$ {Z^{-1}}_{A,B} = {T^{-1}}_{A,B} - (T^{-1}M^T)_{A,*} Y^{-1} (MT^{-1})_{*,B} $$

Both $(T^{-1}M^T)_{A,*}$ and $(MT^{-1})_{*,B}$ have size $r \times r$ and can be computed in $O(r^2)$ time by Claim 3.10. Thus the whole computation takes $O(r^\omega)$ time. □

Claim 3.12. *In each loop iteration, the matrix ${Z(J)^{-1}}_{S_i,S_i}$ can be computed in $O(r^\omega)$ time.*

Proof. Since $Z(J)$ is identical to Z except $Z(J)_{J,J} = 0$, by using Theorem 2.16(3) with $\tilde{M}_{S_i,S_i} - M_{S_i,S_i} = -Z(J)_{J,J}$, we have

$$ {Z(J)^{-1}}_{S_i,S_i} = {Z^{-1}}_{S_i,S_i} + {Z^{-1}}_{S_i,J} (I - Z_{J,J} {Z^{-1}}_{J,J})^{-1} Z_{J,J} {Z^{-1}}_{J,S_i} $$

All the submatrices ${Z^{-1}}_{S_i,S_i}$, ${Z^{-1}}_{S_i,J}$, ${Z^{-1}}_{J,J}$ and ${Z^{-1}}_{J,S_i}$ can be computed in $O(r^\omega)$ by Claim 3.11. Thus the whole computation can be done in $O(r^\omega)$ time. □

Since $|S_i| = r$, each call to BuildParity takes $O(r^\omega)$ time. Hence the overall time complexity of Algorithm 3.7 is $O(m/r \cdot r^\omega) = O(mr^{\omega-1})$.

3.7 Maximum Cardinality Matroid Parity

The algorithms in previous sections can only produce a parity basis if one exists. If there is no parity basis, these algorithms are only be able to report so. In this section, we present how to find the maximum number of pairs that are independent. We are going to show an $O(r^\omega)$ time reduction, that reduce a maximum cardinality matroid parity problem to a problem of computing parity basis. Hence algorithms in previous sections can be applied.

The idea here is to find a maximum rank submatrix of the matrix formulation for matroid parity. Such a submatrix is of full rank and corresponds to a new instance of matroid parity problem which has a parity basis.

Let Y be matrix formulation for the parity problem constructed as in Theorem 2.9. Let r' be the rank of Y. We first find a maximum rank submatrix $Y_{R,C}$ of Y where $|R| = |C| = r'$. This can be done in $O(r^\omega)$ time using a variant of the LUP decomposition algorithm by Harvey (Appendix A of [29]). Since Y is a skew symmetric matrix, $Y_{R,R}$ is also a maximum rank submatrix of Y (Theorem 2.6).

The matrix $Y_{R,R}$ can be interpreted as matrix formulation for a new matroid parity instance. Such an instance contains all the original given pairs, but only contains rows indexed by R. Then

$$\sum_{i=1}^{m} x_i(b_{i_R} \wedge c_{i_R}) = Y_{R,R},$$

where b_{i_R} denotes the vector containing entries of b_i index by R. Since $Y_{R,R}$ is of full rank, such a new instance of matroid parity has a parity basis.

The column pairs that are independent in the new instance are also independent in the original instance. Hence a parity basis of this new instance corresponds to a parity set of the original instance. In addition, this new instance for matroid parity can be solved using any algorithm presented.

Since Algorithm 3.7 has time complexity $O(mr^{\omega-1})$, we want this reduction to be done under the same time, but a naive construction of Y would take $O(mr^2)$ time. In the proof of Theorem 2.9, we have shown that the matrix formulation

Y is equivalent to another matrix Z in Theorem 2.10. In addition, we have $Y = MT^{-1}M^T$ which can be computed in $O(mr^{\omega-1})$ time by Claim 3.10.

3.8 Open Problems

There are some interesting open problems remained. When we solve graphs problems through matroid parity we rely on the fact that each column has one or two non-zero entries. A question to ask is whether we can give a more efficient algorithm if each row has only a few non-zero entries. A positive answer will imply a faster algorithm for the minimum pinning set problem (Section 2.1.4).

Another problem is the weighted linear matroid parity problem. We have shown a faster pseudo-polynomial time algorithm with time complexity $\tilde{O}(Wmr^{\omega})$. There is a gap between this and the $O(W^{1+\varepsilon}mr^{\omega-1})$ algebraic algorithm for linear matroid intersection. Can this gap be closed? Of course the bigger question to ask is whether there exists a truly polynomial time algorithm for the weighted linear matroid parity problem.

We gave faster algorithms for some applications of linear matroid parity. But there remain many important applications of linear matroid parity, for example the spanning tree packing problem. Can we design algebraic algorithms that are faster than the best known combinatorial algorithms?

Chapter 4

Graph Connectivities

The results presented in this chapter are based on joint work with Lap Chi Lau and Kai Man Leung [12].

4.1 Introduction

Graph connectivity is a basic concept that measures the reliability and efficiency of a graph. The edge connectivity from vertex s to vertex t is defined as the size of a minimum s-t cut, or equivalently the maximum number of edge disjoint paths from s to t. Computing edge connectivities is a classical and well-studied problem in combinatorial optimization. Most known algorithms to solve this problem are based on network flow techniques (see e.g. [63]).

The fastest algorithm to compute s-t edge connectivity in a simple directed graph runs in $O(\min\{m^{1/2}, n^{2/3}\} \cdot m)$ time by Even and Tarjan [19], where m is the number of edges and n is the number of vertices. To compute the edge connectivities for many pairs, however, it is not known how to do it faster than computing edge connectivity for each pair separately, even when the pairs share the source or the sink. For instance, it is not known how to compute all pairs edge connectivities faster than computing s-t edge connectivity for $\Omega(n^2)$ pairs. This is in contrast to the problem in undirected graphs, where all pairs edge connectivities can be computed in $\tilde{O}(mn)$ time by constructing a Gomory-Hu tree [4].

In this chapter we study edge connectivities in simple directed planar graphs and its generalizations. In a recent work, we give a new algebraic formulation to the edge connectivities problem using network coding [12]. Construct an $m \times m$ matrix K as follows, if the head of e_i is equal to the tail of e_j then we set $K_{i,j} = k_{i,j}$ where $k_{i,j}$ is a distinct indeterminate, 0 otherwise. Earlier in Theorem 2.26 we have shown the matrix has the following properties:

Theorem 2.26. *The s-t edge connectivity is equal to the rank of the submatrix* $(I-K)^{-1}{}_{\delta^{out}(s),\delta^{in}(t)}$. *In addition, if we substitute indeterminates* $k_{i,j}$ *with random values from a field* $\mathbb{F}$, *then the claim still holds with probability* $1 - O(m^3/|\mathbb{F}|)$.

This formulation allows us to compute edge connectivities between many vertex pairs simultaneously. It implies an $O(m^\omega)$ time algorithm for all pairs edge connectivities in general graphs as shown in Section 2.5. Thus it implies an $O(n^\omega)$ time algorithm for planar graphs, bounded genus graphs and fixed minor free graphs, since these graphs have $O(n)$ edges [45, 68]. This is an improvement over the current algorithms, as the edge connectivity of one pair in a simple directed planar graphs takes $O(n)$ time [6] to compute.

The bottleneck of the $O(m^\omega)$ time algorithm is the step to compute $(I-K)^{-1}$. Thus we are motivated to study in what situation can we compute $(I-K)^{-1}$ more efficiently. One approach to compute the inverse of a matrix is to apply Schur's formula (Theorem 2.14). Recall the theorem,

Theorem 2.14. *Let*

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix},$$

where A *and* D *are square matrices and denote* $S = D - CA^{-1}B$. *If* A *is non-singular then* $\det(M) = \det(A) \times \det(S)$. *Furthermore if* A *and* S *are non-singular then*

$$M^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}.$$

Our observation is that for graphs with good separators, we can find a partition of $(I-K)$ such that B and C are of low rank, so that we can compute $(I-K)^{-1}$ faster by a divide and conquer algorithm.

Using the observation, we come up with a randomized algorithm that compute the edge connectivities between all pairs of vertices in $O(d^{\omega-2}n^{\omega/2+1})$ time for simple directed fixed minor free graphs with maximum degree d. This implies that we can compute the edge connectivity between one pair of vertices in amortized $O(d^{0.4}n^{0.2})$ time. This method is an improvement over the $O(n^{\omega})$ time algorithm when $d = O(\sqrt{n})$.

The rest of this chapter is organized as follows. In Section 4.2, we will show that if a matrix is "well-separable" (to be defined in the next section), then its inverse can be computed more efficiently. Then in Section 4.3, we show that the matrix $I - K$ for fixed minor free graphs is well-separable, and conclude that the edge connectivities in such graphs can be computed efficiently.

4.2 Inverse of Well-Separable Matrix

We say an $n \times n$ matrix M is (r, α)-*well-separable* (α is a constant smaller than one) if M is invertible and it can be written as

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

so that both A and D are (r, α)-well-separable square matrix with dimension no more than αn, and also both B and C are of rank no more than r. In this section we are going to show that the inverse of a well-separable matrix can be computed in $O(r^{\omega-2}n^2)$ time.

To compute M^{-1}, we first compute the inverses of A and D recursively. We will compute M^{-1} using Schur's formula. A key step is to use the Sherman-Morrison-Woodbury formula (Theorem 2.15) to compute S^{-1} efficiently, instead of computing $D - CA^{-1}B$ and then $(D - CA^{-1}B)^{-1}$.

Firstly, since $\text{rank}(B), \text{rank}(C) \leq r$, we can write $B = B_u B_v^T$ and $C = C_u C_v^T$ where B_u, B_v, C_u and C_v are size $O(n) \times r$ matrices. This can be done in $O(r^{\omega-2}n^2)$ time by Theorem 2.13.

Claim 4.1. *If we have A^{-1} and D^{-1}, then we can compute $S^{-1} = (D - CA^{-1}B)^{-1}$ in $O(r^{\omega-2}n^2)$ time.*

Proof. By Schur's formula (Theorem 2.14) $S = D - CA^{-1}B$ is non-singular if M^{-1} and A^{-1} exist. The proof consists of two steps. We will first write $CA^{-1}B$ as a product of two low rank matrices. Then we use the Sherman-Morrison-Woodbury formula to compute S^{-1} efficiently. In the following, rectangular matrix multiplications will be done by dividing each matrix into submatrices of size $r \times r$. Multiplications between these submatrices can be done in $O(r^\omega)$ time.

First we consider $CA^{-1}B$, which is equal to $(CA^{-1}B_u)B_v^T$. $C(A^{-1}B_u)$ is of size $O(n) \times r$ and can be computed using $O(n^2/r^2)$ submatrices multiplications of size $r \times r$. By putting $U = CA^{-1}B_u$ and $V = B_v$, we have $CA^{-1}B = UV^T$ where U and V are of size $O(n) \times r$.

Now we can use Theorem 2.15 (Sherman-Morrison-Woodbury formula) to compute $(D + UV^T)^{-1}$ efficiently since D^{-1} is known and U, V are low rank matrices. By the formula,

$$S^{-1} = D^{-1} - D^{-1}U(I + V^T D^{-1}U)^{-1}V^T D^{-1}.$$

Similar to the above $V^T D^{-1}U$ can be computed using $O(n^2/r^2)$ submatrix multiplications. Since S is non-singular, $(I + V^T D^{-1}U)^{-1}$ exists by Theorem 2.15, and it can be computed in one submatrix multiplication time as $I + V^T D^{-1}U$ is of size $r \times r$. Finally, since $(D^{-1}U)^T$ and $(V^T D^{-1})$ are $r \times O(n)$ matrices, we can compute $(D^{-1}U)(I + V^T D^{-1}U)^{-1}(V^T D^{-1})$ using $O(n^2/r^2)$ submatrix multiplications. Hence the overall time complexity is $O(r^\omega \times n^2/r^2) = O(r^{\omega-2}n^2)$. □

Using Claim 4.1, we can now compute M^{-1} using Schur's formula efficiently.

Claim 4.2. *If we have A^{-1}, D^{-1} and S^{-1}, then we can compute M^{-1} in $O(r^{\omega-2}n^2)$ time.*

Proof. By Schur's formula, to compute M^{-1}, we need to compute $A^{-1}BS^{-1}CA^{-1}$, $A^{-1}BS^{-1}$ and $S^{-1}CA^{-1}$. These multiplications all involve some $O(n) \times r$ matrix B_u, B_v, C_u or C_v follows:

- $A^{-1}BS^{-1}CA^{-1} = (A^{-1}B_u)(B_v^T S^{-1}CA^{-1})$
- $A^{-1}BS^{-1} = (A^{-1}B_u)(B_v^T S^{-1})$
- $S^{-1}CA^{-1} = (S^{-1}C_u)(C_v^T A^{-1})$

All steps during computation of these products involve an $O(n) \times r$ (or $r \times O(n)$) matrix, thus we can avoid computing the product of two $n \times n$ matrices. Hence they all only take $O(n^2/r^2)$ submatrix multiplications to compute. Therefore the total time complexity is $O(r^\omega \times n^2/r^2) = O(r^{\omega-2}n^2)$. □

By Claim 4.1 and Claim 4.2, we can compute M^{-1} in $O(r^{\omega-2}n^2)$ time if we are given A^{-1}, D^{-1} and also low rank decomposition of B and C. We can then use a recursive algorithm to compute M^{-1}.

Theorem 4.3. *If an $n \times n$ matrix M is (r, α)-well-separable, and the separation can be found in $O(n^\gamma)$ time, then M^{-1} can be computed in $O(n^\gamma + r^{\omega-2}n^2)$ time.*

Proof. First we analyse the time to compute M^{-1}. We use a $O(n^\gamma)$ time algorithm to find an (r, α)-well-separable partition of M. By the property of the partition, both B and C are matrices of rank at most r. Then we can write $B = B_u B_v^T$ and $C = C_u C_v^T$ in $O(r^{\omega-2}n^2)$ time where B_u, B_v, C_u and C_v are all $O(n) \times r$ matrices by Theorem 2.13. We then compute A^{-1} and D^{-1} recursively, as A and D by definition are also (r, α)-separable. Using these inverses we can apply Claim 4.1 to compute S^{-1}, then apply Claim 4.2 to compute M^{-1} using A^{-1}, D^{-1} and S^{-1}. Thus, given A^{-1} and D^{-1}, we can compute M^{-1} in $O(r^{\omega-2}n^2)$ time. Let $f(n)$ be the time to compute M^{-1} of size $n \times n$. Then

$$f(n) = f(\alpha n) + f((1-\alpha)n) + O(n^\gamma) + O(r^{\omega-2}n^2),$$

and it remains to show $f(n) = O(n^\gamma + r^{\omega-2}n^2)$.

For simplicity, we prove the case that n^γ is dominated by $r^{\omega-2}n^2$, which is the case that will happen in the next section. The another case follows with a very similar analysis. Let $f(n) = f(\alpha n) + f((1-\alpha)n) + kr^{\omega-2}n^2$ for some constant k,

We will show $f(n) \leq cr^{\omega-2}n^2$ for some large enough constant c using induction.

$$\begin{aligned} f(n) &= cr^{\omega-2}(\alpha n)^2 + cr^{\omega-2}((1-\alpha)n)^2 + kr^{\omega-2}n^2 \\ &= (c\alpha^2 + c(1-\alpha)^2 + k)r^{\omega-2}n^2 \\ &\leq cr^{\omega-2}n^2 \end{aligned}$$

□

4.3 Directed Graphs with Good Separators

In this section we show that all pairs edge connectivities in planar graphs, bounded genus graphs, and fixed minor free graphs can be computed in $O(d^{\omega-2}n^{\omega/2+1})$ time.

We will first see that the underlining matrix $I - K$ for these graphs are well separable. Thus we can apply Theorem 4.3 together with the fact the these graphs have $O(n)$ edges, to obtain a $O(d^{\omega-2}n^{\omega/2+1})$ time algorithm to compute $(I-K)^{-1}$. Finally we can apply Theorem 2.26, and then show that the time to compute the required ranks of all submatrices in $(I-K)^{-1}$ is $O(d^{\omega-2}n^{\omega/2+1})$.

We say an undirected graph $G = (V, E)$ has a $(f(n), \alpha)$-separation, if V can be partitioned into three parts, X, Y, Z such that $|X \cup Z| \leq \alpha|V|$, $|Y \cup Z| \leq \alpha|V|$, $|Z| \leq f(n)$, and there is no edge between X and Y. Planar graphs, bounded genus graphs and fixed minor free graphs all have $(O(\sqrt{n}), 2/3)$-separation which can be found in $O(n^{1.5})$ time [1]. In addition, these graphs are closed under taking subgraphs. So we can recursively separate the separated parts X and Y until the separated pieces are small enough.

For a graph G (and its subgraph) that has $O(n)$ edges and a $(O(\sqrt{n}), 2/3)$-separation, we claim that the matrix $I - K$ of G is $(O(d\sqrt{n}, \alpha)$-well-separable for some constant $\alpha < 1$. Recall that an $n \times n$ matrix is $(f(n), \alpha)$-well-separable if we can partition the matrix so that submatrices A and D are square matrices having dimension $\leq \alpha n$ and are also $(f(n), \alpha)$-well-separable, while submatrices B and D are of rank at most $f(n)$.

To show that $I - K$ is well-separable, we prove by induction on the number of edges of the graph. We will first show the inductive step and then by the base case. For the inductive step, we can use a separator to divide the graph into three parts X, Y, Z such that $|X \cup Z| \leq 2n/3$, $|Y \cup Z| \leq 2n/3$ and $|Z| = O(\sqrt{n})$, and there are no edges between X and Y. Let E_1 be the set of edges with at least one endpoint in X and let E_2 be $E - E_1$. We partition $I - K$ as follows.

$$\begin{array}{c} \\ E_1 \\ E_2 \end{array} \begin{array}{c} \begin{array}{cc} E_1 & E_2 \end{array} \\ \begin{pmatrix} A & B \\ C & D \end{pmatrix} \end{array}$$

Since $|Y| = \Omega(n)$ and each vertex in Y is of degree at least 1, we have $|E_1| = \Omega(n)$ and $|E_2| = \Omega(n)$. Also we have $|E_1| = O(n)$ and $|E_2| = O(n)$ since $m = O(n)$, and thus $|E_1|, |E_2| \leq \alpha n$ for some constant α. Let F_1 be the subset of E_1 with one endpoint in Z, and define F_2 similarly. Then any edge in $E_1 - F_1$ does not share an endpoint with any edge in $E_2 - F_2$. Since $|Z| = O(\sqrt{n})$ and each vertex is of degree at most d, there are at most $O(d\sqrt{n})$ edges with one endpoint in Z, and thus $|F_1| = O(d\sqrt{n})$ and $|F_2| = O(d\sqrt{n})$. Therefore, submatrices B and C have $O(d\sqrt{n})$ non-zero rows and columns, and thus are of rank at most $O(d\sqrt{n})$. Also $I - K$ must be invertible because $I - K$ has ones on its diagonal, and thus $\det(I - K)$ is a non-zero polynomial. Matrices A and D correspond to matrices $I - K$ for subgraphs of G, and by the inductive hypothesis A and D are $(O(d\sqrt{n}), \alpha)$-well-separable. Hence we can conclude that $I - K$ is $(O(d\sqrt{n}), \alpha)$-well-separable.

For the base case of the induction, observe that any small enough graph must be separable. For any graph with the number of edges less than a small constant c, we can safely assume its matrix $I - K$ is $(O(d\sqrt{n}), \alpha)$-well-separable because any way to partition the edge set into halves gives B and C rank less than c.

Now it remains to analyze the probability that all $I - K$, A and D remain invertible after substituting random values to $k_{i,j}$. Note that $\det(I - K)$ is a degree m polynomial not identically equal to zero, because $I - K$ has ones on its diagonal. Thus, by the Schwartz-Zippel Lemma, $I - K$ is invertible with probability at least $1 - m/|\mathbb{F}|$. Since A and D correspond to matrices $I - K$ for

subgraphs of G, we can apply the same argument to A and D recursively, and all the required matrices are invertible with probability at least $1 - O(m \log m)/|\mathbb{F}|$.

As a result $I - K$ is $(O(d\sqrt{n}), \alpha)$-well-separable. We can now apply Theorem 4.3 with $\gamma = 1.5$ [1]. So $(I - K)^{-1}$ can be computed in $O(n^{1.5} + (d\sqrt{n})^{\omega-2}n^2) = O(d^{\omega-2}n^{\omega/2+1})$ time.

Finally we analyze the required time to compute the rank of $(I - K)^{-1}{}_{\delta^{out}(s),\delta^{in}(t)}$ for all source s and sink t. We will show that this can be done in $O(d^{\omega-2}n^2)$ time.

Since the rank of an $a \times b$ matrix can be computed in $O(ab^{\omega-1})$ time by Theorem 2.12. The ranks of $(I - K)^{-1}{}_{\delta^{out}(s),\delta^{in}(t)}$ from one source s to all receivers t can be computed in

$$\sum_{t \in v} O(d^{in}(t) \cdot (d^{out}(s))^{\omega-1}) = O(m \cdot (d^{out}(s))^{\omega-1})$$

time. So the overall time to compute the ranks for all pairs of vertices is

$$\sum_{s \in v} O(n \cdot (d^{out}(s))^{\omega-1})$$

as $m = O(n)$. Now we show that this sum is at most $O(d^{\omega-2}n^2)$. Let $d_i = d^{out}(v_i)$. We partition the vertices into into groups V_j so that $\sum_{i \in V_j} d_i \leq d$, where d is the maximum degree of the graph. Note that there exists a partition so that all V_j (except at most one) has

$$d/2 \leq \sum_{i \in V_j} d_i \leq d,$$

otherwise we can merge two groups that has sum less then $d/2$. Thus there are at most $m/(d/2) + 1 = O(n/d)$ groups. Now we can bound the overall time to compute the ranks,

$$\begin{aligned}\sum_{s \in v} \sum_{t \in v} d^{in}(t) \cdot (d^{in}(s))^{\omega-1} &= n \cdot \sum_{s \in v} (d^{in}(s))^{\omega-1} \\ &= n \cdot \sum_{j=1}^{O(n/d)} \sum_{s \in V_j} (d^{in}(s))^{\omega-1}\end{aligned}$$

$$
\begin{aligned}
&\leq n \cdot \sum_{j=1}^{O(n/d)} \left(\sum_{s \in V_j} d^{in}(s) \right)^{\omega-1} \\
&\leq n \cdot \sum_{j=1}^{O(n/d)} d^{\omega-1} \\
&= n \cdot O(n/d) \cdot d^{\omega-1} \\
&= O(d^{\omega-2} n^2).
\end{aligned}
$$

The first inequality is true because $d^{in}(s)$ are all positive and the exponent $\omega - 1$ is not smaller than one. Hence the total time to compute the ranks is $O(d^{\omega-2}n^2)$, which is dominated by the time to compute $(I-K)^{-1}$. And we have the following conclusion.

Corollary 4.4. *All pairs edge connectivities can be computed in $O(d^{\omega-2}n^{\omega/2+1})$ time for any directed fixed minor free graph with maximum degree d.*

4.4 Open Problems

After the discussion on the algebraic approach on the graph connectivity problem in this chapter, there are some interesting problems remain. Is it possible to give an $O(n^\omega)$ time algorithm to compute the edge connectivity between one pair of vertices? If such algorithm exists then we can also compute all pairs edge connectivities in $O(n^{\omega+2})$ time, which is faster than the $O(m^\omega)$ time algorithm we discussed.

Can we drop the requirement that the maximum degree d of the given graph is small for our all pairs edge connectivities algorithm for planar graphs and its generalization? On other hand, can this algebraic approach be generalized to the capacitated case, where each edge has a capacity?

Finally, is it possible to design a combinatorial algorithm, that is faster than computing the edge connectivities between each pair of vertices separately in directed graphs?

Bibliography

[1] N. Alon, P. Seymour, and R. Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.

[2] M.A. Babenko. A fast algorithm for the path 2-packing problem. *Theory of Computing Systems*, 46(1):59–79, 2010.

[3] P. Berman, M. Fürer, and A. Zelikovsky. Applications of the linear matroid parity algorithm to approximating steiner trees. In *Proceedings of the 1st International Computer Science Symposium in Russia (CSR)*, pages 70–79, 2006.

[4] A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, pages 605–614, 2007.

[5] A. Bouchet and W.H. Cunningham. Delta-matroids, jump systems, and bisubmodular polyhedra. *SIAM Journal on Discrete Mathematics*, 8(1):17–32, 1995.

[6] U. Brandes and D. Wagner. A linear time algorithm for the arc disjoint Menger problem in planar directed graphs. *Algorithmica*, 28:16–36, 2000.

[7] J.R. Bunch and J.E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[8] G. Călinescu, C.G. Fernandes, U. Finkler, and H. Karloff. A better approximation algorithm for finding planar subgraphs. *Journal of Algorithms*, 27(2):269–302, 1998.

[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] P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28:2133–2149, 1999.

[11] H.Y. Cheung, L.C. Lau, and K.M. Leung. Algebraic algorithms for linear matroid parity problems. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 1366–1382, 2011.

[12] H.Y. Cheung, L.C. Lau, and K.M. Leung. Graph connectivities, network coding, and expander graphs. *Manuscript*, 2011.

[13] M. Chudnovsky, W.H. Cunningham, and J. Geelen. An algorithm for packing non-zero A-paths in group-labelled graphs. *Combinatorica*, 28(2):145–161, 2008.

[14] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2001.

[16] W.H. Cunningham and J.F. Geelen. The optimal path-matching problem. *Combinatorica*, 17(3):315–337, 1997.

[17] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Calgary International Conference on Combinatorial Structures and Their Applications*, pages 69–87, 1970.

[18] J. Edmonds. Matroid intersection. *Annals of Discrete Mathematics*, 4:39–49, 1979.

[19] S. Even and R.E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, 1975.

[20] M.L. Furst, J.L. Gross, and L.A. McGeoch. Finding a maximum-genus graph imbedding. *Journal of the ACM*, 35(3):523–534, 1988.

[21] H.N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

[22] H.N. Gabow and M. Stallmann. Efficient algorithms for graphic matroid intersection and parity. In *Proceedings of the 12th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 210–220, 1985.

[23] H.N. Gabow and M. Stallmann. An augmenting path algorithm for linear matroid parity. *Combinatorica*, 6(2):123–150, 1986.

[24] H.N. Gabow and Y. Xu. Efficient algorithms for independent assignment on graphic and linear matroids. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 106–111, 1989.

[25] J. Geelen and S. Iwata. Matroid matching via mixed skew-symmetric matrices. *Combinatorica*, 25(2):187–215, 2005.

[26] C.D. Godsil. *Algebraic combinatorics*. Chapman & Hall mathematics. Chapman & Hall, 1993.

[27] M.X. Goemans. Minimum bounded degree spanning trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2006.

[28] N.J.A. Harvey. An algebraic algorithm for weighted linear matroid intersection. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 444–453, 2007.

[29] N.J.A. Harvey. *Matchings, Matroids and Submodular Functions*. PhD thesis, Massachusetts Institute of Technology, 2008.

[30] N.J.A. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM Journal on Computing*, 39(2):679–702, 2009.

[31] N.J.A. Harvey, D.R. Karger, and K. Murota. Deterministic network coding by matrix completion. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 489–498, 2005.

[32] R. Hassin and A. Levin. An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal of Computing*, 33:261–268, 2004.

[33] R.A. Horn and C.R. Johnson. *Matrix analysis*. Cambridge University Press, 1990.

[34] T.A. Jenkyns. *Matchoids: a generalization of matchings and matroids*. PhD thesis, University of Waterloo, 1974.

[35] P.M. Jensen and B. Korte. Complexity of matroid property algorithms. *SIAM Journal on Computing*, 11:184–190, 1982.

[36] T. Jordán. Rigid and globally rigid graphs with pinned vertices. In *Fete of Combinatorics and Computer Science*, pages 151–172, 2010.

[37] E. Kaltofen and V. Pan. Processor efficient parallel solution of linear systems over an abstract field. In *Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures*, pages 180–191, 1991.

[38] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41:214–235, 1994.

[39] E.L. Lawler. *Combinatorial optimization: networks and matroids*. Oxford University Press, 1976.

[40] J. Lee, M. Sviridenko, and J. Vondrák. Matroid matching: the power of local search. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 369–378, 2010.

[41] L. Lovász. On determinants, matchings and random algorithms. *Fundamentals of Computation Theory*, 79:565–574, 1979.

[42] L. Lovász. Matroid matching and some applications. *Journal of Combinatorial Theory, Series B*, 28(2):208–236, 1980.

[43] L. Lovász. The membership problem in jump systems. *Journal of Combinatorial Theory, Series B*, 70(1):45–66, 1997.

[44] L. Lovász and M.D. Plummer. *Matching theory*. Elsevier, 1986.

[45] W. Mader. Homomorphiesatze fur graphen. *Mathematische Annalen*, 178:154–168, 1968.

[46] W. Mader. Über die maximalzahl kreuzungsfreier H-Wege. *Archiv der Mathematik*, 31(1):387–402, 1978.

[47] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[48] M. Mucha and P. Sankowski. Maximum Matchings via Gaussian Elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

[49] M. Mucha and P. Sankowski. Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica*, 45:3–20, 2006.

[50] K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1987.

[51] K. Murota. *Matrices and matroids for systems analysis*. Springer Verlag, 2009.

[52] H. Narayanan, H. Saran, and V.V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arborescences and edge-disjoint spanning trees. *SIAM Journal on Computing*, 23(2):387–397, 1994.

[53] C.St.J.A Nash-Williams. Connected detachments of graphs and generalized Euler trails. *Journal of the London Mathematical Society*, 2(1):17–29, 1985.

[54] J.B. Orlin. A fast, simpler algorithm for the matroid parity problem. In *Proceedings of the 13th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 240–258, 2008.

[55] J.B. Orlin and J.H. Vande Vate. Solving the linear matroid parity problem as a sequence of matroid intersection problems. *Mathematical Programming: Series A and B*, 47(1):81–106, 1990.

[56] G. Pap. Packing non-returning A-paths. *Combinatorica*, 27(2):247–251, 2007.

[57] G. Pap. Some new results on node-capacitated packing of A-paths. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, pages 599–604, 2007.

[58] G. Pap. Packing non-returning A-paths algorithmically. *Discrete Mathematics*, 308(8):1472–1488, 2008.

[59] H.J. Prömel and A. Steger. A new approximation algorithm for the Steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89 – 101, 2000.

[60] M.O. Rabin and V.V. Vazirani. Maximum matchings in general graphs through randomization. *Journal of Algorithms*, 10(4):557–567, 1989.

[61] P. Sankowski. Weighted bipartite matching in matrix multiplication time. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 274–285, 2006.

[62] P. Sankowski. Processor efficient parallel matching. *Theory of Computing Systems*, 42:73–90, 2008.

[63] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*. Springer Verlag, 2003.

[64] A. Sebő and L. Szegő. The path-packing structure of graphs. In *Proceedings of the 10th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 131–143, 2004.

[65] A. Storjohann. High-order lifting and integrality certification. *Journal of Symbolic Computation*, 36(3-4):613–648, 2003.

[66] Z. Szigeti. On a min-max theorem of cacti. In *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 84–95, 1998.

[67] Z. Szigeti. On the graphic matroid parity problem. *Journal of Combinatorial Theory, Series B*, 88(2):247–260, 2003.

[68] A. Thomason. An extremal function for contractions of graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 95:261–265, 1983.

[69] W.T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107, 1947.

[70] M.A. Woodbury. Inverting modified matrices. *Memorandum Report*, 42:106, 1950.

[71] F. Zhang. *The Schur complement and its applications*. Springer Verlag, 2005.