WARSAW UNIVERSITY

Faculty of Mathematics,
Informatics and Mechanics

FINDING MAXIMUM MATCHINGS VIA GAUSSIAN ELIMINATION

PhD Thesis

Marcin Mucha

Supervisor:
dr hab. Krzysztof Diks

Warsaw, February 2005

Author's declaration

Aware of legal responsibility I herby declare that I have written this thesis myself and all its contents have been obtained by legal means.

Date Author's signature

Supervisor's declaration

This thesis is ready to be reviewed.

Date Supervisor's signature

Abstract

In this work, we present algorithms for finding maximum matchings in general and planar graphs. The algorithm for the general case has $O(n^\omega)$ time complexity, and the algorithm for the planar case has $O(n^{\omega/2})$ time complexity, where ω is the exponent of the best known matrix multiplication algorithm. Since $\omega \leq 2.38$, for dense graphs the general matching algorithm is asymptotically faster than previously known algorithms. In particular, it breaks through the long-standing $O(n^{2.5})$ barrier for the maximum matching problem. The planar matching algorithm is also faster than previously known algorithms. Both algorithms improve on previous results even in the case of bipartite graphs and bipartite planar graphs, respectively. We also give a simpler $O(n^\omega)$ algorithm for bipartite graphs and an elementary $O(n^3)$ algorithm for general graphs.

Our algorithms are based on the algebraic technique introduced by Lovász, Rabin and Vazirani, and its Gaussian elimination based improvement introduced recently by the author and Piotr Sankowski.

All the presented algorithms are Las Vegas algorithms, except for the planar case algorithm, which is Monte Carlo.

This work is based on the previous work by the author and Piotr Sankowski.

Keywords: maximum matching, fast matrix multiplication.

ACM Classification: F.2.2, G.2.2.

Streszczenie

W niniejszej rozprawie przedstawiamy nowe algorytmy znajdujące najliczniejsze skojarzenia w dowolnych grafach w czasie $O(n^\omega)$ i w grafach planarnych w czasie $O(n^{\omega/2})$. $O(n^\omega)$ jest tu złożonością mnożenia macierzy $n \times n$. Wiadomo, że $\omega \leq 2.38$ (patrz [7]). Najlepsze znane algorytmy dla problemu najliczniejszego skojarzenia mają, w przypadku grafów gęstych, złożoność $O(n^{2.5})$, są więc asymptotycznie wolniejsze od algorytmu o złożoności $O(n^\omega)$. Algorytm dla grafów planarnych o złożoności $O(n^{\omega/2})$ jest również asymptotycznie szybszy od znanych algorytmów. Co więcej, nasze algorytmy są szybsze nawet w przypadku, odpowiednio, grafów dwudzielnych i planarnych dwudzielnych. Przedstawiamy także prostszą wersję algorytmu o złożoności $O(n^\omega)$ dla przypadku grafów dwudzielnych oraz elementarny algorytm o złożoności $O(n^3)$ dla dowolnych grafów.

W naszych algorytmach korzystamy z techniki algebraicznej opracowanej przez Lovásza, Rabina i Vaziraniego, ulepszonej niedawno przez autora i Piotra Sankowskiego przy użyciu eliminacji Gaussa.

Wszystkie algorytmy są typu Las Vegas, z wyjątkiem algorytmu dla grafów planarnych, który jest typu Monte Carlo.

Niniejsza praca powstała w oparciu o wcześniejsze prace autora i Piotra Sankowskiego.

Słowa kluczowe: najliczniejsze skojarzenia, szybkie mnożenie macierzy.

Klasyfikacja tematyczna ACM: F.2.2, G.2.2.

Contents

Chapter 1

Introduction

The maximum matching problem. A *matching* in a graph $G = (V, E)$ is a set $M \subseteq E$ of edges of G with no two edges in M having a common endpoint. A *maximum matching* is a matching of maximum cardinality, and a *perfect matching* is a matching of cardinality $|V|/2$.

In the maximum matching problem we look for a maximum matching in a given graph, and in the perfect matching problem we look for a perfect matching, if one exists. In this work we give several algorithms for the maximum/perfect matching problem.

Historical background. Solving the maximum matching problem in time polynomial in n remained an elusive goal for a long time until Edmonds [9] gave the first algorithm in 1965, using ideas of Berge [3]. His algorithm works in time $O(n^4)$, but in 1976 Gabow showed that a more careful implementation works in time $O(n^3)$. In 1975, Even and Kariv [11] improved the Edmonds' algorithm and achieved time complexity of $O(\min\{n^{2.5}, m\sqrt{n}\log n\})$, but their algorithm is very complicated. A bit simpler and faster algorithm was found by Micali and Vazirani [27] in 1980. Their algorithm works in time $O(m\sqrt{n})$. Other algorithms with the same time complexity, but using different methods, have been found since then (see Blum [4] and Gabow and Tarjan [14]).

The maximum matching problem is much easier in case of bipartite graphs. Already in 1931, König [22] and Egerváry [10] gave a characterization of maximum matchings in bipartite graphs together with a constructive proof, yielding a polynomial time algorithm — the so-called Hungarian method — for the maximum bipartite matching problem. In 1973, Hopcrof and Karp [17] showed an $O(m\sqrt{n})$ algorithm. This bound was later slightly improved by Alt, Blum, Mehlhorn and Paul [2] in 1991. Their algorithm works in time $O(n^{3/2}\sqrt{m/\log n})$. Finally, Feder and Motwani [12] constructed an $O(\sqrt{n}m\log(n^2/m)/\log n)$ algorithm using the technique of clique compression. Very recently Fremuth-Paeger and Jungnickel [13] showed that using ideas of Feder and Motwani, the same time complexity can be achieved for general graphs.

Faster algorithms exist for the maximum matching problem in case of planar graphs. For a start, such graphs have $O(n)$ edges, so we have $O(m\sqrt{n}) = O(n^{3/2})$. We can do even better for bipartite planar graphs. Using the duality-based reduction of

maximum flow with multiple sources and sinks to single source shortest paths problem (see Miller and Naor [28]), Klein, Rao, Rauch and Subramanian [21] were able to give a maximum matching algorithm for this case working in time $O(n^{\frac{4}{3}} \log n)$. However, the reduction they use does not carry over to the case of general planar graphs.

Algebraic algorithms. In 1979 Lovász [25] showed, using a completely new, algebraic approach, that it is possible to test whether a given graph has a perfect matching in randomized time $O(n^\omega)$. Here, $O(n^\omega)$ is the time required to multiply two $n \times n$ matrices. The algorithm of Coppersmith and Winograd [7] does that in time $O(n^{2.38})$, so for dense graphs Lovász's algorithm is faster then any of the algorithms actually finding a perfect matching.

A natural question is thus whether Lovász's approach can be used to construct a pefect matching in time $O(n^\omega)$. Rabin and Vazirani [33] showed an algorithm that works in time $O(n^{\omega+1})$. A lot of matching related information about a graph can be gained in time $O(n^\omega)$ (see Cheriyan [6]). For example, it is possible to find its Gallai-Edmonds decomposition, canonical partition, identify the allowed edges, etc. However, no $O(n^\omega)$ matching algorithm has been found so far.

The ideas of Lovász, Rabin and Vazirani also yield an RNC algorithm for the perfect matching problem (see Mulmuley, V. Vazirani and U. Vazirani [31], Galil and Pan [38], and Karp, Upfal and Wigderson [20]). These algorithms can be made Las Vegas (see Karloff [19] and Wein [36]).

Our results. We show an improvement of the technique of Lovász, Rabin and Vazirani, based on the Gaussian elimination algorithm. We then use the improved technique to give the following algorithms:

- a very simple $O(n^3)$ maximum matching algorithm,
- a simple $O(n^\omega)$ maximum matching algorithm for bipartite graphs,
- an $O(n^\omega)$ maximum matching algorithm,
- an $O(n^{\omega/2})$ maximum matching algorithm for planar graphs.

The first of these algorithms is slower than the best known combinatorial algorithms. However, its simplicity makes it ideal for practical purposes. For dense graphs, the $O(n^\omega)$ algorithms are asymptotically faster than the combinatorial algorithms. The $O(n^{\omega/2})$ matching algorithm is faster than the best known algorithms for planar graphs, even in the bipartite case.

This work is based on two papers, written by the author and Piotr Sankowski. The algorithms for general graphs were first given in [30], and the algorithms for planar graphs were given in [29]. However, several proofs have been simplified, and some additional considerations have been included (e.g. Chapter 5 on making the algorithm Las Vegas). Most importantly, the presentation in this work is much more detailed, since we are not limited in any way, as is the case with conference papers.

Organization of this work. We start by recalling some basic definitions and facts in Chapter 2. In Chapter 3 we discuss the algebraic technique for finding maximum matchings and our improvement of this technique, based on Gaussian elimination. We show that it gives a very simple $O(n^3)$ algorithm for the maximum matching problem, as well as $O(n^\omega)$ matching algorithms for bipartite and general graphs. In Chapter 4 we apply the ideas introduced in Chapter 3 to planar graphs, and give an $O(n^{\omega/2})$ maximum matching algorithm for this case. In Chapter 5 we discuss the problem of making our algorithms Las Vegas. Finally in Chapter 6 we give some open problems related to this work.

Acknowledgements. I would like to thank Piotr Sankowski, the co-author of the papers this work is based on. Working with Piotr has always been a lot of fun.

I would also like to thank my supervisor Krzysztof Diks for his invaluable support throughout my PhD studies, and in particular for his help in writing this PhD thesis.

Chapter 2

Definitions and Preliminaries

In this chapter we provide some basic definitions and facts from graph theory (Section 2.1), linear algebra (Section 2.2), matrix computations (Section 2.3) and randomized algorithms (Section 2.4).

It is mainly meant for reference, so we do not recommend reading it in its entirety, with a possible exception of Section 2.3 and Subsection 2.4.2. In Section 2.3 we give a somewhat lengthy exposition of the Gaussian elimination algorithm and its applications. Gaussian elimination is the single most important tool in the chapters to follow and getting acquainted with the results and ideas presented in Section 2.3 might prove useful. In Subsection 2.4.2 we describe a generic method of randomization based on the Zippel-Schwartz Lemma. Although this method is well-known and has widely been used before, our approach is much more formal than usual. Also, Subsection 2.4.2 introduces several new definitions.

2.1 Graphs

2.1.1 Basic Definitions

A *graph* is a pair $G = (V, E)$, where V is a finite set of *vertices*, and $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$ is a set of *edges*. The number of vertices is usually denoted by $n = |V|$, and the number of edges is denoted by $m = |E|$. Also, we will often assume that $V = \{v_1, \ldots, v_n\}$.

For any edge $e = \{u, v\}$, we say that u and v are the *endpoints* of e, and that e is *incident* to u and v. Two edges e_1, e_2 are *incident* if they have a common endpoint. If there is no danger of confusion, we denote $\{u, v\}$ by uv.

Two vertices joined by an edge are called *adjacent*. For any vertex v, $N(v)$ denotes the set of all vertices adjacent to v. Similarly, for any set $U \subseteq V$ of vertices, $N(U)$ denotes the set of all vertices adjacent to any of the vertices in U, i.e. $N(U) = \bigcup_{u \in U} N(u)$. For any vertex v, the number $|N(v)|$ is called the *degree* of v.

For any set $U \subseteq V$ of vertices, $G - U$ denotes the graph obtained by *removing* (or *deleting*) U from G, i.e. $G - U = (V \setminus U, E \cap \{\{u, v\} : u, v \in V \setminus U\})$. Similarly, for any set $F \subseteq E$ of edges, $G - F$ denotes the graph obtained by *removing* (or *deleting*) F from G, i.e. $G - F = (V, E \setminus F)$.

For any set $U \subseteq V$, by $G[U]$ we denote the subgraph of G *induced* by U, i.e. $G[U] = G-(V\setminus U)$. By G/U we denote the graph obtained by *contracting* U, i.e. G/U contains all vertices in $V \setminus U$ and a new node u, which replaces the set U. All edges of G not incident to a vertex in U are kept, all edges with both endpoints in U are removed, and for all edges with exactly one endpoint in U, this endpoint is replaced by u.

A sequence $p = v_0, v_1, \ldots, v_k$ of vertices of $G = (V, E)$ is called a *path* in G if all v_i are distinct and $v_i v_{i+1} \in E$ for $i = 0, 1, \ldots, k-1$. The number k is the *length* of p. The vertices v_0 and v_k are called the *endpoints* of p. A path with endpoints s, t is an s-t-path.

Similarly, a sequence $c = v_0, v_1, \ldots, v_k$ of vertices is called a *cycle* in G if $v_0 = v_k$, but other than that all v_i are distinct, and $v_i v_{i+1} \in E$ for $i = 0, 1, \ldots, k-1$. The number k is the *length* of c.

We will often view a path $p = v_0, v_1, \ldots, v_k$ in G as a subgraph $(\{v_0, v_1, \ldots, v_k\}, \{v_i v_{i+1} : i = 0, 1, \ldots, k-1\})$. Similarly, we will view a cycle $c = v_0, v_1, \ldots, v_k$ as a subgraph $(\{v_0, v_1, \ldots, v_k\}, \{v_i v_{i+1} : i = 0, 1, \ldots, k-1\})$.

A graph G is *connected* if every pair of its vertices is connected by a path. The *connected components* (or simply *components*) of a graph G are maximal connected subgraphs of G. The number of connected components of G is denoted $\mathrm{comp}(G)$.

A graph $G = (U \cup V, E)$, whose vertex set is a sum of two disjoint sets U, V such that each edge of G connects a vertex in U with a vertex in V, is called a *bipartite graph*. The sets U, V are called the *parts* of G. A bipartite graph is called *balanced* if its parts have equal size. For a balanced bipartite graph $G = (U \cup V, E)$, we often use the notation $n = |U| = |V|$. This is inconsistent with our previous notation, but it will always be clear from the context which definition we are referring to. Also, we will often assume that $U = \{u_1, u_2, \ldots, u_{|U|}\}$ and $V = \{v_1, v_2, \ldots, v_{|V|}\}$.

Graphs can be represented by matrices. For any graph $G = (V, E)$, the *adjacency matrix* $A(G)$ of G is an $n \times n$ matrix, defined as follows:

$$A(G)_{i,j} = \begin{cases} 1 & \text{if } v_i v_j \in E \\ 0 & \text{otherwise} \end{cases}$$

If G is bipartite, $G = (U \cup V, E)$, it can also be represented by its *bipartite adjacency matrix* $A(G)$. This is a $|U| \times |V|$ matrix, defined as follows:

$$A(G)_{i,j} = \begin{cases} 1 & \text{if } u_i v_j \in E \\ 0 & \text{otherwise} \end{cases}$$

Notice that for bipartite graphs, $A(G)$ denotes both the bipartite and the general adjacency matrix. It will always be clear from the context which of the two matrices is referred to.

2.1.2 Planar Graphs and Their Properties

Every graph can be drawn in the plane by representing the vertices with points and edges with line segments, where an edge $e = uv$ is represented with a line segment joining the points corresponding to u and v. A graph is called *planar* if it can be

drawn in the plane in such a way that the lines representing the edges only intersect at their endpoints.

We have the following:

Theorem 2.1. *Let $G = (V, E)$ be a planar graph, $n = |V|$, $m = |E|$. If $n \geq 3$, then*

$$m \leq 3n - 6.$$

A *separator* in a graph $G = (V, E)$ is a set $S \subseteq V$ such that connected components of $G - S$ have at most $\frac{2}{3}|V|$ vertices. An $s(n)$-*separator* is a separator of size $\leq s(n)$. Separators of size $O(\sqrt{n})$ are called *small* separators.

The following theorem of Lipton, Rose and Tarjan [23] gives one of the most important properties of planar graphs.

Theorem 2.2 (Separator Theorem). *Planar graphs have $O(\sqrt{n})$-separators. Moreover, such separators can be found in time $O(n)$.*

2.1.3 Matchings in Graphs

A *matching* M in a graph $G = (V, E)$ is a set of edges, such that no two edges of M have a common endpoint. A vertex v is *matched* by M if M contains an edge incident to v. A matching is *maximum* if it has the largest possible cardinality. The size of a maximum matching in G is denoted $\nu(G)$.

A matching is *perfect* if it matches all vertices, and *near perfect* if it matches all vertices but one. Obviously, only graphs with even number of vertices can have perfect matchings, and only graphs with odd number of vertices can have near perfect matchings.

Any subset of a matching is called a *submatching*. A matching is *allowed* if it is a submatching of some maximum matching. In particular, an edge is *allowed* if it is contained in some maximum matching.

2.2 Linear Algebra

2.2.1 Basic Definitions

By $\mathbb{R}$ ($\mathbb{Q}$, $\mathbb{C}$) we denote the field of real (rational, complex) numbers, by $\mathbb{F}_p$ (p prime) we denote the finite field of cardinality p, by $\mathbb{Z}$ we denote the ring of integers, and by $\mathbb{N}$ we denote the set of natural numbers (with 0). For any ring R and any set X, by $R[X]$ we denote the ring of polynomials with coefficients from R and variables from X. By $R(X)$ we denote the field of fractions of $R[X]$, i.e. the field of rational functions with coefficients from R and variables from X.

For any field F, we consider the set F^n of vectors with n components from F. Addition of vectors and multiplication of vectors by scalars from F are as usual. The set F^n with these operations is a vector space over F. A vector is always considered as a *column vector*, unless otherwise stated. The superscript "T" denotes *transposition*. So, for any $v \in F^n$, v^T is a *row vector*.

A vector $v \in F^n$ is called a *linear combination* of vectors $v_1, v_2, \ldots, v_k \in F_n$ if for some $\lambda_1, \lambda_2, \ldots, \lambda_k \in F$,

$$v = \sum_{i=1}^{k} \lambda_i v_i.$$

For any vectors $v_1, v_2, \ldots, v_k \in F_n$, the set of all linear combinations of $v_1, v_2, \ldots, v_k$ forms a *vector space spanned by* $v_1, v_2, \ldots, v_k$, denoted $\mathrm{span}\{v_1, \ldots, v_k\}$.

A set S of vectors is *linearly independent* if none of its members is a linear combination of the others. Otherwise S is *linearly dependent*. A *basis* of a vector space V is a maximal set of linearly independent vectors of V.

For $u, v \in F^n$, we define the *inner product* of u and v as $u^T v = \sum_{i=1}^{n} u_i v_i$. We say that u and v are *orthogonal*, denoted $u \perp v$, if $u^T v = 0$. For any subspace $V \subseteq F^n$, the set of vectors orthogonal to all vectors of V is a subspace of F^n, denoted $V^\perp$. We have the following:

Theorem 2.3. *Let $V \subseteq F^n$ be any subspace. Then $V = (V^\perp)^\perp$.*

The j-th unit vector, whose j-th component is 1 and all the other components are zero, is denoted e_j.

2.2.2 Matrix Algebra

For any ring R, $R^{m \times n}$ denotes the set of all $m \times n$ *matrices* with entries in R. For a matrix $A \in R^{m \times n}$, we assume that the row index set is $\{1, \ldots, m\}$ and the column index set is $\{1, \ldots, n\}$. The entry in the i-th row and j-th column of A is denoted $A_{i,j}$.

Let $A \in R^{m \times n}$. For any $I \subseteq \{1, \ldots, m\}$ and $J \subseteq \{1, \ldots, n\}$, by $A_{I,J}$ we denote the submatrix of A induced by rows and columns whose indices belong to I and J, respectively. A submatrix of the form $A_{I,I}$ is called a *principal submatrix* of A. We will sometimes write $A_{I,\cdot}$ for $A_{I,\{1,\ldots,n\}}$, and $A_{\cdot,J}$ for $A_{\{1,\ldots,m\},J}$.

Similarly, by $A^{I,J}$ we denote the submatrix of A induced by rows not in I and columns not in J.

The kernel of a matrix $A \in \mathbb{F}^{m \times n}$ is the space $\ker A \subseteq \mathbb{F}^n$ of all vectors v such that $Av = 0$.

The *determinant* of an $n \times n$ matrix A is denoted $\det A$. We often use the following formula.

$$\det A = \sum_{\pi \in \Sigma_n} (-1)^{\mathrm{sgn}\,\pi} \prod_{k=1}^{n} A_{k,\pi(k)} \tag{2.1}$$

Here, Σ_n is a set of all permutations of the set $\{1, \ldots, n\}$, and $\mathrm{sgn}\,\pi$ denotes the sign of π.

The inverse matrix of an $n \times n$ matrix A is denoted by A^{-1}. If a matrix has an inverse matrix then it is called *non-singular*, otherwise it is called *singular*. A matrix is non-singular iff its determinant is non-zero.

The *rank* of a matrix A, denoted $\operatorname{rank} A$, is the maximum number of rows (columns) in a non-singular square submatrix of A. Equivalently, the rank of A is the maximum number of linearly independent rows (columns) of A.

The *adjoint matrix* of a square matrix A, denoted $\operatorname{adj} A$, is an $n \times n$ matrix such that $(\operatorname{adj} A)_{i,j} = (-1)^{i+j} \det A^{j,i}$. Here, we use the short notation $A^{j,i}$ instead of $A^{\{j\},\{i\}}$. We use this kind of short notation in other cases as well, as long as there is no danger of confusion, e.g. $G - v$ instead of $G - \{v\}$.

The following theorem gives a very useful formula for the inverse matrix.

Theorem 2.4 (Adjoint Formula). *If A is non-singular, then*

$$A^{-1} = \operatorname{adj} A / \det A.$$

Theorem 2.5 (Laplace's Expansion Formula). *Let A be an $n \times n$ matrix. Then, for any row index $i \in \{1, \ldots, n\}$*

$$\det A = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} \det A^{i,j} = \sum_{j=1}^{n} A_{i,j} (\operatorname{adj} A)_{j,i}.$$

Similarly, for any column index $j \in \{1, \ldots, n\}$

$$\det A = \sum_{i=1}^{n} (-1)^{i+j} A_{i,j} \det A^{i,j} = \sum_{i=1}^{n} A_{i,j} (\operatorname{adj} A)_{j,i}.$$

The *identity matrix* is denoted by I or by I_n if we want to emphasize its dimension. The *zero matrix* is denoted by 0.

A matrix $A \in R^{n \times n}$ is *diagonal* if $A_{i,j} = 0$ for all $i \neq j$, $i, j = 1, 2, \ldots, n$. Let $\operatorname{diag}(a_1, a_2, \ldots, a_n)$ denote a diagonal matrix such that $\operatorname{diag}(a_1, a_2, \ldots, a_n)_{i,i} = a_i$.

A matrix is *lower-triangular* (*upper-triangular*) if $A_{i,j} = 0$ for $j > i$ ($j < i$), $i, j = 1, 2, \ldots, n$. A matrix is *unit lower-triangular* (*unit upper-triangular*) if it is lower-triangular (upper-triangular) and $A_{i,i} = 1$ for all $i \in 1, 2, \ldots, n$.

A *permutation matrix* corresponding to a permutation $\pi \in \Sigma_n$ is an $n \times n$ matrix P_π such that $P_{i,\pi(i)} = 1$, for all $i \in \{1, \ldots, n\}$, and all the other entries of P_π are equal to 0. We have $P_\pi^T = P_{\pi^{-1}}$. Moreover, permutation matrices are non-singular, and we have $P_\pi^{-1} = P_\pi^T = P_{\pi^{-1}}$. If A is an $n \times n$ matrix, and $\pi \in \Sigma_n$ is a permutation, then $P_\pi A$ is equal to A with its rows permuted using π, i.e. the i-th row of $P_\pi A$ is the $\pi(i)$-the row of A. Similarly AP_π is equal to A with its columns permuted using π^{-1}.

2.2.3 Properties of Skew-Symmetric Matrices

An $n \times n$ matrix A is called *skew-symmetric* if $A = -A^T$, i.e. $A_{i,j} = -A_{j,i}$ for all $i, j \in \{1, \ldots, n\}$. In this section we recall some of the most important properties of such matrices. Since they play a crucial role in our algorithms and are not very well known, we have decided to provide their proofs as well.

Theorem 2.6. *Let A be an $n \times n$ skew-symmetric matrix, and let $I = \{i_1, i_2, \ldots, i_k\}$ be such that $A_{i_1,\cdot}, A_{i_2,\cdot}, \ldots, A_{i_k,\cdot}$ is a maximal set of linearly independent rows of A. Then the principal submatrix $A_{I,I}$ is non-singular.*

Proof. Without loss of generality, let us assume that $I = \{1, 2, \ldots, k\}$. Since $A_{1,\cdot}$, $A_{2,\cdot}, \ldots, A_{k,\cdot}$ is a maximal set of linearly independent rows of A, we also have that $A_{\cdot,1}, A_{\cdot,2}, \ldots, A_{\cdot,k}$ is a maximal set of linearly independent columns of A, because A is skew-symmetric. Thus all of the columns $A_{\cdot,i}$, for $i = k+1, \ldots, n$, are linear combinations of $A_{\cdot,1}, A_{\cdot,2}, \ldots, A_{\cdot,k}$. It follows that $\operatorname{rank} A_{I,I} = \operatorname{rank} A_{I,\cdot} = k$, and so $A_{I,I}$ is non-singular. □

Corollary 2.7. *Any skew-symmetric matrix A has a maximum non-singular submatrix, which is principal.*

Lemma 2.8. *Let A be an $n \times n$ skew-symmetric matrix. If n is odd, then A is singular.*

Proof. From the definition of the determinant (2.1) it follows that $\det A = \det A^T$. Since $A^T = -A$, again using the definition (2.1), we have $\det A = (-1)^n \det A^T$. For odd n, we get $\det A^T = -\det A^T$, and so A is singular. □

Corollary 2.7 and Lemma 2.8 give the following.

Theorem 2.9. *The rank of a skew-symmetric matrix is even.*

2.2.4 Symmetric Positive Definite Matrices

An $n \times n$ matrix A is called *symmetric* if $A = A^T$, i.e. $A_{i,j} = A_{j,i}$ for all $i, j \in \{1, \ldots, n\}$.

For $F = \mathbb{R}$ or $\mathbb{C}$, a matrix $A \in F^{n \times n}$ is called *positive definite* if $v^T A v > 0$ for every non-zero vector $v \in F^n$.

Remark 2.10. *Note that this definition does not make sense for matrices over a fields of rational functions $\mathbb{Z}(X)$ or over a finite field $\mathbb{F}_p$.*

Theorem 2.11. *If A is positive definite, then all its diagonal entries are positive.*

Proof. $A_{i,i} = e_i^T A e_i$. □

Theorem 2.12. *A matrix A is positive definite iff $A = XDX^T$ for some non-singular matrix X and some diagonal matrix D with all diagonal entries positive.*

2.3 Matrix Computations

In this section we discuss effective algorithmic implementations of some of the operations introduced in the previous section: computing the determinant and the rank of a matrix, finding the inverse matrix, etc. But most importantly, we give a detailed description of the Gaussian elimination algorithm, the driving force behind our matching algorithms. Incidentally, all the other algorithms presented in this section rely on Gaussian elimination.

In Subsection 2.3.1 we discuss the Gaussian elimination algorithm and its applications. In Subsection 2.3.2 we show how the fast matrix multiplication algorithm of

Coppersmith and Winograd can be used to speed up the Gaussian elimination algorithm, and consequently all of its applications. Finally, in Subsection 2.3.3 we discuss the nested dissection algorithm — a fast implementation of Gaussian elimination in the case of so-called planar matrices.

Compared to previous sections, this one gives a much more thorough treatment of the subject. However, the presention is rather informal at times, as its aim is to acquaint the reader with the basic intuitions behind Gaussian elimination and the notation used, rather than to give rigorous derivations of all the algorithms. Further information concerning Gaussian elimination and matrix computations in general can be found in most textbooks on algorithms, for example Cormen, Leiserson and Rivest [8], or Aho, Hopcroft and Ullman [1]. The second of these is particularly useful, since it contains a complete analysis of the Hopcroft-Bunch implementation of Gaussian elimination.

2.3.1 Basic Gaussian Elimination

The purpose of Gaussian elimination is to represent a given matrix A as $A = PLDU$ (or $A = LDUP$), where L is unit lower-triangular, D is diagonal, U is unit upper-triangular, and P is a permutation matrix.

Let A have form:

$$A = A_1 = \begin{pmatrix} a_1 & v_1^T \\ u_1 & B_1 \end{pmatrix}$$

If $a_1 \neq 0$, we have the following identity.

(2.2)

$$\begin{pmatrix} a_1 & v_1^T \\ u_1 & B_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ u_1/a_1 & I_{n-1} \end{pmatrix} \begin{pmatrix} a_1 & 0 \\ 0 & B_1 - u_1 v_1^T/a_1 \end{pmatrix} \begin{pmatrix} 1 & v_1^T/a_1 \\ 0 & I_{n-1} \end{pmatrix} = L_1 A_2 U_1$$

This is the basic step of *Gaussian elimination*. Here, we are eliminating the first row (v_1^T) and the first column (u_1) of A_1. The element at the intersection of the eliminated row-column pair is called the *pivot*.

If we represent A_2 as

$$A_2 = \begin{pmatrix} a_1 & 0 & 0 \\ 0 & a_2 & v_2^T \\ 0 & u_2 & B_2 \end{pmatrix}$$

then, if only $a_2 \neq 0$, we can repeat the basic step and get

$$A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & u_2/a_2 & I_{n-1} \end{pmatrix} \begin{pmatrix} a_1 & 0 & 0 \\ 0 & a_2 & 0 \\ 0 & 0 & B_2 - u_2 v_2^T/a_2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & v_2^T/a_2 \\ 0 & 0 & I_{n-2} \end{pmatrix} = L_2 A_3 U_2$$

In the same manner we define matrices A_i, B_i, L_i, U_i for all $i \in \{1, \ldots, n-1\}$, provided that $a_i \neq 0$ for all $i \in \{1, \ldots, n-1\}$. We get the following factorization of A.

$$A = L_1 L_2 \ldots L_{n-1} D U_{n-1} \ldots U_2 U_1$$

where $D = \mathrm{diag}(a_1, a_2, \ldots, a_n)$.

It is easy to verify that $L = L_1 L_2 \ldots L_{n-1}$ is a unit lower-triangular matrix. Moreover $L = L_1 + L_2 + \ldots + L_{n-1} - (n-2)I_n$. Similarly, $U = U_{n-1} \ldots U_2 U_1$ is unit upper-triangular, and $U = U_{n-1} + \ldots + U_2 + U_1 - (n-2)I_n$.

We thus have the desired factorization.

$$A = LDU$$

The above factorization is even better than $A = PLDU$ we aimed at. This is because we made the assumption that $a_i \neq 0$ for all $i \in \{1, \ldots, n-1\}$. In general it may happen that $a_i = 0$. We then exchange the i-th row, containing $a_i = 0$, with some other row j, having a non-zero element in the i-th column. Such a row exists if A is non-singular. The process of exchanging rows in order get a non-zero pivot is called the *pivoting*. After the row exchange, we proceed with the elimination.

It might seem that pivoting makes our previous considerations useless, but it is not so. Imagine that we have done all the row exchanges before the elimination started. In that case we would need no pivoting during the elimination. Still the resulting L, D, U matrices would be the same. It follows, that Gaussian elimination with pivoting gives an LDU factorization of A with permuted rows, i.e. $P_\pi A$ for some permutation matrix P_π. We get the desired factorization $A = P_{\pi^{-1}} LDU$. We can also pivot columns instead of rows. This corresponds to factorizing AP_π for some permutation matrix P_π and we get a factorization $A = LDUP_{\pi^{-1}}$.

Theorem 2.13. *If A is a non-singular matrix, then the factorization $A = PLDU$ ($A = LDUP$) can be found in time $O(n^3)$ using Gaussian elimination.*

Corollary 2.14. *If A is a non-singular matrix, then* $\det A$ *and A^{-1} can be computed in time $O(n^3)$ using Gaussian elimination.*

Proof. If $A = PLDU$, then $\det A = \det D$. Also, $A^{-1} = U^{-1}D^{-1}L^{-1}P^{-1}$. The hard part here is to invert L and U, and this can be done in time $O(n^3)$ using the following recursive formulas.

$$L^{-1} = \begin{pmatrix} L_{S,S} & 0 \\ L_{T,S} & L_{T,T} \end{pmatrix}^{-1} = \begin{pmatrix} L_{S,S}^{-1} & 0 \\ -L_{T,T}^{-1} L_{T,S} L_{S,S}^{-1} & L_{T,T}^{-1} \end{pmatrix}$$

$$U^{-1} = \begin{pmatrix} U_{S,S} & U_{S,T} \\ 0 & U_{T,T} \end{pmatrix}^{-1} = \begin{pmatrix} U_{S,S}^{-1} & -U_{S,S}^{-1} U_{S,T} U_{T,T}^{-1} \\ 0 & U_{T,T}^{-1} \end{pmatrix},$$

where $S = \{1, \ldots, \lfloor n/2 \rfloor\}$ and $T = \{\lfloor n/2 \rfloor + 1, \ldots, n\}$. □

If A is singular, then during the elimination of A it may happen that the whole column containing a_i consists of zeros. We then find another column, not entirely zero, exchange it with the i-th column and proceed as usual. At some point, the lower right, uneliminated part of A consists of only zeros, and we finish the elimination. We get a factorization of the form $P_1 A P_2 = LDU$, where $D = \mathrm{diag}(a_1, \ldots, a_k, 0, 0, \ldots)$ and $a_i \neq 0$ for $i \leq k$. We need both P_1 and P_2, because we use both row and column exchanges.

Theorem 2.15. *For any matrix A, the factorization $A = P_1LDUP_2$, where $D = \operatorname{diag}(a_1, \ldots, a_k, 0, 0, \ldots)$ and $a_i \neq 0$ for $i \leq k$, can be found in time $O(n^3)$ using Gaussian elimination.*

Corollary 2.16. *For any matrix A, the rank of A and the maximum non-singular submatrix of A can be computed in time $O(n^3)$ using Gaussian elimination.*

Proof. Let $P_{\pi_1}AP_{\pi_2} = LDU$, where $D = \operatorname{diag}(a_1, \ldots, a_k, 0, 0, \ldots)$ and $a_i \neq 0$ for $i \leq k$. Then $\operatorname{rank} A = k$. Also $(P_{\pi_1}AP_{\pi_2})_{\{1,\ldots,k\},\{1,\ldots,k\}}$ is a maximum non-singular submatrix of $P_{\pi_1}AP_{\pi_2}$, so $A_{\{\pi_1(1),\ldots,\pi_1(k)\},\{\pi_2(1),\ldots,\pi_2(k)\}}$ is a maximum non-singular submatrix of A. □

Corollary 2.17. *For any $n \times n$ matrix A, the basis of $\ker A$ can be found in time $O(n^3)$ using Gaussian elimination.*

Proof. Consider the factorization $P_1AP_2 = LDU$, where $D = \operatorname{diag}(a_1, \ldots, a_k, 0, 0, \ldots)$ and $a_i \neq 0$ for $i \leq k$. We have

$$\ker A = \ker(LDUP_2^{-1}) = \ker(DUP_2^{-1}) = (UP_2^{-1})^{-1}(\ker D) = (P_2U^{-1})(\ker D).$$

Since vectors $e_{k+1}, e_{k+2}, \ldots, e_n$ form a basis of $\ker D$, the columns numbered $k+1, k+2, \ldots, n$ of P_2U^{-1} form a basis of $\ker A$. □

Notice that in order to compute $\det A$, $\operatorname{rank} A$ or find a maximum non-singular submatrix of A, we do not need the L and U matrices. We only need the permutation matrices (and the pivot values in case of $\det A$), i.e. the information on how the elimination proceeded.

This simplified Gaussian elimination, without constructing the LDU factorization, but only pairing the rows and columns, is all we need most of the time. For simplicity of presentation, this is the only form of Gaussian elimination we consider from now on.

We now present the simplified form of the basic Gaussian elimination algorithm (Algorithm 1).

Algorithm 1 Gaussian elimination

GAUSSIAN-ELIMINATION(A):
  for $i := 1$ **to** n **do**
    if there exists j such that $A_{i,j} \neq 0$ **then**
      choose any such j
      eliminate the i-th row and the j-th column

The next theorem describes the structure of a partially eliminated matrix.

Theorem 2.18. *Let A be an $n \times n$ matrix, and let*

$$A = \begin{pmatrix} A_{S,S} & A_{S,T} \\ A_{T,S} & A_{T,T} \end{pmatrix},$$

where $S = \{1, \ldots, k\}$. *If Gaussian elimination of the first* k *rows and columns of* A *does not require pivoting, then it gives the following factorization of* A

$$A = \begin{pmatrix} L & 0 \\ A_{T,S}U^{-1}D^{-1} & I_{n-k} \end{pmatrix} \begin{pmatrix} D & 0 \\ 0 & A_{T,T} - A_{T,S}A_{S,S}^{-1}A_{S,T} \end{pmatrix} \begin{pmatrix} U & D^{-1}L^{-1}A_{S,T} \\ 0 & I_{n-k} \end{pmatrix},$$

where LDU *is the factorization resulting from performing Gaussian elimination on* $A_{S,S}$.

Proof. After eliminating the first k rows and columns of A, we have

$$A = \begin{pmatrix} L & 0 \\ \hat{L} & I_{n-k} \end{pmatrix} \begin{pmatrix} D & 0 \\ 0 & \hat{A} \end{pmatrix} \begin{pmatrix} U & \hat{U} \\ 0 & I_{n-k} \end{pmatrix},$$

where $A_{S,S} = LDU$. By carrying out the matrix multiplications we get

$$A = \begin{pmatrix} LD & 0 \\ \hat{L}D & \hat{A} \end{pmatrix} \begin{pmatrix} U & \hat{U} \\ 0 & I_{n-k} \end{pmatrix} = \begin{pmatrix} LDU & LD\hat{U} \\ \hat{L}DU & \hat{L}D\hat{U} + \hat{A} \end{pmatrix}$$

Thus

$$\hat{L} = A_{T,S}U^{-1}D^{-1}, \quad \hat{U} = D^{-1}L^{-1}A_{S,T},$$

$$\hat{A} = A_{T,T} - \hat{L}D\hat{U} = A_{T,T} - \hat{L}DU(U^{-1}D^{-1}L^{-1})LD\hat{U} = A_{T,T} - A_{T,S}A_{S,S}^{-1}A_{S,T},$$

as claimed. □

Remark 2.19. *The formula given in Theorem 2.18 is very similar to formula* (2.2)*, only "the pivot" is now a submatrix and not a single element. Theorem 2.18 may be thought of as a variation of Gaussian elimination which allows for elimination of several rows and columns in one step. From our point of view, however, this theorem is only interesting because it gives a description of a partially eliminated matrix.*

2.3.2 Fast Matrix Multiplication

Recall the following theorem, due to Coppersmith and Winograd [7].

Theorem 2.20 (Coppersmith, Winograd). *Two* $n \times n$ *matrices over a ring can be multiplied in time* $O(n^\omega)$*, where* $2 \leq \omega < 2.376$.

We will refer to the algorithm given in Theorem 2.20 as the *fast matrix multiplication* algorithm. This algorithm can be used to perform Gaussian elimination in time $O(n^\omega)$. The first $O(n^\omega)$ implementation of Gaussian elimination was given by Hopcroft and Bunch [5] (a more readable description can be found in [1]). The algorithms given in this section are simplified versions of the Hopcroft-Bunch algorithm but the presentation is different.

Consider performing Gaussian elimination on a matrix M using Algorithm 1 and let $M_{A,B}$ consist of rows and columns of M that are not eliminated for several iterations. $M_{A,B}$ is updated in every iteration, but the results of these updates are not used until some row in A or some column in B is eliminated. Each update to $M_{A,B}$ amounts to substracting a matrix of the form cuv^T from $M_{A,B}$. Let $c_1u_1v_1^T, \ldots, c_ku_kv_k^T$ be a sequence of such updates. The accumulated update to $M_{A,B}$ is

$$c_1u_1v_1^T + \ldots + c_ku_kv_k^T = \begin{pmatrix} c_1u_1 & \cdots & c_ku_k \end{pmatrix} \begin{pmatrix} v_1^T \\ \vdots \\ v_k^T \end{pmatrix} \tag{2.3}$$

Computing the left side requires $k|A||B|$ operations, but the right side can be computed much faster using fast matrix multiplication.

The basic idea is to accumulate updates to the parts of M that are eliminated in later stages and only perform updates in large batches. Let us start with the simple case where no pivoting is required. Algorithm 2 shows how this can be done.

Algorithm 2 Fast Gaussian elimination with no pivoting

```
FAST-ELIMINATION-NO-PIVOTING(A):
    ELIMINATE-ROWS-AND-COLUMNS(A, 1, n)

ELIMINATE-ROWS-AND-COLUMNS(A, p, q):
    if p = q then
        lazily eliminate the p-th row and the p-th column
    else
        m := ⌊(p+q)/2⌋
        ELIMINATE-ROWS-AND-COLUMNS(A, p, m)
        update A_{m+1,...,q},{m+1,...,n}
        update A_{q+1,...,n},{m+1,...,q}
        ELIMINATE-ROWS-AND-COLUMNS(A, m + 1, q)
```

Here, "lazy elimination" means storing the expression of the form uv^T/c describing the changes to the rows and columns not yet eliminated. These changes are then performed in batches during the updates, using the identity (2.3).

Theorem 2.21. *Algorithm 2 performs Gaussian elimination with no pivoting in time $O(n^\omega)$.*

Proof. First of all, notice that Algorithm 2 indeed performs Gaussian elimination. This is because the eliminated rows and columns are always up-to-date.

Let us now analyze the time complexity of this algorithm. Consider a call to ELIMINATE-ROWS-AND-COLUMNS(X, p, q), and let $x = q - m$. The cost of the updates in this call is proportional to the cost of multiplying the $x \times x$ matrix by a

$x \times n$ matrix. This can be done in time $n/x x^\omega = n x^{\omega-1}$ by splitting the $x \times n$ matrix into n/x square $x \times x$ matrices. Let us assume, without loss of generality, that n is a power of 2. Then for every $j \in \{0, \ldots, \log n\}$, the value $x = 2^j$ appears $n/2^j$ times throughout the whole execution of the algorithm. We get the total time complexity of

$$\sum_{j=0}^{\log n} n/2^j n(2^j)^{\omega-1} = n^2 \sum_{j=0}^{\log n} (2^{\omega-2})^j \leq n^2 (2^{\omega-2})^{\log n+1} = O(n^\omega).$$

□

Remark 2.22. *Even if pivoting is necessary, we can still use Algorithm 2. If we get a zero pivot, we simply skip over the corresponding row-column pair without eliminating it. Thus we only eliminate a subset of rows and columns of A. This kind of elimination will be called* Gaussian elimination with skipping. *Algorithm 2 with skipping is used in Section 3.5 of Chapter 3.*

In the general case, when pivoting is neccessay, we get the following algorithm 3.

Algorithm 3 Fast Gaussian elimination (the Hopcroft-Bunch algorithm)

```
FAST-ELIMINATION(A):
    ELIMINATE-COLUMNS(A, 1, n)

ELIMINATE-COLUMNS(A, p, q):
    if p = q then
        find an uneliminated row r such that A_{r,p} ≠ 0
        lazily eliminate the r-th row and the p-th column
    else
        m := ⌊(p+q)/2⌋
        ELIMINATE-COLUMNS(A, p, m)
        update the uneliminated rows in columns m + 1, ..., q
        ELIMINATE-COLUMNS(A, m + 1, q)
```

Theorem 2.23. *Algorithm 3 performs Gaussian elimination in time $O(n^\omega)$.*

Proof. The proof of this theorem is essentially identical to the proof of Theorem 2.21. There is however one thing we have to take care of.

Consider a call ELIMINATE-COLUMNS(A,p,q). The recursive call ELIMINATE-COLUMNS(A,p,m) lazily eliminates columns $p, \ldots, m$. For $i = p, \ldots, m$, let r_i be the row eliminated with the i-th column. After eliminating columns $p, \ldots, m$ the submatrix $A_{\{r_p,\ldots,r_m\},\{m+1,\ldots,q\}}$ still has the values from before the elimination. But this submatrix has to be up-to-date before we start updating the uneliminated rows in columns $m+1, \ldots, q$. This can be done in $O((m-p)^\omega)$ time, again using the lazy updating. □

Remark 2.24. *The construction of the LDU factorization can be included in Algorithm 3 and we get an algorithm equivalent to the classical Hopcroft-Bunch algorithm. However, this improvement makes the algorithm, as well as its analysis, significantly more complex.*

By repeating the reasoning from Corollaries 2.14, 2.16 and 2.17 we get the following.

Corollary 2.25. *If A is an $n \times n$ matrix, then* $\det A$, $\operatorname{rank} A$, $\ker A$ *and a maximum non-singular submatrix of A can all be found in time $O(n^\omega)$ using the Hopcroft-Bunch algorithm.*

If A is non-singular, then A^{-1} can also be found in time $O(n^\omega)$.

Note that the factorization of A is only needed for the inverse and the kernel. The determinant, the rank and a maximum non-singular submatrix can all be found using Algorithm 3.

2.3.3 Nested Dissection

For any $n \times n$ matrix A, a *graph corresponding to* A is the graph $G(A) = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ and $v_i v_j \in E$ iff $i \neq j$ and $A_{i,j} \neq 0$ or $A_{j,i} \neq 0$. The graph $G(A)$ is thus a representation of the non-zero structure of A. In particular, we have $G(A(G)) = G$, where $A(G)$ is the adjacency matrix of G. Note that in the construction of $G(A)$ we ignore the diagonal entries of A.

We say that A is a *planar matrix* if $G(A)$ is a planar graph. Using the separator theorem (Theorem 2.2) for planar graphs, it is possible to perform Gaussian elimination on a planar matrix in time $O(n^{\omega/2})$. This implementation of Gaussian elimination is called the *nested dissection*. In this subsection we briefly recall how it works.

We say that $G = (V, E)$ has an *$s(n)$-separator family* (with respect to some constant n_0) if either $|V| \leq n_0$ or G has an $s(n)$-separator S such that the connected components of $G - S$ also have $s(n)$-separator families with respect to n_0. The partition resulting from recursive application of this definition is the *$s(n)$-separator tree*. The root of the $s(n)$-separator tree of G is an $s(n)$-separator S in G, its children are $s(n)$-separators in connected components of $G - S$ and so on.

The following is an easy corollary of Theorem 2.2.

Fact 2.26. *Planar graphs have $O(\sqrt{n})$-separator families. Moreover, an $O(\sqrt{n})$-separator tree for a planar graph can be found in time $O(n \log n)$.*

Lipton and Tarjan [24] showed the following:

Theorem 2.27 (Nested dissection). *Let X be a symmetric positive definite matrix and let $G(X)$ have an $O(\sqrt{n})$-separator family. Given an $O(\sqrt{n})$ separator tree for $G(X)$, Gaussian elimination on X can be performed in time $O(n^{\omega/2})$. The resulting factorization of X has the form $X = LDL^T$, where L is unit lower-triangular and has $O(n \log n)$ non-zero entries and D is diagonal.*

Remark 2.28. *The statement of Theorem 2.27 is quite different from that of Lipton and Tarjan. Also, they only give an $O(n^{3/2})$ complexity bound and a vague remark on improving it. The $O(n^{\omega/2})$ implementation was given by Pan and Reif [32]. However, they focus on the parallelization of nested dissection and seem to believe that sequential $O(n^{\omega/2})$ algorithm has already been given by Lipton and Tarjan.*

We are not going to present the details of this algorithm. The basic idea is to permute rows and columns of X using the $O(\sqrt{n})$-separator tree. Vertices of the top-level separator S correspond to the last $|S|$ rows and columns, etc.. Ordering rows and columns in this way guarantees that the matrix always has $O(n \log n)$ non-zero entries throughout the elimination.

Remark 2.29. *The assumption of X being symmetric positive definite is needed to assure that no pivoting is necessary, as it would spoil the separator based ordering of rows and columns. If we can guarantee this (i.e. no pivoting) in some other way, then the assumption can be omitted.*

Here is a very useful property of the factorization produced by the nested dissection:

Fact 2.30. *Let $X = LDL^T$ be the factorization produced by performing the nested dissection on X using an $O(\sqrt{n})$-separator tree T. Then both L and L^T have $O(\sqrt{n})$-separator families and T is an $O(\sqrt{n})$-separator tree for L and L^T.*

2.4 Randomization

2.4.1 Basic definitions

A *randomized algorithm* is an algorithm which is allowed to access a source of random bits. Consider a randomized algorithm and an input of size n for this algorithm. We say that an event occurs with *small probability* if its probability is $O(1/n)$. Similarly, we say that an event occurs with *high probability* if its probability is $1 - O(1/n)$, i.e. its complement occurs with small probability.

A randomized algorithm is a *Monte Carlo* algorithm if, for any fixed input, it may produce an incorrect output with small probability. A randomized algorithm is a *Las Vegas* algorithm if, for any fixed input, it either produces a correct output or reports failure, the latter with small probability. The difference between the two types of algorithms is that with a Las Vegas algorithm we know when it fails.

Obviously, we have the following:

Fact 2.31. *A Las Vegas algorithm $\mathcal{A}$ for a problem $\mathcal{P}$ can be converted into a Monte Carlo algorithm for $\mathcal{P}$ with the same time complexity.*

Proof. We execute $\mathcal{A}$, and if it reports failure, we return any (possibly incorrect) output. □

On the other hand:

Fact 2.32. *An $O(T(n))$ Monte Carlo algorithm $\mathcal{A}$ for a problem $\mathcal{P}$ can be converted into a Las Vegas algorithm for $\mathcal{P}$ with the same time complexity, provided that the correctness of the output of $\mathcal{A}$ can be verified in time $O(T(n))$.*

Proof. We execute $\mathcal{A}$, verify the correctness of the output, and then return it only if it is correct, otherwise we report failure. □

In the next subsection we introduce a generic method for designing randomized Monte Carlo algorithms. We use this method to produce several algorithms in Chapters 3 and 4. In Chapter 5 we discuss the problem of making these algorithms Las Vegas.

2.4.2 Randomization via Zippel-Schwartz Lemma

In this subsection we introduce the randomization method used throughout this work. This method is based on the celebrated Zippel-Schwartz Lemma.

Consider an algorithm $\mathcal{A}$ which uses $\mathbb{Z}(X)$ arithmetic for some set of variables X, i.e. it adds, multiplies, substracts and divides rational functions from $\mathbb{Z}(X)$. Each argument of a $\mathbb{Z}(X)$ operation has to be either a result of one of the previous operations or an element of X (viewed as a monomial). The only way to access the result of an operation is to test if it is identically zero. Let us call these tests *zero-tests*.

How efficiently can this kind of algorithm be implemented? If we settle for a randomized solution, we can do the following. We pick a prime p and substitute each variable in X with a random element from the finite field $\mathbb{F}_p$. Instead of $\mathbb{Z}(X)$ arithmetic we now perform $\mathbb{F}_p$ arithmetic, which can be done efficiently if $p = O(\mathrm{poly}(n))$. Let us call this substituted algorithm an $\mathbb{F}_p$ *implementation* of $\mathcal{A}$. The problem with $\mathbb{F}_p$ implementations is that some of the zero-tests might return true, even if the corresponding rational function is non-zero. We may even end up dividing by zero. Other than that, an $\mathbb{F}_p$ implementation is a perfectly good implementation of $\mathcal{A}$, since $\mathcal{A}$ only accesses the rational functions through zero-tests.

Let us assume that before performing a division, $\mathcal{A}$ always zero-tests the divisor. This way, to guarantee the correctness of any implementation of $\mathcal{A}$, we only need to guarantee that all the zero-tests give correct results.

For any input D, let $\mathrm{gen}_{\mathcal{A}}(D)$ be the set of the rational functions zero-tested by $\mathcal{A}$, when given D.

Let $\mathrm{gen}_{\mathcal{A}}(n) = \bigcup_{|D|=n} \mathrm{gen}_{\mathcal{A}}(D)$. The following theorem shows that under some additional assumptions, we can guarantee that all the zero-tests will give the correct answers with high probability.

Theorem 2.33. *Let $\mathcal{A}$ be an algorithm performing at most $T(n)$ zero-tests in $\mathbb{Z}(X)$ arithmetic for an input of size n. Suppose that every $f \in \mathrm{gen}_{\mathcal{A}}(n)$ has a representation $f = g/h$ such that $\deg g \leq D(n)$ and g has an $O(1)$ non-zero coefficient. Then, with high probability, the $\mathbb{F}_p$ implementation of $\mathcal{A}$ correctly answers all zero-tests, for any prime $p \geq nD(n)T(n)$.*

In order to prove the above theorem, we first recall a well known lemma, due to Zippel [39] and Schwartz [34].

Lemma 2.34 (Zippel,Schwartz). *If $p(x_1, \ldots, x_m)$ is a non-zero polynomial of degree d with coefficients in a field and S is a subset of the field, then the probability that p evaluates to 0 on a random element $(s_1, s_2, \ldots, s_m) \in S^m$ is at most $d/|S|$.*

Proof (of Theorem 2.33). Let p be a prime $\geq nD(n)T(n)$. Consider any input D of size n and any $f \in \mathrm{gen}_{\mathcal{A}}(D)$, $f \neq 0$. Since $f = g/h$, where g has an $O(1)$ non-zero coefficient, g is also a non-zero polynomial over $\mathbb{F}_p$. By Lemma 2.34, the probability that g (or, equivalently f) evaluates to 0 on a random element of $\mathbb{F}_p^{|X|}$ is $\leq \frac{D(n)}{nD(n)T(n)} = \frac{1}{nT(n)}$. Since $|\mathrm{gen}_{\mathcal{A}}(D)| \leq T(n)$, probability of any non-zero $f \in \mathrm{gen}_{\mathcal{A}}(D)$ evaluating to 0 on a random element of $\mathbb{F}_p^{|X|}$ is $\leq T(n)\frac{1}{nT(n)} = \frac{1}{n}$. □

Remark 2.35. *Theorem 2.33 is not the only approach to randomization we are going to use in this work. However, it gives a flavour of the Zippel-Schwartz Lemma based randomization.*

Also, notice that:

Fact 2.36. *For any $p = O(\mathrm{poly}(n))$, the $\mathbb{F}_p$ implementation of an algorithm performing $O(T(n))$ operations in $\mathbb{Z}(X)$ works in time $O(T(n)\log n)$. If the number of divisions performed by the algorithm is $o(T(n))$, then the $\mathbb{F}_p$ implementation works in time $O(T(n))$.*

Proof. If $p = O(\mathrm{poly}(n))$, all operations except for division work in time $O(1)$, and division works in time $O(\log n)$. □

Remark 2.37. *Since zero-testing a rational function is equivalent to zero-testing its numerator, one might wonder why not define $\mathrm{gen}_{\mathcal{A}}(D)$ and $\mathrm{gen}_{\mathcal{A}}(n)$ as sets of numerators of rational functions computed by $\mathcal{A}$. This would simplify the statement of Theorem 2.33. The problem with this approach is that every rational function has infinitely many representations as a quotient of two polynomials and it is not easy to single out the one that is most suitable.*

Remark 2.38. *Here, we are only concerned with efficient implementation of the $\mathbb{Z}(X)$ arithmetic. Algorithms discussed in this work perform some other operations as well, but the number of these operations is of the same order as the number of $\mathbb{Z}(X)$ operations, so we only consider the latter.*

In the remainder of this work, we give several algorithms working over $\mathbb{Z}(X)$ for some set of variables X, and use Theorem 2.33 (or some similar method) and Fact 2.36 to get efficient randomized implementations of these algorithms.

The randomization technique described above has been widely used before (e.g. [6, 33]). However, the usual approach is to describe and analyse the algorithm over $\mathbb{F}_p$. The issues of randomization and correctness of the algorithm are then intermixed, since all the crucial properties of the algorithm only hold with large probability. To avoid this kind of confusion, we introduced an intermediate stage — an algorithm working over $\mathbb{Z}(X)$. We believe that this approach, by separating the randomization considerations from the proof of correctness and the complexity analysis, facilitates understanding of the algorithms.

Chapter 3

Maximum Matchings via Gaussian Elimination

In this chapter we show how Gaussian elimination can be used to find maximum matchings in graphs. We start with presenting the algorithms of Lovász in Section 3.1 and Rabin and Vazirani in Section 3.2. In Section 3.3 we introduce our improvement of the Rabin-Vazirani algorithm, based on Gaussian elimination. It immediately gives an elementary $O(n^3)$ matching algorithm. In Section 3.4 we show that using the Hopcroft-Bunch implementation of Gaussian elimination we can achieve $O(n^\omega)$ complexity for bipartite graphs. The general case is much harder. Before giving an $O(n^\omega)$ solution in Section 3.6, we first show in Section 3.5 that a maximal allowed submatching of any matching can be found in $O(n^\omega)$. This is a key ingredient of the general matching algorithm.

In all these considerations we concentrate on the perfect matching problem. In Section 3.7 we show that the maximum matching problem reduces in time $O(n^\omega)$ to the perfect matching problem.

The results presented in this chapter are based on the papers of Lovász [25] (Section 3.1), Rabin and Vazirani [33] (Section 3.2), Mucha and Sankowski [30] (Sections 3.3, 3.4, 3.5 and 3.6), Ibarra and Moran [18], Rabin and Vazirani [33] and Cheriyan [6] (Section 3.7).

3.1 Algorithm of Lovász

Let $G = (V, E)$ be a graph and let $V = \{v_1, \ldots, v_n\}$. A *symbolic adjacency matrix* of G is an $n \times n$ matrix $\tilde{A}(G)$ such that

$$\tilde{A}(G)_{i,j} = \begin{cases} x_{i,j} & \text{if } v_iv_j \in E \text{ and } i < j \\ -x_{j,i} & \text{if } v_iv_j \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases},$$

where $x_{i,j}$ is a unique variable corresponding to edge v_iv_j. Let $\tilde{E} = \{x_{i,j} : v_iv_j \in E \text{ and } i < j\}$ be the set of all these variables. Then $\tilde{A}(G)$ is a matrix over $\mathbb{Z}(\tilde{E})$.

If $G = (U \cup V, E)$ is a bipartite graph, where $U = \{u_1, \ldots, u_{|U|}\}$, $V = \{v_1, \ldots, v_{|V|}\}$, then the *bipartite symbolic adjacency matrix* $\tilde{A}(G)$ of G is a $|U| \times |V|$ matrix such that

$$\tilde{A}(G)_{i,j} = \begin{cases} x_{i,j} & \text{if } u_i v_j \in E \\ 0 & \text{otherwise} \end{cases},$$

Remark 3.1. *In the papers of Lovász [25] and Rabin and Vazirani [33] only the general case is considered. This is to be expected as there is no point in proving theorems for bipartite graphs if they hold in the general case. However, bipartite graphs and bipartite symbolic adjacency matrices are crucial for our considerations (see Section 3.4 and 3.6), so we need to prove the bipartite versions of all theorems.*

For most of this chapter, we only use balanced bipartite graphs. The bipartite symbolic adjacency matrix is then a square matrix, which is required in our considerations. The general case is considered in Section 3.7.

Remark 3.2. *Note, that we use $\tilde{A}(G)$ to denote both bipartite and general versions of the symbolic adjacency matrix. It will always be clear from the context which one we refer to.*

Tutte [35] observed the following:

Theorem 3.3. *Let $\tilde{A}(G)$ be the (bipartite) symbolic adjacency matrix of a (balanced bipartite) graph G. Then $\det \tilde{A}(G) \neq 0$ iff G has a perfect matching.*

In order to prove the non-bipartite case of Theorem 3.3 we first need to characterize graphs having perfect matchings in terms of cycle covers. A *cycle cover* of G is a set of vertex disjoint cycles covering all vertices of G. A cycle cover is an *even cycle cover* if all its cycles have even length.

Lemma 3.4. *A graph G has a perfect matching if and only if it has an even cycle cover.*

Proof. If G has a perfect matching M, then the set of two-vertex cycles corresponding to the edges of M is an even cycle cover of G.

Conversely, if G has an even cycle cover C, then taking every second edge from each cycle gives a perfect matching in G. □

We are now ready to prove Theorem 3.3

Proof of Theorem 3.3. We have:

$$\det \tilde{A}(G) = \sum_{\pi \in \Sigma_n} (-1)^{\mathrm{sgn}(\pi)} \prod_{k=1}^{n} \tilde{A}(G)_{k,\pi(k)} \tag{3.1}$$

If G is a balanced bipartite graph and $\tilde{A}(G)$ is a bipartite symbolic adjacency matrix, then the non-zero terms of the right side of (3.1) correspond to the perfect matchings of G, which ends the proof of this case.

In the general case, the non-zero terms of $\det \tilde{A}(G)$ do not directly correspond to perfect matchings in G, but we will show that they do correspond to even cycle covers of G and the claim will follow from Lemma 3.4.

Non-zero elements of the right side of (3.1) now have form $(-1)^{\mathrm{sgn}(\pi)} \prod_{k=1}^{n} \pm x_{k,\pi(k)}$, where $v_1 v_{\pi(1)}, \ldots, v_n v_{\pi(n)}$ are edges of G. These edges form a cycle cover of G, corresponding to the cycle decomposition of π. Reversing the direction of any cycle of length > 2 in π leads to the same cycle cover of G and the question is: what happens to the sign of the corresponding non-zero term. Consider a permutation π' which is equal to π with a single cycle reversed. Reversing a cycle does not change the sign of the permutation, but changes the signs of all the variables $x_{i,\pi(i)}$ corresponding to the edges of this cycle. Thus the terms corresponding to π and π' have the same sign if the length of the reversed cycle is even, and opposite signs if the length is odd. It follows that the terms corresponding to even cycle covers add up and give non-zero monomials of $\det \tilde{A}(G)$, while all the other non-zero terms cancel out. □

The proof of Tutte's Theorem also suggests the following:

Theorem 3.5. *Let $\tilde{A}(G)$ be the (bipartite) symbolic adjacency matrix of a (balanced bipartite) graph G. If the determinant of $\tilde{A}(G)$ is non-zero, then it has an $O(1)$ non-zero coefficient.*

Proof. Let us first consider the bipartite case. Let $G = (U \cup V, E)$ and let us assume, without loss of generality, that $M = \{u_1 v_1, u_2 v_2, \ldots, u_n v_n\}$ is a perfect matching in G. Then $x_{1,1} x_{2,2} \ldots x_{n,n}$ has coefficient ± 1 in $\det \tilde{A}(G)$.

In the general case, let $M = \{v_1 v_2, v_3 v_4, \ldots, v_{n-1} v_n\}$ be a perfect matching in G. Then $x_{1,2}^2 x_{2,3}^2 \cdot \ldots \cdot x_{n-1,n}^2$ has coefficient ± 1 in $\det \tilde{A}(G)$. □

Theorem 3.3 suggests Algorithm 4 for deciding whether a given graph G has a perfect matching.

Algorithm 4 Lovász's testing algorithm

TEST-PERFECT-MATCHING(G):
  compute $\det \tilde{A}(G)$ using Gaussian elimination
  if $\det \tilde{A}(G) \neq 0$ **then**
    return "YES"
  else
    return "NO"

Theorem 3.6. *For any prime $p = \Theta(n^2)$, the $\mathbb{F}_p$ implementation of Algorithm 4 is an $O(n^\omega)$ Monte Carlo algorithm deciding whether a given graph G has a perfect matching.*

Proof. The determinant $\det \tilde{A}(G)$ is a polynomial of degree n. If it is non-zero, then it is also non-zero over $\mathbb{F}_p$, by Theorem 3.5. By Zippel-Schwartz Lemma, with probability at least $1 - n/p = 1 - O(1/n)$, it evaluates to a non-zero value for a random substitution of $\tilde{E}$ with elements of $\mathbb{F}_p$ and Gaussian elimination gives a non-zero determinant.

The complexity bound follows from Fact 2.36, since Gaussian elimination only requires n divisions. □

Remark 3.7. *We could, of course, try to use Theorem 2.33 to prove Theorem 3.6, but this would give an inferior bound on p. The key idea here is that we do not actually care if all the zero-tests performed by the $\mathbb{F}_p$ implementation are correct. We only need the Gaussian elimination algorithm to succeed, and for this a non-zero determinant is sufficient. Still, Theorem 2.33 will prove useful in case of more complex algorithms in Sections 3.5 and 3.6 of this chapter.*

Remark 3.8. *We require p to be $\Theta(n^2)$, and not only $\Omega(n^2)$, to guarantee that $p \in \text{poly}(n)$. We could of course make this assumption directly, but assuring $p = \Theta(n^2)$ is not a problem and the statement of the theorem is shorter. There is no point in taking p larger than necessary anyway. The same remark applies to most of the randomization considerations in the remainder of this work.*

3.2 Algorithm of Rabin and Vazirani

In the previous section we have proved that $\det \tilde{A}(G) \neq 0$ iff G has a perfect matching. In this case the matrix $\tilde{A}(G)$ has an inverse which, as Rabin and Vazirani noticed in [33], encodes very useful information about G.

Let us first consider the bipartite case. As a consequence of Theorem 3.3 we get the following.

Theorem 3.9. *Let $G = (U \cup V, E)$ be a bipartite graph having a perfect matching and let $\tilde{A} = \tilde{A}(G)$ be its bipartite symbolic adjacency matrix. Then $(\tilde{A}^{-1})_{i,j} \neq 0$ iff $G - \{u_j, v_i\}$ has a perfect matching.*

Proof. Since $\tilde{A}^{j,i} = \tilde{A}(G - \{u_j, v_i\})$, the claim follows from the adjoint formula for the inverse matrix (Theorem 2.4) and Tutte's Theorem (Theorem 3.3). □

As a special case, if $u_j v_i$ is an edge of G, Theorem 3.9 says that $(\tilde{A}^{-1})_{i,j} \neq 0$ iff $u_j v_i$ is allowed.

A similar theorem, due tu Rabin and Vazirani [33], holds in the general case.

Theorem 3.10 (Rabin,Vazirani). *Let $G = (V, E)$ be a graph having a perfect matching, and let $\tilde{A} = \tilde{A}(G)$ be its symbolic adjacency matrix. Then $(\tilde{A}^{-1})_{i,j} \neq 0$ iff the graph $G - \{v_i, v_j\}$ has a perfect matching.*

Remark 3.11. *To be precise, Rabin and Vazirani only show in [33] that if $(\tilde{A}^{-1})_{i,j} \neq 0$, then $G - \{v_i, v_j\}$ has a perfect matching, as this is all they need in their algorithm. However, Cheriyan [6] seems to believe that they actually prove Theorem 3.10.*

In the proof of the above theorem we will again use the adjoint formula for the inverse matrix. Notice, however, that $\tilde{A}^{j,i}$ only misses the j-th row and the i-th column, while $\tilde{A}(G - \{v_i, v_j\}) = \tilde{A}^{\{i,j\},\{i,j\}}$ misses two rows and two columns, so we first need to prove the following.

Lemma 3.12. *Let A be an $n \times n$ non-singular skew-symmetric matrix. Then $A^{i,j}$ is non-singular iff $A^{\{i,j\},\{i,j\}}$ is non-singular, for any $i, j \in \{1, \ldots, n\}$, $i \neq j$.*

Proof. If $A^{i,j}$ is non-singular, then it has rank $n-1$. Since $A^{\{i,j\},\{i,j\}}$ misses one of $A^{i,j}$'s rows and one of its columns, it has rank $\geq n-3$. But $A^{\{i,j\},\{i,j\}}$ is skew-symmetric, so by Corollary 2.9 its rank is even, and it has to be $n-2$. Thus $A^{\{i,j\},\{i,j\}}$ is non-singular.

Conversely, let $A^{\{i,j\},\{i,j\}}$ be non-singular. We then have $\operatorname{rank} A^{\{i,j\},\{i,j\}} = n-2$, so $\operatorname{rank} A^{i,\{i,j\}} \geq n-2$. By Corollary 2.9, we also have $\operatorname{rank} A^{i,i} = n-2$, so $\operatorname{rank} A^{i,\{i,j\}} = \operatorname{rank} A^{i,i} = n-2$. But this means that the j-th column of $A^{i,\emptyset}$ is a linear combination of its other columns, so $\operatorname{rank} A^{i,j} = \operatorname{rank} A^{i,\emptyset} = n-1$. □

Proof (of Theorem 3.10). The claim follows from the adjoint formula for the inverse matrix (Theorem 2.4), Lemma 3.12 and Tutte's Theorem (Theorem 3.3). □

Again, if $v_i v_j$ is an edge in G, then $A^{-1}(G)_{j,i} \neq 0$ iff $v_i v_j$ is allowed.
Theorem 3.10 can be used to find perfect matchings (Algorithm 5).

Algorithm 5 Matching algorithm of Rabin and Vazirani

```
RABIN-VAZIRANI(G):
    M := ∅
    for i := 1 to n do
        if v_i is not yet matched then
            compute Ã(G)^{-1}
            find j, such that v_i v_j ∈ E(G) and (Ã(G)^{-1})_{j,i} ≠ 0
            M := M ∪ {v_i v_j}
            G := G − {v_i, v_j}
    return M
```

In the case of balanced bipartite graphs, we can also use the bipartite adjacency matrix (Algorithm 6).

Theorem 3.13. *For any prime $p = \Theta(n^2)$, the $\mathbb{F}_p$ implementation of Algorithm 5 (Algorithm 6) is an $O(n^{\omega+1})$ Monte Carlo algorithm finding a perfect matching in a (bipartite) graph G having one.*

To prove the Theorem 3.13 we need the following lemma.

Lemma 3.14. *Let M be an $n \times n$ matrix. If M is non-singular, then for every $i \in \{1, \ldots, n\}$, there exists $j \in \{1, \ldots, n\}$ such that $M_{i,j} \neq 0$ and $(M^{-1})_{j,i} \neq 0$.*

Algorithm 6 Matching algorithm of Rabin and Vazirani (bipartite version)

RABIN-VAZIRANI-BIPARTITE(G):
  $M := \emptyset$
  for $i := 1$ **to** n **do**
    compute $\tilde{A}(G)^{-1}$ { the bipartite symbolic adjacency matrix }
    find j, such that $u_i v_j \in E(G)$ and $(\tilde{A}(G)^{-1})_{j,i} \neq 0$
    $M := M \cup \{u_i v_j\}$
    $G := G - \{u_i, v_j\}$
  return M

Proof. This is an immediate consequence of the Laplace's Expansion (Theorem 2.5). □

As a corollary we get the following.

Theorem 3.15. *Let $G = (U \cup V, E)$ be a bipartite graph having a perfect matching, let $\tilde{A} = \tilde{A}(G)$ be its bipartite symbolic adjacency matrix, and let $s : \tilde{E} \to \mathbb{F}_p$ be a substitution, such that $\det \tilde{A}(s) \neq 0$. Then for every $u_i \in U$ there exists $v_j \in V$ such that $u_i v_j \in E$ and $(\tilde{A}(s)^{-1})_{j,i} \neq 0$, i.e. $\tilde{A}(s)^{-1}$ encodes the edge $u_i v_j$ as being allowed. Moreover, $\det(\tilde{A}(G - \{u_i, v_j\})(s)) \neq 0$.*

Proof. The theorem follows from Lemma 3.14 if we take $M = \tilde{A}(s)$. The second part of the theorem follows from the adjoint formula for the inverse matrix (Theorem 2.4). □

Similar theorem, due to Rabin and Vazirani [33], is also true in the general case.

Theorem 3.16. *Let $G = (V, E)$ be a graph having a perfect matching, let $\tilde{A} = \tilde{A}(G)$ be its symbolic adjacency matrix, and let $s : \tilde{E} \to \mathbb{F}_p$ be a substitution, such that $\det \tilde{A}(s) \neq 0$. Then for every $v_i \in V$ there exists $v_j \in V$ such that $v_i v_j \in E$ and $(\tilde{A}(s)^{-1})_{j,i} \neq 0$, i.e. $\tilde{A}(s)^{-1}$ encodes the edge $v_i v_j$ as being allowed. Moreover, $\det(\tilde{A}(G - \{v_i, v_j\})(s)) \neq 0$.*

Proof. The first part of the theorem again follows from Lemma 3.14, and the second part follows from Lemma 3.12. □

We are now ready to prove Theorem 3.13.

Proof (of Theorem 3.13). If the substitution $s : \tilde{E} \to \mathbb{F}_p$ chosen by the $\mathbb{F}_p$ implementation of Algorithm 6 satisfies $\det \tilde{A}(s) \neq 0$, then by iterated application of Theorem 3.15, the algorithm finds a perfect matching. The same is true for Algorithm 5, by Theorem 3.16.

Thus, we only need to guarantee that with high probability $\det \tilde{A}(s) \neq 0$, and for $p = \Theta(n^2)$ this follows from the Zippel-Schwartz Lemma. □

3.3 Gaussian Elimination

The bottleneck of the algorithm of Rabin and Vazirani (Algorithm 5) and its bipartite version (Algorithm 6) is the computation of the inverse of $\tilde{A}(G)$. In every iteration we remove only two rows and two columns from $\tilde{A}(G)$ (a single row and a single column in case of Algorithm 6), but the inverse is recomputed from scratch. Rabin and Vazirani suggested in [33] that it should be possible to find a method of updating the inverse matrix, faster than recomputing it from scratch. Recently we have suggested such a method, based on the Gaussian elimination algorithm (see [30]). We present it in this section.

Our method is based on the following well known fact.

Theorem 3.17 (Elimination Theorem). *Let*

$$A = \begin{pmatrix} a_{1,1} & v^T \\ u & B \end{pmatrix} \quad A^{-1} = \begin{pmatrix} \hat{a}_{1,1} & \hat{v}^T \\ \hat{u} & \hat{B} \end{pmatrix},$$

where $\hat{a}_{1,1} \neq 0$. *Then* $B^{-1} = \hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}$.

Proof. Since $AA^{-1} = I$, we have

$$\begin{pmatrix} a_{1,1}\hat{a}_{1,1} + v^T\hat{u} & a_{1,1}\hat{v}^T + v^T\hat{B} \\ u\hat{a}_{1,1} + B\hat{u} & u\hat{v}^T + B\hat{B} \end{pmatrix} = \begin{pmatrix} I_1 & 0 \\ 0 & I_{n-1} \end{pmatrix}.$$

Using these equalities we get

$$B(\hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}) = I_{n-1} - u\hat{v}^T - B\hat{u}\hat{v}^T/\hat{a}_{1,1} =$$
$$I_{n-1} - u\hat{v}^T + u\hat{a}_{1,1}\hat{v}^T/\hat{a}_{1,1} = I_{n-1} - u\hat{v}^T + u\hat{v}^T = I_{n-1}.$$

and so $B^{-1} = \hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}$ as claimed. □

Notice that the modification of $\hat{B}$ described in the above theorem is a single step of Gaussian elimination. Here, we are eliminating the first column and the first row of A^{-1}.

As an immediate consequence of Theorem 3.17 we get the following algorithms finding perfect matchings in general (Algorithm 7) and balanced bipartite (Algorithm 8) graphs. These algorithm are simply modifications of Algorithm 5 and Algorithm 6, based on the Elimination Theorem.

Remark 3.18. *Note that the two-step elimination in Algorithm 7 always succeeds. This is because* $(\tilde{A}(G)^{-1})_{i,i} = (\tilde{A}(G)^{-1})_{j,j} = 0$, *so the the value of* $(\tilde{A}(G)^{-1})_{j,i}$ *does not change during the elimination of the* i*-th row and the* j*-th column.*

Theorem 3.19. *For any prime* $p = \Theta(n^2)$, *the* $\mathbb{F}_p$ *implementation of Algorithm 7 (Algorithm 8) is an* $O(n^3)$ *Monte Carlo algorithm finding a perfect matching in a (balanced bipartite) graph* G *having one.*

Algorithm 7 Simple matching algorithm

SIMPLE-MATCHING(G):
  $M := \emptyset$
  compute $\tilde{A}(G)^{-1}$
  for $i := 1$ **to** n **do**
    if the i-th row is not yet eliminated **then**
      find j such that $v_i v_j \in E(G)$ and $(\tilde{A}(G)^{-1})_{j,i} \neq 0$
      $M := M \cup \{v_i v_j\}$
      $G := G - \{v_i, v_j\}$
      update $\tilde{A}(G)^{-1}$ by eliminating the i-th row and the j-th column
        and then the j-th row and the i-th column
  return M

Algorithm 8 Simple matching algorithm (bipartite version)

SIMPLE-BIPARTITE-MATCHING(G):
  $M := \emptyset$
  compute $\tilde{A}(G)^{-1}$ { the bipartite symbolic adjacency matrix }
  for $i := 1$ **to** n **do**
    find j such that $u_i v_j \in E(G)$ and $(\tilde{A}(G)^{-1})_{j,i} \neq 0$
    $M := M \cup \{u_i v_j\}$
    $G := G - \{u_i, v_j\}$
    update $\tilde{A}(G)^{-1}$ by eliminating the i-th row and the j-th column
  return M

Proof. The first part of the theorem follows from the fact that both algorithms are equivalent to the algorithm of Rabin and Vazirani. The time complexity is $O(n^3)$ by Fact 2.36, because both algorithms perform $O(n^3)$ operations in $\mathbb{Z}(\tilde{E})$ and only n of them are divisions. □

Even though Algorithm 7 and Algorithm 8 are slower than the Micali-Vazirani algorithm or other fast implementations of the Edmonds algorithm, they are extremely simple and might be a serious practical alternative. In the remainder of this chapter we show that by using fast implementations of Gaussian elimination, we can beat the combinatorial algorithms in terms of asymptotic complexity as well.

3.4 Bipartite Matching Algorithm

Throughout the last three sections we accompanied every theorem and every algorithm with its bipartite version. This might have seemed redundant, since one can

always use the general algorithm.

However, in this section we show that Algorithm 8, the bipartite version of Algorithm 7, can easily be modified to work in time $O(n^\omega)$. Similar improvement in case of Algorithm 7 is not likely to succeed. Although we give an $O(n^\omega)$ algorithm for the general case in the next section, it is not only much more complicated, but also uses the bipartite algorithm of this section as a subroutine.

When we compare Algorithm 8 with the Gaussian elimination algorithm (Algorithm 1), we can see that they are essentially the same algorithm. The only difference is that in Algorithm 8 we choose a pivot which not only is non-zero, but also corresponds to a graph edge. This additional requirement can easily be included in Algorithm 1, as well as in the Hopcroft-Bunch algorithm (Algorithm 3). We get Algorithm 9.

Algorithm 9 $O(n^\omega)$ bipartite matching algorithm

```
BIPARTITE-MATCHING(G):
    A := Ã(G)^{-1}
    MATCH-U-VERTICES(1, n)

MATCH-U-VERTICES(p, q):
    if p = q then
        find an unmatched vertex v_r, such that u_p v_r ∈ E and A_{r,p} ≠ 0
        lazily eliminate the r-th row and the p-th column of A
    else
        m := ⌊(p+q)/2⌋
        MATCH-U-VERTICES(p, m)
        update the uneliminated rows in columns m + 1, ..., q of A
        MATCH-U-VERTICESS(m + 1, q)
```

Theorem 3.20. *For any prime $p = \Theta(n^2)$, the $\mathbb{F}_p$ implementation of Algorithm 9 is an $O(n^\omega)$ Monte Carlo algorithm finding a perfect matching in a bipartite graph G having one.*

Proof. Algorithm 9 is almost identical to the Hopcroft-Bunch algorithm, only some matrix related notions have been exchanged with graph related ones.

Randomization analysis is the same as for Algorithm 8, and the time complexity is equal to the complexity of the Hopcroft-Bunch algorithm. □

In case of general graphs, deleting v_i and v_j from G corresponds to eliminating the i-th row and the j-th column and then the j-th row and the i-th column. This is not how the usual Gaussian elimination works, so we cannot use the Hopcroft-Bunch algorithm directly. We believe that there is no way to perform this kind of Gaussian elimination in time $O(n^\omega)$ using the lazy computation techniques only. In order to

apply these we need some kind of information about the elimination order, so that we can delay updating the parts that are eliminated later. In Algorithm 7 we have no such information.

3.5 Matching Verification

Before we start with the general case, we first introduce a very interesting application of the Gaussian elimination technique. Incidentally, it is also the key ingredient of our matching algorithm for the general case.

Theorem 3.21. *Let G be a graph having a perfect matching. For any matching M of G, an inclusion maximal allowed submatching M' of M can be found using $O(n^\omega)$ operations in $\mathbb{Z}(\tilde{E})$.*

Proof. Consider the following algorithm.

Let $M = \{v_1v_2, v_3v_4, \ldots, v_{k-1}v_k\}$ and let $v_{k+1}, v_{k+2}, \ldots, v_n$ be the unmatched vertices. Compute the inverse $A(G)^{-1}$ and permute its rows and columns so that the row order is v_1,v_2,v_3,v_4, $\ldots$, v_n and the column order is v_2,v_1,v_4,v_3, $\ldots$, v_n,v_{n-1}. Now, perform Gaussian elimination of the first k rows and k columns using Algorithm 2 with skipping (see Remark 2.22). The eliminated row-column pairs correspond to a maximal submatching M' of M. □

Randomization of the verification algorithm given above cannot be carried out using the simple technique used so far. Instead, we use Theorem 2.33, but first we need to characterize the rational functions zero-tested by the verification algorithm.

Lemma 3.22. *Let $G = (V, E)$ be a (bipartite) graph having a perfect matching, and let $\tilde{A} = \tilde{A}(G)$ be its (bipartite) symbolic adjacency matrix. Then, all the non-zero entries of $\tilde{A}^{-1}$ have form g/h, where g is a polynomial of degree $O(n)$ and has a non-zero $O(1)$ coefficient.*

Proof. Let $(\tilde{A}^{-1})_{i,j} = \det \tilde{A}^{j,i} / \det \tilde{A}$ be non-zero. Since $\det \tilde{A}^{j,i}$ is a polynomial of degree $n-1$, we only need to show that it has a non-zero $O(1)$ coefficient.

In the bipartite case $\tilde{A}^{j,i} = \tilde{A}(G - \{u_j, v_i\})$, so the claim follows from Theorem 3.5.

Let us consider the general case. Working with the determinant of $\tilde{A}^{j,i}$ is a bit awkward, instead we consider the following matrix:

$$(B^{j,i})_{x,y} = \begin{cases} \tilde{A}_{x,y} & \text{if } (x \neq j) \text{ and } (y \neq i) \\ 1 & \text{if } (x = j) \text{ and } (y = i) \\ 0 & \text{otherwise} \end{cases},$$

From Laplace's expansion formula (Theorem 2.5) it follows that

$$\det \tilde{A}^{j,i} = (-1)^{i+j} \det B^{j,i}.$$

We have

$$\begin{aligned}\det B^{j,i} &= \sum_{\pi\in\Sigma_n,\pi(j)=i} (-1)^{\mathrm{sgn}(\pi)} \prod_{k=1}^{n} B^{j,i}_{k,\pi(k)} = \\ &= \sum_{\pi\in\Sigma_n,\pi(j)=i} (-1)^{\mathrm{sgn}(\pi)} \prod_{k\in\{1,\ldots,j-1,j+1,\ldots,n\}} \tilde{A}_{k,\pi(k)}\end{aligned}$$

The non-zero terms of this sum have the form $(-1)^{\mathrm{sgn}(\pi)} \prod_{k\in\{1,\ldots,j-1,j+1,\ldots,n\}} x_{k,\pi(k)}$, where $\pi(j) = i$. For any such term, the cycle decomposition of π gives a covering of G consisting of a v_i-v_j-path (but not necessarily a cycle, since we do not know if v_i and v_j are neighbours) and a set of cycles. If any of these cycles has odd length, then the term corresponding to π cancels out with some other term, just like in the proof of Tutte's Theorem. If, on the other hand, all the cycles have even length, then the the term corresponding to π gives a non-zero contribution to $\det B^{j,i}$.

Since $\det \tilde{A}^{j,i} \neq 0$, $G - \{v_i, v_j\}$ has a perfect matching M. Let M_G be a perfect matching of G. Consider the symmetric difference $M \oplus M_G$, i.e. the set of edges contained in exactly one of these matchings. Let $H = (V, M \oplus M_G)$. The vertices v_i and v_j have degree 1 in H and all the other vertices of G have degree 0 or 2 in H. The graph H is thus a sum of disjoint cycles and v_i-v_j-path p. Moreover, M_G matches the vertices not on p with other vertices not on p. Two-edge cycles corresponding to the edges of M_G not contained in p, together with p, form a covering of G with even length cycles and a v_i-v_j-path. The monomial in $\det B^{j,i}$ corresponding to this covering has coefficient ± 1, as required. □

Remark 3.23. *The ideas used in the above proof lead to an alternative, combinatorial proof of Lemma 3.12.*

Theorem 3.24. *For any prime $p = \Theta(n^3)$, the $\mathbb{F}_p$ implementation of the algorithm described in the proof of Theorem 3.21 is an $O(n^\omega)$ Monte Carlo algorithm finding an inclusion maximal allowed submatching of a given matching.*

Proof. It follows from Remark 3.18, that after eliminating the first row-column pair corresponding to vertices v_{2i-1}, v_{2i}, the second one can be eliminated without zero-testing the pivot. The only elements zero-tested by the algorithm are thus (the numerators of) the pivots used in odd-numbered iterations. There are at most $n/2$ such pivots and each of them is an element of $\tilde{A}^{-1}(H)$ for some subgraph $H \subseteq G$, so by Lemma 3.22 it has the form g/h, where g is a polynomial of degree $\leq n$ having an $O(1)$ non-zero coefficient. The assertion of the theorem now follows from Theorem 2.33.

As usual, the time complexity of the $\mathbb{F}_p$ implementation is $O(n^\omega)$, because Gaussian elimination uses only n divisions. □

3.6 General Matching Algorithm

Our discussion at the end of Section 3.4 suggests that Gaussian elimination by itself is not sufficient to solve the general case of the perfect matching problem. In this section we combine it with structural properties of graphs having a perfect matching, and achieve an $O(n^\omega)$ algorithm for the general case.

3.6.1 Basic Idea

Algorithm 10 $O(n^\omega)$ matching algorithm

GENERAL-MATCHING(G):
  find a matching M_G of size $\geq n/4$ in G using the greedy algorithm
  find a maximal allowed submatching M of M_G using the verification algorithm
  if $|M| \geq n/8$ **then**
    $M := M \cup$ GENERAL-MATCHING($G - V(M)$)
  else
    $M := M \cup$ DECOMPOSE($G - V(M)$)
  return M

The idea of Algorithm 10 is that if $|M| \geq n/8$, then we have to find a perfect matching in a graph smaller by a constant factor, and if $|M| \leq n/8$ then it is possible to use some structural properties to decompose G into smaller pieces and find a perfect matching in each of them separately. The decomposition uses the fact that G has a large matching consisting of unallowed edges only (i.e. $M_G - M$).

3.6.2 Canonical Partition

We now describe the details of the DECOMPOSE procedure, used in Algorithm 10.

Let us start with a few definitions. A graph $G = (V, E)$ is called *factor-critical* if for any $v \in V$, $G - v$ has a perfect matching. A graph G is called *elementary* if G has a perfect matching and allowed edges of G form a connected subgraph of G.

For any graph G, let us define a relation $\sim_G$ on the set V of vertices of G as follows: $u \sim_G v$ iff either $u = v$ or $G - \{u, v\}$ does not have a perfect matching.

The following theorem is due to Lovász (for a proof, see [26])

Theorem 3.25. *If G is elementary, then $\sim_G$ is an equivalence relation.*

Let $P(G) = V(G)/\sim_G$ be the set of equivalence classes of $\sim_G$, the so-called *canonical partition* of G. Recall that $G - \{u, v\}$ has a perfect matching iff $\tilde{A}(G)^{-1} \neq 0$, so $\tilde{A}(G)^{-1}$ encodes the canonical partition.

The partition $P(G)$ has very nice structural properties as the following theorem shows (for a proof, see [26])

Theorem 3.26. *Let G be elementary, let $S \in P(G)$ with $|S| \geq 2$ and let C be any component of $G - S$. Then:*

1. *the bipartite graph G'_S obtained from G by contracting each component of $G - S$ to a single vertex and deleting edges in S is elementary,*
2. *the graph C is factor-critical,*
3. *the graph C' obtained from $G[V(C) \cup S]$ by contracting the set S to a single vertex u_C is elementary,*

4. $P(C') = \{\{u_C\}\} \cup \{T \cap V(C) | T \in P(G)\}$.

A set $S \in P(G)$ with $|S| \geq 2$ will be called a *non-trivial class* of the canonical partition of G. If $P(G)$ has a non-trivial class S, then it follows from Theorem 3.26 that the number of connected components in $G - S$ is equal to $|S|$ and that every perfect matching of G matches vertices of S with vertices in different components of $G - S$. Moreover, any such matching of vertices of S can be extended to a perfect matching of G.

The decomposition algorithm (Algorithm 11) breaks G down into bipartite and non-bipartite pieces and reduces the problem of finding a perfect matching in G to the problem of finding perfect matchings in all the pieces using the $O(n^\omega)$ bipartite matching algorithm (Algorithm 9) and Algorithm 10.

Algorithm 11 The decomposition procedure

DECOMPOSE(G):
1. **if** G is not elementary **then**
2. find elementary components of G
3. call DECOMPOSE for each of them
4. **return** the union of the matchings found
5. **else if** there is no non-trivial class S in $P(G)$ **then**
6. **return** GENERAL-MATCHING(G)
7. **else**
8. let $S \in P(G)$ be a non-trivial class in $P(G)$
9. let $C_1, \ldots, C_k$ be connected components of $G - S$ with C_1 being the largest
10. let C_1' be $G[S \cup V(C_1)]$ with S contracted to a single vertex s
11. $M :=$ DECOMPOSE(C_1')
12. let c_1 be the vertex matched with s in M
13. let $v_1 \in S$ be any neighbour of c_1
14. $M := M \setminus \{c_1 s\} \cup \{c_1 v_1\}$
15. extend $\{c_1 v_1\}$ to a matching M_B which matches all vertices of S
16. with vertices in different C_i using BIPARTITE-MATCHING algorithm
17. $M := M \cup M_B$
18. **for** $i := 2$ **to** k **do**
19. let c_i be the vertex of C_i matched by M
20. $C_i := C_i - c_i$
21. $M :=$ $M \cup$ GENERAL-MATCHING(C_i)
22. **return** M

Theorem 3.27. *Algorithm 11 finds a perfect matching in a given graph G having one.*

Proof. It follows from Theorem 3.26 that any perfect matching in G is a sum of $k+1$ perfect matchings. One of them is a perfect bipartite matching between S and a set C containing a single vertex c_i from each of the C_i and the other k matchings are perfect matchings in $C_i - c_i$.

Algorithm 11 first constructs the matching in $C_1 - c_1$ and matches c_1 with some v_1 in S. This is done by calling DECOMPOSE(C_1') and then substituting edge $c_1 s$, matching an artificial vertex s, with an edge $c_1 v_1$.

Next the remaining vertices of S are matched with vertices in different C_i. This is done by finding a perfect matching in a bipartite graph G'_S defined in Theorem 3.26. All edges of this graph are allowed by the definition of $\sim_G$ and $P(G)$, so it is possible to find a perfect matching in G'_S consistent with edge $c_1 v_1$.

Finally perfect matchings in graphs $C_i - c_i$ are found using GENERAL-MATCHING. These graphs all have perfect matchings, because C_i are factor-critical, again by Theorem 3.26. □

Algorithm 11 reduces the problem of finding a perfect matching in G to the problem of finding perfect matchings in smaller bipartite and non-bipartite graphs. We now show that the total size of all these graphs is n, and that the non-bipartite graphs are all smaller than G by a constant factor. It follows that the complexity of both Algorithm 10 and Algorithm 11 is $O(n^\omega)$, provided that we can implement the decomposition algorithm itself with $O(n^\omega)$ operations. This last problem is quite technical and is considered in Subsection 3.6.3.

Lemma 3.28. *The total number of vertices in graphs for which Algorithm 11 calls BIPARTITE-MATCHING or GENERAL-MATCHING is equal to n.*

Proof. Every edge of the matching M returned by Algorithm 11 is either an element of one of the matchings found by calls to BIPARTITE-MATCHING or GENERAL-MATCHING or is a substitute of such edge ($c_1 v_1$ is a substitute for $c_1 s$), and hence the claim. □

Lemma 3.29. *If G contains a matching $\hat{M}$ of size $\geq n/6$, consisting of only unallowed edges, then DECOMPOSITION(G) calls GENERAL-MATCHING for graphs with $\leq 7/9n$ vertices.*

Proof. Since C_1 is the largest component of $G - S$, all the other C_i have less than $n/2$ vertices, so the calls to GENERAL-MATCHING(C_i) in line 21 satisfy the claim of the theorem.

The only other call to GENERAL-MATCHING is made in line 6 when canonical partition $P(G)$ has no non-trivial class. This can happen after any number of recursive calls in line 11. We will show that when GENERAL-MATCHING is finally called in line 6, G has at most $7/9n$ vertices.

Initially, G has a matching $\hat{M}$ of size $\geq n/6$, consisting of only unallowed edges. Thus, $\hat{M}$ matches at least $n/3$ vertices. Let $\hat{V}$ be the set of these vertices. Notice that if, at any level of recursion, G contains a vertex $v \in \hat{V}$, then it also contains the vertex u matched with v in $\hat{M}$. This is because edges of $\hat{M}$ are unallowed, so they

are always contained either in S or in some C_i. Also, if G contains any edge $uv \in \hat{M}$, then $P(G)$ contains a non-trivial class — the one containing u and v.

When DECOMPOSE is called recursively in line 11, it is called for a graph that is missing at least 3 of G's vertices (at least two vertices in S and at least one vertex in C_2) and gains exactly one artificial vertex s. When GENERAL-MATCHING(G) is called in line 6, G does not have any vertex from $\hat{V}$, so it is missing at least $n/3$ vertices. Thus, G then has at most $n - 2/3|\hat{V}| \leq n - 2/9n = 7/9n$ vertices. □

In Subsection 3.6.3 we will show the following:

Lemma 3.30. *The DECOMPOSE procedure together with all its recursive calls, but excluding the calls to GENERAL-MATCHING and BIPARTITE-MATCHING, can be implemented using $O(n^2 \log^k n)$ operations, for some k.*

We are now ready to prove the main theorem of this section.

Theorem 3.31. *The GENERAL-MATCHING algorithm (Algorithm 10) finds a perfect matching in a given graph G having one, using $O(n^\omega)$ operations.*

Proof. The GENERAL-MATCHING algorithm uses $O(n^\omega)$ operations for BIPARTITE-MATCHING calls and for book-keeping.

It also, directly or indirectly (through the DECOMPOSE procedure), calls itself recursively. In both cases the recursive calls of GENERAL-MATCHING are made for graphs smaller than G by a constant factor. For direct calls, this is obvious. For indirect calls, notice that when DECOMPOSE($G - V(M)$) is called by GENERAL-MATCHING(G), we have $|M| \geq n/8$, so $G - V(M)$ has at most $n - n/4 = 3n/4$ vertices and we can use Lemma 3.29.

Also, the total number of vertices in the graphs for which GENERAL-MATCHING is called recursively is at most n, by Lemma 3.28. By easy induction (or by the so-called Master's Theorem), GENERAL-MATCHING uses $O(n^\omega)$ operations. □

3.6.3 Implementation Details

So far we have ignored the problem of performing the decomposition and focused on the complexity of finding perfect matchings in its parts. The ideas that lead to an $\tilde{O}(n^2)$ implementation of the decomposition are implicit in the work of Cheriyan [6], and we now give a detailed description by proving Lemma 3.30. Here, $\tilde{O}$ is the so-called "soft O" notation. We write $f(n) = \tilde{O}(g(n))$ iff $f(n) = O(g(n) \log^k n)$ for some constant k.

We need the dynamic connectivity algorithm of Holm, de Lichtenberg and Thorup [16]. It supports the following operations on a dynamic graph:

- INSERT(e) — inserts an edge e into G;
- DELETE(e) — deletes an edge e from G;
- SIZE(v) — returns the size the connected component containing v;

- CONNECTED(u, v) — tests if u and v are connected by a path in G;

All these operations require $\tilde{O}(1)$ time.

Proof (of Lemma 3.30). First of all, notice that when DECOMPOSE(G) is called by GENERAL-MATCHING, the matrix $\tilde{A}(G)^{-1}$ is already computed, and it gives the canonical partition of elementary components of G. Also, when DECOMPOSE(G) makes a recursive call to DECOMPOSE(C_1'), we do not need to compute $\tilde{A}(C_1')$, because by Theorem 3.26 the canonical partition of C_1' is induced by the canonical partition of G. Thus in every call to DECOMPOSE(G), not necessarily the top-level one, we know the canonical partition of G.

Algorithm 12 gives another view on Algorithm 11, emphasizing the implementations details. Only the decomposition itself is included in this description, construction of a perfect matching does not cause any problems.

Algorithm 12 Implementation details of the DECOMPOSE procedure

DECOMPOSE(G): { implementation details }
{ cases where no decomposition is performed }
1. choose a non-trivial class S in $P(G)$
2. call DELETE(e) for every edge e between $G[S]$ and $G - S$
   let T be the set of all endpoints in $G - S$ of these edges
3. call SIZE(v) for every $v \in T$
   let u be a vertex for which the value returned was the largest
4. $u \in C_1$, find $C_2 \cup \ldots \cup C_k$ by calling CONNECTED(u, v) for all $v \in T$
5. identify $C_2, \ldots, C_k$ by starting a DFS from every vertex in $C_2 \cup \ldots \cup C_k$
   $C_1 := G - S - C_1 - \ldots - C_k$
6. add a new vertex s to C_1
   call INSERT(s, v) for all $v \in T \setminus (C_2 \cup \ldots \cup C_k)$
7. call DECOMPOSE(C_1)
   { matching construction }

Steps 3., 4. and 6. require $\tilde{O}(n)$ operations. Since the whole decomposition process makes $O(n)$ recursive calls to DECOMPOSE, the total number of operations required to perform these steps is $\tilde{O}(n^2)$.

The complexity of step 1. is $O(n^2)$ operations for the whole partition. To see this, notice that for each v we only need to test once if v is contained in a non-trivial class of $P(G)$. If v is an element of such S, then we use S as a basis for decomposition, and then C_1 does not contain v. If, on the other hand, v is not an element of a non-trivial class of $P(G)$, then it will never be, because of part 4. of Theorem 3.26.

The number of operations required to perform steps 2. and 5. is proportional to the number of edges between $G[S]$ and $G - S$ and edges inside smaller components. These sets of edges are disjoint for different recursion levels, so the total complexity of steps 2. and 5. is $\tilde{O}(m) = \tilde{O}(n^2)$ operations.

The decomposition algorithm thus requires $\tilde{O}(n^2)$ operations, as claimed. □

3.6.4 Randomization

Theorem 3.32. *For any prime $p = \Theta(n^{\omega+2})$, the $\mathbb{F}_p$ implementation of Algorithm 10 is an $O(n^\omega)$ Monte Carlo algorithm finding a maximum matching in a given graph G having one.*

Proof. Algorithm 10 performs zero-tests:

- during the matching verification, and
- to retrieve information about the canonical partition.

All the rational functions tested in these two situations are elements of $\tilde{A}^{-1}(H)$ for some subgraph $H \subseteq G$ and thus have form g/h, where g has degree $O(n)$ and an $O(1)$ non-zero coefficient (recall the proof of Theorem 3.24).

Algorithm 10 also makes calls to the bipartite matching algorithm. In order to make these calls find perfect matchings, it is enough to guarantee that the determinants of the corresponding submatrices of $\tilde{A}(G)$ are non-zero (recall the proof of Theorem 3.13). Again, these determinants have degree $O(n)$ and an $O(1)$ non-zero coefficient.

The assertion of the theorem now follows from Theorem 2.33. □

Remark 3.33. *More careful analysis shows that $p = \Theta(n^4)$ is enough.*

3.7 Maximum Matchings

So far we have only considered perfect matchings in this chapter. We now show that the problem of finding a maximum matching can be reduced to the problem of finding a perfect matching. There are several methods of doing this, the one we have chosen is likely to be among the simplest ones, it is also in the same spirit as the other results of this chapter, and as we shall see in the next chapter, it can be easily modified to work for planar graphs.

The following generalization of Tutte's Theorem has been proved by Lovász [25].

Theorem 3.34 (Lovász). *If $\tilde{A}(G)$ is the bipartite symbolic adjacency matrix of a bipartite graph G, then* $\operatorname{rank} \tilde{A}(G) = \nu(G)$.

If $\tilde{A}(G)$ is the symbolic adjacency matrix of a graph G, then $\operatorname{rank} \tilde{A}(G) = 2\nu(G)$.

Proof. In the biparite case, non-singular submatrices of $\tilde{A}(G)$ correspond to subgraphs of G having a perfect matching. The claim then follows immediately from Tutte's Theorem.

In the general case, this reasoning only works in one direction: if G has a matching of size s, then $\operatorname{rank} \tilde{A}(G) \geq 2s$.

Since $\tilde{A}(G)$ is skew-symmetric, it has even rank by Corollary 2.9. Let $\operatorname{rank} \tilde{A}(G) = 2s$. By Corollary 2.7 $\tilde{A}(G)$ has a non-singular principal submatrix $\tilde{A}(G)_{U,U}$ where $|U| = 2s$. By Tutte's Theorem $G[U]$ has a perfect matching of size s. □

Consider Algorithm 13.

Algorithm 13 Reduction of maximum matching to perfect matching

REDUCTION(G):
find a maximum non-singular submatrix $\tilde{A}(G)_{U,W}$ of $\tilde{A}(G)$
return U

Theorem 3.35. *For any graph $G = (V, E)$, Algorithm 13 finds a maximum set $U \subseteq V$, such that $G[U]$ has a perfect matching, using $O(n^\omega)$ operations.*

Proof. Since $\tilde{A}(G)$ is skew-symmetric, we have $\operatorname{rank} \tilde{A}(G)_{U,U} = \operatorname{rank} \tilde{A}(G)_{U,W}$ by Theorem 2.6, so $\tilde{A}(G)_{U,U}$ is a maximum non-singular principal submatrix of $\tilde{A}(G)$. The claim now follows from Tutte's Theorem.

By Theorem 2.16, a maximum non-singular submatrix $\tilde{A}(G)_{U,W}$ of $\tilde{A}$ can be found using $O(n^\omega)$ operations. □

Randomization of this algorithm follows the usual pattern.

Theorem 3.36. *For any prime $p = \Theta(n^2)$, the $\mathbb{F}_p$ implementation of Algorithm 4 is an $O(n^\omega)$ Monte Carlo algorithm finding a maximum set $U \subseteq V$, such that $G[U]$ has a perfect matching.*

Proof. This proof is similar to the proofs of Theorem 3.6 and Theorem 3.13.

Given a graph G, choose any maximum non-singular principal submatrix $\tilde{A}(G)_{U,U}$ of $\tilde{A}(G)$. The determinant $\det \tilde{A}(G)_{U,U}$ is a polynomial of degree $\leq n$ and has an $O(1)$ non-zero coefficient by Theorem 3.5. By Zippel-Schwartz Lemma, with probability $\geq 1 - n/p = 1 - O(1/n)$ this polynomial evaluates to a non-zero value for a random substitution $s : \mathbb{Z}(\tilde{E}) \to \mathbb{Z}_p$. We then have $\operatorname{rank} \tilde{A}(G)(s) = \operatorname{rank} \tilde{A}(G)$ and an $\mathbb{F}_p$ implementation of Algorithm 13 works.

The complexity bound follows from 2.36, the maximum non-singular submatrix is found using Gaussian elimination and Gaussian elimination only requires n divisions. □

Remark 3.37. *Another simple reduction, based on Theorem 3.34, has been proposed by Rabin and Vazirani [33]: Add $n - \nu(G)$ new vertices to G and connect each of these vertices with every vertex of G. The resulting graph has a perfect matching, and any such matching contains a maximum matching in G.*

This reduction is a bit simpler than the one we use, but it does not extend to the case of planar graphs.

Chapter 4

Perfect Matchings in Planar Graphs

In this chapter we apply the techniques introduced in the previous chapter to the problem of finding maximum matchings in planar graphs. Recall from Subsection 2.3.3 that under certain assumptions we can perform Gaussian elimination on a planar matrix in time $O(n^{\omega/2})$ using the nested dissection algorithm. We show that nested dissection can be used to solve the planar case of the maximum matching problem within the same time bounds.

In Section 4.1 we recall a well-known reduction of the planar case of the maximum (perfect) matching problem to the case of bounded degree planar graphs. In Section 4.2 we show an $O(n^{\omega/2})$ implementation of the Lovász's algorithm in the planar case. We also introduce the basic ingredients of our planar matching framework, which we then use to give an $O(n^{\omega/2})$ perfect matching algorithm for planar graphs in Section 4.3. Finally, in Section 4.4 we show that the maximum matching problems for planar graphs reduces in time $O(n^{\omega/2})$ to the perfect matching problem in planar graphs.

All the results presented in this chapter are based on the paper by Mucha and Sankowski [29], except for Section 4.1 which describes a well-known technique (see Wilson [37]).

4.1 Degree Reduction

We start by showing that we can restrict our considerations to the case of planar graphs with vertex degrees ≤ 3. This assumption will be extremely useful in the remainder of this chapter. The following is a well-known reduction:

Theorem 4.1. *The problem of finding perfect (maximum) matchings in planar graphs is reducible in $O(n)$ time to the problem of finding perfect (maximum) matchings in planar graphs with maximum vertex degree 3. This reduction adds $O(n)$ new vertices.*

Proof. We will prove the theorem using the technique of vertex splitting.

Suppose that G has a vertex v with degree > 3, and let $N(v)$ be the set of its neighbours. We choose 2 neighbours $w_1, w_2 \in N(v)$ and replace v with three vertices v_1, v_2, v_3 as shown in Fig. 4.1. The neighbours of v_1 are w_1, w_2 and v_2, the neighbours

of v_2 are v_1 and v_3, and the neighbours of v_3 are v_2 and all the neighbours of v except w_1 and w_2. There is a one-to-one mapping between perfect matchings in G and in $\hat{G}$, as shown in the figure.

A single splitting operation reduces the degree of a single high degree vertex by 2, and adds 2 new vertices. Reducing the degrees of all the vertices to ≤ 3 requires only $O(m) = O(n)$ splitting operations and adds $O(n)$ new vertices, so the resulting graph has $O(n)$ vertices.

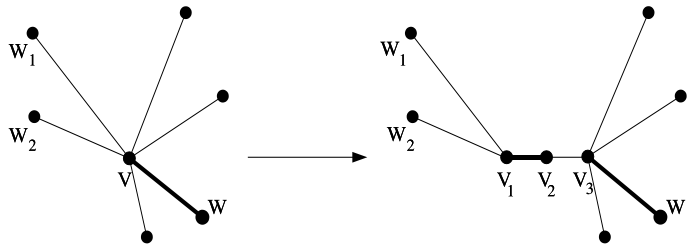

Figure 4.1: Vertex splitting. On the left there is a high degree vertex v. It is matched in the perfect matching with one of its neighbours w. On the right there is the graph after splitting this vertex into v_1, v_2, v_3. Now v_3 is matched with w and v_1 is matched with v_2. Perfect matchings in the two graphs are in one to one correspondence.

Even if G has no perfect matching, we can still use this reduction. The translation of perfect matchings in the original graph G to perfect matchings in the bounded degree graph $\hat{G}$ works for maximum matchings as well. In order to go from $\hat{G}$ to G, we need to first guarantee that v_2 is matched. Let M be a matching in $\hat{G}$. If v_2 is not matched in M, then either v_1 or v_3 has to be matched. Suppose v_1 is matched with some vertex u. Then $M \cup \{v_1v_2\} - \{v_1u\}$ is a maximum matching in which v_2 is matched. Once v_2 is matched we can use the correspondence shown in the figure to get a matching in G. Notice, however, that this translation is not one-to-one.

The number of unmatched vertices in a maximum matching is the same for G and $\hat{G}$. □

In the remainder of this chapter we assume that all the graphs have vertex degrees bounded by 3.

4.2 Testing Algorithm

We start by adapting Lovász's testing algorithm (Algorithm 4) to the planar case. This allows us to introduce the basic ingredients of the planar matching algorithm without getting involved in obscure details.

4.2.1 General Idea

In order to achieve an $O(n^{\omega/2})$ implementation of Lovász's testing algorithm, we would like to perform Gaussian elimination on $\tilde{A}(G)$ using $O(n^{\omega/2})$ operations. The problem

is that there is no easy way to avoid pivoting, which is required if we are going to use the nested dissection algorithm. A common solution to this kind of problem it to consider $\tilde{A}(G)\tilde{A}(G)^T$ instead. We get Algorithm 14.

Algorithm 14 Testing algorithm for planar graphs

PLANAR-TEST-PERFECT-MATCHING(G):
  compute $\tilde{B}(G) = \tilde{A}(G)\tilde{A}(G)^T$
  compute $\det \tilde{B}(G)$ using the nested dissection
  if $\det \tilde{B}(G) \neq 0$ **then**
    return "YES"
  else
    return "NO"

Notice that $\tilde{B}(G)$ is non-singular iff $\tilde{A}(G)$ is non-singular. In the next section we verify that performing the nested dissection on $\tilde{B}(G)$ requires only $O(n^{\omega/2})$ operations and in Section 4.2.3 we show that we can get a Monte Carlo testing algorithm by implementing Algorithm 14 over a finite field $\mathbb{F}_p$, for a suitable choice of p.

4.2.2 Symbolic Nested Dissection

In this subsection we show that performing the nested dissection on the matrix $\tilde{A}(G)\tilde{A}(G)^T$ in Algorithm 14 requires $O(n^{\omega/2})$ operations.

For any graph G, let $G^2 = (V, E_2)$, where $uv \in E_2$ if there exists a vertex $w \in V$ such that $uw, wv \in E$. Notice the following:

Lemma 4.2 (Thick Separator Lemma). *If G is a bounded degree planar graph and S is a small separator in G, then also $T = S \cup N(S)$ is a small separator in G^2.*

Proof. The fact that T is a separator in G^2 follows from the definition of G^2. The size of T is $O(\sqrt{n})$ because G has bounded degree. □

The separator T constructed in Lemma 4.2 will be called the *thick separator* corresponding to S. Obviously, T is also a small separator in G.

We are now ready to make the first step towards applying the nested dissection algorithm.

Theorem 4.3. *Let G be a bounded degree planar graph. Then $G(\tilde{A}(G)\tilde{A}(G)^T)$ has an $O(\sqrt{n})$-separator family.*

Proof. Notice that $G(\tilde{A}(G)\tilde{A}(G)^T) \subseteq G^2$. The assertion follows by recursive application of the Thick Separator Lemma. □

In order to use Theorem 2.27 we need to prove that no pivoting is required during the elimination of $\tilde{A}(G)\tilde{A}(G)^T$. For any non-singular matrix X over $\mathbb{R}$ (or $\mathbb{C}$), the matrix XX^T is symmetric positive definite (see Theorem 2.12) and Theorem 2.27 can

be used. In case of matrices over $\mathbb{Z}(\tilde{E})$ the usual notion of positive definiteness does not make sense, so we cannot use Theorem 2.27 directly.

Let us call a matrix X over $\mathbb{Z}(\tilde{E})$ *symmetric positive definite* if it has the form $X = YY^T$ for some non-singular Y. Notice, that we are mimicking the usual definition, due to the characterization given in Theorem 2.12.

We have the following:

Theorem 4.4 (Symbolic nested dissection). *Theorem 2.27 holds for matrices over $\mathbb{Z}(\tilde{E})$, i.e. if X is a symmetric positive definite matrix over $\mathbb{Z}(\tilde{E})$ and $G(X)$ has an $O(\sqrt{n})$-separator family, then given an $O(\sqrt{n})$ separator tree for $G(X)$, Gaussian elimination on X can be performed using $O(n^{\omega/2})$ operations. The resulting factorization of X has the form $X = LDL^T$, where L is unit lower-triangular and has $O(n \log n)$ non-zero entries and D is diagonal.*

Proof. Let $X = YY^T$ be a symmetric positive definite matrix over $\mathbb{Z}(\tilde{E})$. We need to guarantee that no pivoting is required during the elimination of X. Since $\det Y \neq 0$, there exists a substitution $s : \tilde{E} \to \mathbb{R}$, such that $\det Y(s) \neq 0$.

Since $Y(s)$ is a non-singular real matrix, $X(s) = Y(s)Y(s)^T$ is symmetric positive definite in the usual sense. By Theorem 2.27 (and Remark 2.29), no pivoting is required during the elimination of $X(s)$. The same has to be true for X. □

Using Theorem 4.4 we get the following:

Theorem 4.5. *Algorithm 14 tests whether a given planar graph G has a perfect matching using $O(n^{\omega/2})$ operations.*

4.2.3 Working over a Finite Field

In this subsection we show that, for a random prime $p = \Theta(n^4)$, an $\mathbb{F}_p$ implementation of Algorithm 14 is a Monte Carlo testing algorithm.

It is the first time we need a random large prime and not just any large prime. This is because polynomials computed by Algorithm 14 have very large coefficients and it is not easy to guarantee that they are non-zero over $\mathbb{F}_p$.

In particular, we need the following theorem, stronger than Theorem 2.33.

Theorem 4.6. *Let $T(n)$ and $D(n)$ be such that $T(n), D(n) = \Omega(n)$ and $T(n), D(n) = O(\mathrm{poly}(n))$. Let $\mathcal{A}$ be an algorithm performing at most $T(n)$ zero-tests in $\mathbb{Z}(X)$ arithmetic for any input data of size n. Suppose that every $f \in \mathrm{gen}_{\mathcal{A}}(n)$ has a representation $f = g_f/h_f$, such that $\deg g_f \leq D(n)$ and g_f has an $O((nT(n)D(n))^n)$ non-zero coefficient. Then, with high probability, the $\mathbb{F}_p$ implementation of $\mathcal{A}$ correctly answers all zero-tests, for a random prime $p = \Theta(nT(n)D(n))$.*

Proof. Consider any input D of size n. We first prove that with high probability all $f \in \mathrm{gen}_{\mathcal{A}}(D)$ are not identically zero over $\mathbb{F}_p$.

For each $f \in \mathrm{gen}_{\mathcal{A}}(D)$, choose a representation $f = g_f/h_f$, such that $\deg g_f \leq D(n)$ and g_f has an $O((nT(n)D(n))^n)$ non-zero coefficient c_f. Since c_f can only have $O(n)$ prime divisors of order $\Theta(nT(n)D(n))$, we have at most $O(nT(n))$ distinct prime

divisors of order $\Theta(nT(n)D(n))$ for all the c_f, $f \in \text{gen}_{\mathcal{A}}(D)$. Let p be a random prime of order $\Theta(nT(n)D(n))$. If p does not divide any of c_f, then all g_f are non-zero over $\mathbb{F}_p$. Since there are $\Theta(nT(n)D(n)/\log n)$ distinct primes of order $O(nT(n)D(n))$, the probability of that event is $1 - O(\frac{nT(n)}{nT(n)D(n)}) = 1 - O(1/n)$.

We can now use the Zippel-Schwartz Lemma. Since all g_f have degree $\leq D(n)$, the probability of getting a false zero testing any of them is $O(\frac{D(n)}{nT(n)D(n)}) = O(\frac{1}{nT(n)})$. The sum of these probabilities over all g_f gives $O(1/n)$, so the probability that all zero-tests give correct answers is $1 - O(1/n)$. □

All the rational functions zero-tested by Algorithm 14 are entries of the intermediate results of Gaussian elimination performed on $\tilde{A}\tilde{A}^T$. In order to use Theorem 4.6 we need to prove that all these functions satisfy its assumptions.

First notice that the elements of $\tilde{A}\tilde{A}^T$ have a very simple form.

Lemma 4.7. *Non-zero elements of $\tilde{A}\tilde{A}^T$ are polynomials consisting of at most 3 different monomials, all of degree 2 and with ± 1 coefficients. Moreover, there are at most 10 non-zero entries in each row or column of $\tilde{A}\tilde{A}^T$.*

Proof. The first part of the theorem follows directly from the fact that all vertices of G have degree at most 3. To prove the second part, notice that $(\tilde{A}\tilde{A}^T)_{i,j}$ can be only non-zero if either $i = j$ or i and j have a common neighbour. There are at most 10 such j for every i. □

For any polynomial $f \in \mathbb{Z}(\tilde{E})$, let the *weight* of f, denoted $|f|$, be the sum of the absolute values of coefficients of f. Notice that $|fg| \leq |f||g|$, $|f + g| \leq |f| + |g|$.

We have the following bound.

Lemma 4.8. *The weight of a determinant of any submatrix of $\tilde{A}\tilde{A}^T$ is $O(n^{2n})$. The degree of any such determinant is $O(n)$.*

Proof. The determinant of a $k \times k$ submatrix of $\tilde{A}\tilde{A}^T$ is the sum of at most $k!$ products, each of them consisting of exactly k non-zero entries of $\tilde{A}\tilde{A}^T$. By Lemma 4.7, expanding this determinant gives at most $3^k k! = O(n^{2n})$ terms, all with ± 1 coefficients, so the weight of the determinant is at most $O(n^{2n})$. Obviously, the degree of the determinant is $O(n)$. □

The adjoint formula for the inverse matrix (Theorem 2.4) gives the following.

Corollary 4.9. *The entries of the inverse of any non-singular submatrix of $\tilde{A}\tilde{A}^T$ have form g/h, where g and h are polynomials of degree $O(n)$ and weight $O(n^{2n})$.*

Theorem 4.10. *At any stage of Gaussian elimination performed on the matrix $\tilde{A}(G)\tilde{A}(G)^T$, the non-zero entries of the uneliminated part of $\tilde{A}\tilde{A}^T$ have form g/h, where both g and h are polynomials of degree $O(n)$ and weight $O(n^{2n})$.*

Proof. Let $B = \tilde{A}\tilde{A}^T$. Suppose that k steps of Gaussian elimination have been performed on B and let B' be resulting matrix. Let $S = \{1, \ldots, k\}$ be the set of eliminated rows (columns), and let $T = \{k+1, \ldots, n\}$ be the set of uneliminated rows

(columns). When nested dissection is performed on B, we only need the elements from the part of the matrix that was not yet eliminated, i.e. the elements of $B'_{T,T}$. By Theorem 2.18, we have $B'_{T,T} = B_{T,T} - B_{T,S}B_{S,S}^{-1}B_{S,T}$. Non-zero entries in $B_{T,S}$ and $B_{S,T}$ are polynomials of weight ≤ 3 and there are at most 10 such entries in every row or column, by Lemma 4.7. Non-zero entries of $B_{S,S}^{-1}$ have form g/h, where g and h are polynomials of weight $O(n^{2n})$, by Corollary 4.9 (also note that h can be assumed to be the same for all non-zero entries). This gives a weight bound of $O(10 \cdot 10 \cdot 3 \cdot 3 \cdot n^{2n}) = O(n^{2n})$ for numerators and denominators of the entries of $B_{T,S}B_{S,S}^{-1}B_{S,T}$. The bound holds for $B'_{T,T}$ as well, because entries of $B_{T,T}$ have weights at most 3. The degree bound is obvious. □

We are now ready to use Theorem 4.6.

Theorem 4.11. *For a random prime $p = \Theta(n^{2+\omega/2})$, the $\mathbb{F}_p$ implementation of Algorithm 14 is an $O(n^{\omega/2})$ Monte Carlo algorithm deciding whether a given planar graph G has a perfect matching.*

Proof. We apply Theorem 4.6 with $T(n) = O(n^{\omega/2})$ and $D(n) = O(n)$. The assumptions of this theorem are satisfied, due to Theorem 4.10. □

4.3 The Matching Algorithm

We now present the matching algorithm for planar graphs. We use the ideas introduced in the previous section: performing Gaussian elimination on the matrix $\tilde{A}(G)\tilde{A}(G)^T$ and randomization based on Theorem 4.6 and Theorem 4.10.

4.3.1 General Idea

Our matching algorithm (Algorithm 15) is a recursive algorithm based on the separator decomposition of planar graphs. We start by finding a small separator, next we find an allowed matching incident on all its vertices, and then find perfect matchings in connected components of the unmatched portion of the graph.

Algorithm 15 Matching algorithm for planar graphs

PLANAR-PERFECT-MATCHING(G):
  let S be a small separator in G
  let T be the thick separator corresponding to S
  find the factorization $\tilde{A}(G)\tilde{A}(G)^T = LDL^T$ using the nested dissection with T as a top-level separator
  compute $(\tilde{A}(G)^{-1})_{T,T}$ using L and D
  use $(\tilde{A}(G)^{-1})_{T,T}$ to find an allowed matching M incident on all vertices of S
  find perfect matchings in connected components of $G - V(M)$

In the remainder of this section, we show that both computing $(\tilde{A}(G)^{-1})_{T,T}$ and finding an allowed matching only require $O(n^{\omega/2})$ operations. This gives the complexity bound of $O(n^{\omega/2})$ operations for the whole algorithm as well.

Remark 4.12. *The above statement is not true if $\omega = 2$. In this case, additional $O(\log^2 n)$ factor appears in the complexity of Algorithm 15. In the remainder we assume that $\omega > 2$ to avoid unnecessary complications.*

4.3.2 Computing the important part of $\tilde{A}^{-1}$

By performing the nested dissection algorithm on the matrix $\tilde{A}\tilde{A}^T$, we find $n \times n$ matrices L and D such that $\tilde{A}\tilde{A}^T = LDL^T$. We now show how L and D can be used to compute $(\tilde{A}^{-1})_{T,T}$ with $O(n^{\omega/2})$ operations. Let us represent $\tilde{A}$, L and D as block matrices

$$\tilde{A} = \begin{pmatrix} \tilde{A}_{U,U} & \tilde{A}_{U,T} \\ \tilde{A}_{T,U} & \tilde{A}_{T,T} \end{pmatrix}, \quad D = \begin{pmatrix} D_{U,U} & 0 \\ 0 & D_{T,T} \end{pmatrix},$$

$$L = \begin{pmatrix} L_{U,U} & 0 \\ L_{T,U} & L_{T,T} \end{pmatrix}, \quad L^{-1} = \begin{pmatrix} L_{U,U}^{-1} & 0 \\ -L_{T,T}^{-1}L_{T,U}L_{U,U}^{-1} & L_{T,T}^{-1} \end{pmatrix},$$

where lower right blocks in all matrices correspond to the vertices of the thick separator T, and $U = V - T$. Since $\tilde{A}\tilde{A}^T = LDL^T$, we have

$$(\tilde{A}^T)^{-1} = (L^T)^{-1}D^{-1}L^{-1}\tilde{A},$$

where the interesting part of $(\tilde{A}^T)^{-1}$ is

$$\begin{aligned} (\tilde{A}^T)^{-1}_{T,T} &= ((L^T)^{-1})_{T,V}D^{-1}L^{-1}\tilde{A}_{V,T} = \\ &= (L^T_{T,T})^{-1}D^{-1}_{T,T}(L^{-1})_{T,V}\tilde{A}_{V,T} = \\ &= (L^T_{T,T})^{-1}D^{-1}_{T,T}L^{-1}_{T,T}\tilde{A}_{T,T} + (L^T_{T,T})^{-1}D^{-1}_{T,T}(L^{-1})_{T,U}\tilde{A}_{U,T}. \end{aligned}$$

The first component can be easily computed with $O(n^{\omega/2})$ operations using fast matrix multiplication. The second component can be written as

$$(L^T_{T,T})^{-1}D^{-1}_{T,T}(L^{-1})_{T,U}\tilde{A}_{U,T} = -(L^T_{T,T})^{-1}D^{-1}_{T,T}L^{-1}_{T,T}L_{T,U}L^{-1}_{U,U}\tilde{A}_{U,T}$$

and the only hard part here is to compute $X = -L_{T,U}L^{-1}_{U,U}\tilde{A}_{U,T}$. Consider the matrix

$$B = \begin{pmatrix} L_{U,U} & \tilde{A}_{U,T} \\ L_{T,U} & 0 \end{pmatrix}.$$

When Gaussian elimination is performed on rows and columns in U, the lower right submatrix becomes X (see Theorem 2.18). The elimination can be performed with use of the nested dissection algorithm in time $O(n^{\omega/2})$. This is possible, because the separator tree for $\tilde{A}\tilde{A}^T$ is a valid separator tree for B (recall Fact 2.30). Since $L_{U,U}$ is lower-triangular, there are no problems with diagonal zeros, even though B is not symmetric positive definite.

4.3.3 Matching the Separator

We now show how the separator vertices can be matched using the matching verification algorithm. Consider Algorithm 16.

Algorithm 16 A procedure for finding an allowed matching of the separator.

FIND-ALLOWED-SEPARATOR-MATCHING:
  $M := \emptyset$
  $G_T := (T, E(T) - E(T - S))$
  repeat
    let M_G be a maximal matching in G_T using only allowed edges
    use $(\tilde{A}^{-1})_{T,T}$ to find a maximal allowed submatching M'_G of M_G
    $M := M \cup M'_G$
    $G_T = G_T - V(M'_G)$
    mark edges in $M_G - M'_G$ as not allowed
  until M matches all vertices of S

Lemma 4.13. *Algorithm 16 finds an allowed matching of the separator.*

Proof. The algorithm first finds a maximal matching M_G of S using only allowed edges. All the edges of M_G are then either included in M or marked as not allowed and the whole procedure is repeated. Since G_T is always non-empty, the algorithm stops after a finite number of iterations. □

Lemma 4.14. *Algorithm 16 makes at most* 5 *iterations.*

Proof. Consider the allowed matching M found by Algorithm 16 and a matching M_G constructed in any iteration. For every edge $e \in M$, either e incident with at last one edge of $e' \in M_G$, or $e \in M_G$, because of the maximality of M_G. If e is in M_G, it is immediately added to M. Otherwise, an edge $e' \in M_G$ incident to e is marked as not allowed. Every edge $e \in M$ has at most 4 incident edges, so the main loop of Algorithm 16 is executed at most 5 times. □

Lemma 4.15. *Each iteration of Algorithm 16 requires $O(n^{\omega/2})$ operations.*

Proof. The dominating step of Algorithm 16 is finding the maximal allowed submatching M'_G of M_G. This can be done using the verification algorithm (see Theorem 3.21) with $O(n^{\omega/2})$ operations. This algorithm works on the matrix $\tilde{A}(G)^{-1}$, but it never uses any values from outside the submatrix $(\tilde{A}(G)^{-1})_{T,T}$.

Let $\tilde{A}'$ be the result of running the verification algorithm on the matrix $(\tilde{A}^{-1})_{T,T}$. Due to Theorem 3.17, $\tilde{A}'_{T',T'} = (\tilde{A}(G - V(M'_G))^{-1})_{T',T'}$, where T' is obtained from T by removing the vertices matched by M'_G. Thus, the inverse $\tilde{A}(G)^{-1}_{T,T}$ does not need to be computed from scratch in every iteration. □

The following Theorem follows from Lemmas 4.13, 4.14 and 4.15.

Theorem 4.16. *Algorithm 16 finds an allowed matching of the separator using $O(n^{\omega/2})$ operations.*

4.3.4 Working over a Finite Field

We have the following theorem, similar to Theorem 4.11.

Theorem 4.17. *For a random prime $p = \Theta(n^{2+\omega/2})$, the $\mathbb{F}_p$ implementation of Algorithm 15 is an $O(n^{\omega/2})$ Monte Carlo algorithm finding a perfect matching in a given planar graph G having one.*

Proof. The proof is similar to the proof of Theorem 4.11. The rational functions zero-tested by the algorithm are entries of:

1. partially eliminated $\tilde{A}\tilde{A}^T$, or
2. partially eliminated $\tilde{A}^{-1}_{T,T}$, or
3. partially eliminated $L_{U,U}$.

All these functions satisfy the assumptions of Theorem 4.10. The entries of partially eliminated $\tilde{A}\tilde{A}^T$ have been considered in the analysis of the testing algorithm (Algorithm 14). By the Elimination Theorem (Theorem 3.17), partially eliminated $\tilde{A}^{-1}_{T,T}$ is just an inverse of a submatrix of $\tilde{A}_{T,T}$, so they satisfy these assumptions by the adjoint formula. Finally, the diagonal entries of $L_{U,U}$ (only diagonal entries are tested when nested dissection is used) are all equal to 1.

The assertion of the theorem follows from Theorem 4.10. □

4.4 Maximum Matchings

We now show that the problem of finding a maximum matching in a planar graph can be reduced to the problem of finding a perfect matching in a planar graph.

We use the method introduced in Section 3.7. Algorithm 17 is a natural adaptation of Algorithm 13 to our planar framework, but it is not clear at all that it works.

Algorithm 17 Reduction of maximum matching to perfect matching (planar case)

REDUCTION(G):
perform the nested dissection with skipping on $\tilde{A}(G)\tilde{A}(G)^T$
return the set U of eliminated rows (columns)

Let us first prove the following:

Lemma 4.18. *For any subset $U \subseteq V$, $\tilde{A}(G)_{U,U}$ is non-singular iff $(\tilde{A}(G)\tilde{A}(G)^T)_{U,U}$ is non-singular.*

Proof. Let $\tilde{B} = \tilde{A}\tilde{A}^T$.

Suppose $\tilde{B}_{U,U}$ is non-singular. We have $\tilde{B}_{U,U} = \tilde{A}_{U,V}(\tilde{A}^T)_{V,U}$ so $\operatorname{rank}\tilde{B}_{U,U} = \operatorname{rank}\tilde{A}_{U,V}$. But since $\tilde{A}$ is skew-symmetric, we also have $\operatorname{rank}\tilde{A}_{U,V} = \operatorname{rank}\tilde{A}_{U,U}$ by Theorem 2.6, so $\tilde{A}_{U,U}$ is non-singular.

Conversely, let $\tilde{A}_{U,U}$ be non-singular. We have

$$\tilde{B}_{U,U} = \tilde{A}_{U,V}(\tilde{A}^T)_{V,U} = -\tilde{A}_{U,V}\tilde{A}_{V,U} = -\tilde{A}_{U,U}\tilde{A}_{U,U} - \tilde{A}_{U,V-U}\tilde{A}_{V-U,U},$$

Since $\tilde{A}_{U,U}\tilde{A}_{U,U}$ and $\tilde{A}_{U,V-U}\tilde{A}_{V-U,U}$ use disjoint sets of variables, $\tilde{B}_{U,U}$ is non-singular. □

We are now ready to prove the following:

Theorem 4.19. *For any graph $G = (V, E)$, Algorithm 17 finds a maximum set $U \subseteq V$, such that $G[U]$ has a perfect matching, using $O(n^{\omega/2})$ operations.*

Proof. Algorithm 17 finds a maximum non-singular principal submatrix $(\tilde{A}\tilde{A}^T)_{U,U}$ of $\tilde{A}\tilde{A}^T$. By Lemma 4.18 this corresponds to a maximum non-singular principal submatrix $\tilde{A}_{U,U}$ of $\tilde{A}$, which by Tutte's Theorem gives a maximum set U, such that $G[U]$ has a perfect matching.

Since we can assume that G has degree bounded by 3 (recall that Theorem 4.1 works for maximum matchings as well), the complexity is $O(n^{\omega/2})$. □

Following the usual pattern we get:

Theorem 4.20. *For a random prime $p = \Theta(n^{2+\omega/2})$, the $\mathbb{F}_p$ implementation of Algorithm 17 is an $O(n^{\omega/2})$ Monte Carlo algorithm finding a maximum set $U \subseteq V$, such that $G[U]$ has a perfect matching.*

Remark 4.21. *Actually, a random prime $p = \Theta(n^2)$ is sufficient. Note that for the $\mathbb{F}_p$ implementation to work it is enough if $\operatorname{rank}\tilde{A}\tilde{A}^T = \operatorname{rank}(\tilde{A}\tilde{A}^T)(s)$, where s is a substitution with elements of $\mathbb{F}_p$. We can choose any maximum non-singular submatrix $X = (\tilde{A}\tilde{A}^T)_{U,U}$ of $\tilde{A}\tilde{A}^T$ and try to guarantee that $\det X(s) \neq 0$. By Lemma 4.8, $\det X$ has a non-zero $O(n^{2n})$ coefficient and by the prime-density argument, $\det X$ is non-zero over $\mathbb{F}_p$ for a random prime $p = \Theta(n^2)$. We can then use the Zippel-Schwartz Lemma.*

Chapter 5

Randomization

The theme of this chapter is constructing Las Vegas versions of the algorithms presented so far.

In Section 5.1 we use the standard technique (see for example Karloff [19] or Cheriyan [6]), based on the Gallai-Edmonds decomposition of a graph, to show that all our algorithms for the maximum matching problem can be made Las Vegas, except for the planar case algorithm (Algorithm 15).

In Section 5.2 we discuss the problem of making the $\sim_G$ (see Subsection 3.6.2 for a definition) computation Las Vegas and give some partial results, based on ideas of Cheriyan [6]. We also consider the related problem of making the matching verification algorithm Las Vegas.

5.1 Making the Maximum Matching Algorithms Las Vegas

Recall that in order to make any Monte Carlo algorithm a Las Vegas one, we need to be able to verify the correctness of its output (see Fact 2.31). In case of a maximum matching algorithm this amounts to deciding if the matching found by the algorithm is indeed maximum. The standard method, also used in the parallel setting (see Karloff [19]), is based on the Gallai-Edmonds decomposition. Cheriyan has already shown in [6] that it can be implemented in time $O(n^\omega)$. In this section, we use his results to make our matching algorithms Las Vegas.

5.1.1 Gallai-Edmonds Decomposition

We start by providing a few more definitions. A vertex $v \in V$ of a graph $G = (V, E)$ is called *critical* if v is matched in all maximum matchings of G. Otherwise v is called *non-critical*. By $D(G)$ we will denote the set of all non-critical vertices of G, by $A(G)$ the set of all critical vertices of G that are adjacent to some non-critical vertices, and by $C(G) = V(G) \setminus (D(G) \cup A(G))$ the set of the remaining vertices.

We are now ready to give the statement of the Gallai-Edmonds decomposition theorem.

Theorem 5.1 (Gallai-Edmonds decomposition). *Let $G = (V, E)$ be a graph and let $D = D(G)$, $A = A(G)$ and $C = C(G)$ be defined as above. Then:*

- *each component of $G[C]$ has a perfect matching,*
- *each component of $G[D]$ is factor-critical,*
- *every maximum matching of G contains a perfect matching of $G[C]$, a near perfect matching of each component of $G[D]$, and matches all vertices of A with vertices in different components of $G[D]$,*
- *every maximum matching of G misses $\mathrm{comp}(G[D]) - |A|$ vertices, i.e. $n - 2\nu(G) = \mathrm{comp}(G[D]) - |A|$.*

The last part of Gallai-Edmonds Theorem is especially important. It means that using the Gallai-Edmonds decomposition of a graph G one can easily verify if a given matching M of G is maximum: it is enough to test if the equality $n - 2|M| = \mathrm{comp}(G[D]) - |A|$ holds.

5.1.2 Cheriyan's Algorithm

In this subsection we describe a Monte Carlo algorithm, due to Cheriyan [6], finding the Gallai-Edmonds decomposition of a given graph G in time $O(n^\omega)$.

Lemma 5.2. *Let G be a graph, let $\tilde{A} = \tilde{A}(G)$ be its symbolic adjacency matrix. Then, for any vertex v_i of G,* $\operatorname{rank}\begin{pmatrix} \tilde{A} \\ e_i^T \end{pmatrix} > \operatorname{rank}\tilde{A}$ *iff v_i is non-critical. Otherwise* $\operatorname{rank}\begin{pmatrix} \tilde{A} \\ e_i^T \end{pmatrix} = \operatorname{rank}\tilde{A}$.

Proof. Consider the graph $\hat{G} = (V \cup \{v_{n+1}\}, E \cup \{v_i v_{n+1}\})$, i.e. $\hat{G}$ is equal to G with an additional vertex v_{n+1}, connected with v_i only. We have

$$\tilde{A}(\hat{G}) = \begin{pmatrix} \tilde{A}(G) & x_{i,n+1}e_i \\ -x_{i,n+1}e_i^T & 0 \end{pmatrix}.$$

If v_i is critical in G, then $\nu(\hat{G}) = \nu(G)$, so $\operatorname{rank}\tilde{A}(\hat{G}) = \operatorname{rank}\tilde{A}(G)$, by Theorem 3.34. We then have $\operatorname{rank}\tilde{A} = \operatorname{rank}\begin{pmatrix} \tilde{A} \\ -x_{i,n+1}e_i^T \end{pmatrix}$, and so $\operatorname{rank}\tilde{A} = \operatorname{rank}\begin{pmatrix} \tilde{A} \\ e_i^T \end{pmatrix}$.

Similarly, if v_i is non-critical in G, then $\nu(\hat{G}) = \nu(G) + 1$, so $\operatorname{rank}\tilde{A}(\hat{G}) = \operatorname{rank}\tilde{A}(G) + 2$, by Theorem 3.34. We then have $\operatorname{rank}\begin{pmatrix} \tilde{A} \\ -x_{i,n+1}e_i^T \end{pmatrix} = \operatorname{rank}\tilde{A} + 1$, and so $\operatorname{rank}\begin{pmatrix} \tilde{A} \\ e_i^T \end{pmatrix} = \operatorname{rank}\tilde{A} + 1 > \operatorname{rank}\tilde{A}$, as claimed. □

Lemma 5.2 characterizes critical and non-critical vertices in terms of $\tilde{A}(G)$. Algorithm 18 uses this characterization to find the set $C[G] \cup A[G]$ of critical vertices of G.

Algorithm 18 Cheriyan's algorithm the finding critical vertices of G

CRITICAL-VERTICES(G):
{ preprocessing, for randomization purposes only }
find a maximum set $W \subseteq V$ such that $G[W]$ has a perfect matching
renumber the vertices of G so that the vertices of W have numbers $1, \ldots, |W|$
{ the actual algorithm }
find a basis $k_1, \ldots, k_{n-2\nu(G)}$ of $\ker \tilde{A}(G)$
return $K = \{v_i : \forall_j (k_j)_i = 0\}$

Remark 5.3. *The first two lines of Algorithm 18 are only necessary to make the randomization of the algorithm easier. They have no impact on the algorithm's correctness.*

Theorem 5.4. *For a given graph G, Algorithm 18 finds the set of critical vertices of G, using $O(n^\omega)$ operations.*

Proof. First, let us note that Algorithm 18 indeed uses $O(n^\omega)$ operations, by Corollary 2.25.

We need to prove, that the set $K \subseteq V$ returned by Algorithm 18 is the set of critical vertices of G.

Let $S = \operatorname{span}\{\tilde{A}^T_{1,\cdot}, \tilde{A}^T_{2,\cdot}, \ldots, \tilde{A}^T_{n,\cdot}\}$, i.e. the vector space spanned by (transposed) rows of $\tilde{A}$. We have $\ker \tilde{A} = S^\perp$, so by Theorem 2.3, $S = (\ker \tilde{A})^\perp$. By Lemma 5.2 v_i is critical iff $\operatorname{rank} \begin{pmatrix} \tilde{A} \\ e_i^T \end{pmatrix} = \operatorname{rank} \tilde{A}$ and this equality holds iff $e_i \in S$. Thus v_i is critical iff $e_i \in (\ker \tilde{A})^\perp$, which is exactly what Algorithm 18 tests. □

Lemma 5.5. *The entries of the basis vectors $k_1, \ldots, k_{n-2\nu(G)}$ have form g/h, where g is a polynomial of degree $O(n)$ and has an $O(1)$ non-zero coefficient.*

Proof. Recall how the basis of $\ker \tilde{A}$ is found. We perform Gaussian elimination on $\tilde{A}$ and find a factorization $\tilde{A} = LDU$, where $D = \operatorname{diag}(a_1, a_2, \ldots, a_{2\nu(G)}, 0, 0, \ldots, 0)$. Then, $\ker \tilde{A}$ is spanned by the vectors $U^{-1}(e_i)$ for $i = 2\nu(G) + 1, n$. (Here, we assumed that no pivoting is neccessary. Later, we will show that this assumption can be omitted.)

By Theorem 2.18, we have

$$U = \begin{pmatrix} U_W & D_W^{-1} L_W^{-1} \tilde{A}_{W,V-W} \\ 0 & I_{n-k} \end{pmatrix},$$

where $\tilde{A}_{W,W} = L_W D_W U_W$. Inverting U gives

$$U^{-1} = \begin{pmatrix} U_W^{-1} & U_W^{-1} D_W^{-1} L_W^{-1} \tilde{A}_{W,V-W} \\ 0 & I_{n-k} \end{pmatrix} = \begin{pmatrix} U_W^{-1} & \tilde{A}_{W,W}^{-1} \tilde{A}_{W,V-W} \\ 0 & I_{n-k} \end{pmatrix},$$

so the basis of $\ker \tilde{A}$ found by Algorithm 18 consists of the columns of the matrix

$$\begin{pmatrix} \tilde{A}_{W,W}^{-1}\tilde{A}_{W,V-W} \\ I_{n-k} \end{pmatrix}$$

.

We need to prove that the entries of $\tilde{A}_{W,W}^{-1}\tilde{A}_{W,V-W}$ satisfy the assertions of the theorem.

By Lemma 3.22, entries of $\tilde{A}_{W,W}^{-1}$ have form $g/\det \tilde{A}_{W,W}$, where g has degree $O(n)$ and an $O(1)$ non-zero coefficient. Also, the entries of $\tilde{A}_{W,V-W}$ have form $\pm x_{i,j}$, and no two entries use the same $x_{i,j}$. The claim follows.

Note that the assumption that no pivoting is necessary during the elimination of $\tilde{A}$ can easily be dropped. The important thing is that the eliminated rows and columns of $\tilde{A}$ correspond to W. Whether they are permuted does not matter. □

Theorem 5.6. *For any prime $p = \Theta(n^3)$, the $\mathbb{F}_p$ implementation of Algorithm 18 is an $O(n^\omega)$ Monte Carlo algorithm finding the set of critical vertices of a given graph G.*

Proof. Let $s : \tilde{E} \to \mathbb{F}_p$ be any substitution. When the $\mathbb{F}_p$ implementation of Algorithm 18 is performed on $\tilde{A}(s)$, it gives correct results if only it picks the set W correctly (i.e. the same as the symbolic version), and if for any non-critical vertex v_i at least one of the entries $(k_j)_i$ is non-zero.

By Lemma 5.5 and the Zippel-Schwartz Lemma, the second of these conditions is satisfied with probability $1 - O(1/n)$.

The only problem is how to bound the probability of picking the correct set W. Assume without loss of generality that Algorithm 18 picks W by finding a maximum non-singular submatrix $\tilde{A}_{U,W}$ of $\tilde{A}$ using Gaussian elimination, i.e. the set W is the column set of the submatrix found. Since Gaussian elimination always finds the maximum submatrix $\tilde{A}_{U,W}$ with W lexicographically first, the $\mathbb{F}_p$ implementation picks W as well, if only $\det \tilde{A}(s)_{W,W} \neq 0$. By Theorem 3.5 and the Zippel-Schwartz Lemma, the probability of this event is $1 - O(1/n^2)$.

As usual, the time complexity of the $\mathbb{F}_p$ implementation is $O(n^\omega)$, because Gaussian elimination only uses n divisions. □

As a consequence, we get the following $O(n^\omega)$ Monte Carlo algorithm for finding the Gallai-Edmonds decomposition of G (Algorithm 19).

Algorithm 19 Cheriyan's algorithm finding the Gallai-Edmonds decomposition

GALLAI-EDMONDS(G):
  $K :=$ CRITICAL-VERTICES(G) { the $\mathbb{F}_p$ implementation for $p = \Theta(n^3)$ }
  $D := V - K, C := K - N(D), A := K \cap N(D)$
  return C, A, D

5.1.3 Las Vegas algorithms

We first present a Las Vegas reduction of the maximum matching problem to the perfect matching problem (Algorithm 20).

Algorithm 20 Las Vegas reduction of maximum matching to perfect matching

CHERIYAN-REDUCTION(G):
  find the sets C, A, D
  let $\text{comp}_{odd}(G - A)$ be the number of odd sized components in $G - A$
  find a maximum set $U \subseteq V$ such that $G[U]$ has a perfect matching
  if $n - |U| = \text{comp}_{odd}(G - A) - |A|$ **then**
    return U
  else
    return "FAILURE"

Remark 5.7. *Cheriyan performs some additional verification of the sets C, A, D (e.g. whether all components of $G[D]$ have odd size, etc.) but this is not really necessary.*

Theorem 5.8. *Algorithm 20 is an $O(n^\omega)$ Las Vegas reduction of the maximum matching problem to the perfect matching problem.*

Proof. If the equality $n - |U| = \text{comp}_{odd}(G - A) - |A|$ holds, then U is maximum, because any matching has to miss at least one vertex from at least $\text{comp}_{odd}(G-A)-|A|$ odd components of $G - A$. Moreover, if both U and C, A, D are computed correctly, then the equality does hold, due to Gallai-Edmonds Theorem. Since both C, A, D and U are found using Monte Carlo algorithms, with high probability Algorithm 20 finds a maximum set U such that $G[U]$ has a perfect matching, otherwise it reports failure. It is thus a Las Vegas algorithm, as claimed. □

Theorem 5.9. *The Monte Carlo implementations of the $O(n^3)$ matching algorithms (Algorithm 8 and Algorithm 7) and the $O(n^\omega)$ matching algorithms (Algorithm 9 and Algorithm 10) can be made Las Vegas using Algorithm 20.*

Proof. This is obvious. We just perform any of these algorithms on a subgraph $G[U]$ found by the Las Vegas reduction, and if it does not find a perfect matching, we report failure. □

Remark 5.10. *In fact, we can do even better.*

Recall that the Monte Carlo reduction used in Algorithm 20 not only finds a set U such that $G[U]$ has a perfect matching, i.e. $\det \tilde{A}_{U,U} \neq 0$, but also a substitution s such that $\det \tilde{A}_{U,U}(s) \neq 0$.

Once such substition is known, all the matching algorithms listed in the statement of Theorem 5.9, except for the $O(n^\omega)$ general matching algorithm (Algorithm 10), are deterministic! To see this, recall the proof of Theorem 3.13.

Remark 5.11. *Most of the considerations of this section can be adapted to the planar case using the ideas from Chapter 4. However, Algorithm 18 cannot be implemented in time $O(n^{\omega/2})$, because the basis vectors $k_1, \ldots, k_{n-2\nu(G)}$ and the matrix $\tilde{A}^{-1}_{W,W}$ used to compute them might not be sparse (see also Chapter 6).*

5.2 Las Vegas algorithm for computing $\sim_G$

By the Rabin-Vazirani Theorem (Theorem 3.10), the matrix $\tilde{A}(G)^{-1}$ encodes the relation $\sim_G$ defined in Subsection 3.6.2. As a consequence, we get an $O(n^\omega)$ Monte Carlo algorithm for computing $\sim_G$: compute $\tilde{A}(G)^{-1}(s)$ for a random substitution $s : \tilde{E} \to \mathbb{F}_p$, where p is a suitable prime. It is interesting to ask whether this algorithm can be made Las Vegas.

One of the consequences would be a Las Vegas version of the matching verification algorithm given in Theorem 3.21. Recall that the matching verification algorithm finds, for a given matching M in $G = (V, E)$, a maximal allowed submatching M' of M. In order to make this algorithm Las Vegas, we need to verify that M' is indeed maximum, i.e. that all the edges of $M \setminus M'$ are not allowed in $G - V(M')$. This information is encoded in $\sim_G$.

The next two theorems are due to Cheriyan [6] and give some partial results.

Theorem 5.12. *The set of allowed edges of a bipartite graph can be computed in $O(n^\omega)$ time Las Vegas.*

The proof of Theorem 5.12 can be found in [6]. Note that only the set of allowed edges is computed here and not the whole $\sim_G$ relation.

Theorem 5.13. *If G is elementary, then $\sim_G$ can be computed in $O(n^\omega)$ time Las Vegas.*

Since Cheriyan does not give a complete proof of this theorem, but instead refers to personal communication with La Poutré, we have decided to reproduce his proof here, filling in the missing details.

A *barrier* in a graph $G = (V, E)$ is a set $S \subseteq V$ of vertices of G such that $n - 2\nu(G) = \mathrm{comp}_{odd}(G - S) - |S|$, where $\mathrm{comp}_{odd}(G - S)$ denotes the number of odd sized components of $G - S$. For example, the set $A(G)$ in the Gallai-Edmonds decomposition is a barrier.

Cheriyan's proof of Theorem 5.13 is based on the following characterization of $\sim_G$ (for a proof, see Lovász , Plummer [26]):

Theorem 5.14. *If G is elementary, then $P(G) = V/\sim_G$ is a set of all maximal barriers in G.*

Corollary 5.15. *Let G be elementary and let $P = \{S_1, S_2, \ldots, S_k\}$ be a partition of V. If all S_i are barriers in G, then P is a refinement of the canonical partition of G.*

Algorithm 21 Algorithm verifying if $\sim = \sim_G$

```
VERIFY(G,∼):
    if ∼ is not an equivalence relation then
        return "FAILURE"
    else
        let P := V/∼ = {S_1, S_2, ..., S_k}
        for i := 1 to k do
            { verify that S_i is a barrier }
            odd := 0
            forall st ∈ E such that s ∈ S_i and t ∉ S_i do
                DELETE(e)
                if not CONNECTED(s,t) and SIZE(t) is odd then
                    { a new odd sized component created }
                    odd := odd + 1
            if odd < |S_i| then
                return "FAILURE"
            add back all edges between S_i and V − S_i
        return "CORRECT"
```

Proof (of Theorem 5.13). Perform the Monte Carlo algorithm computing $\sim_G$ and let $\sim$ be the relation found. We need to verify the correctness of $\sim$, i.e. check if $\sim = \sim_G$.

Consider Algorithm 21. We first verify if $\sim_G$ is an equivalence relation (which it should be, by Theorem 3.25). Next we test if all of its equivalence classes S_i are barriers. If they are, then $\sim_G$ is computed correctly. This is because of Corollary 5.15 and the fact that we always have $\sim_G \subseteq \sim$ (we can get a false zero in $\tilde{A}(G)(s)^{-1}$, but not a false non-zero).

In order to efficiently verify that all S_i are barriers, we use the dynamic connectivity algorithm of Holm, de Lichtenberg and Thorup [16], introduced in Subsection 3.6.3. The correctness of the verification procedure is obvious. It works in time $\tilde{O}(m+n)$, because for every edge st of G, we have at most one call to each of DELETE(st), DELETE(ts), CONNECTED(s,t), CONNECTED(t,s), SIZE(s), SIZE(t), INSERT(st) and INSERT(ts), and each of these calls takes $\tilde{O}(1)$ time. □

Theorem 5.12 and 5.13 are the only known results concerning the computation of $\sim_G$ in $O(n^\omega)$ time Las Vegas. Whether this can be done for general graphs is a very interesting open problem.

Chapter 6

Open Problems

In this chapter we briefly discuss some open problems related to this work.

Las Vegas algorithm for the planar case. An open problem that seems to be an easy one is making the planar matching algorithm Las Vegas. As mentioned in Section 5.1 the method used in the general case can be adapted to the planar case. The only problem is that the basis of $\ker \tilde{A}(G)$ is not, in general, sparse (i.e. may have a lot of non-zero entries), so we need a way to efficiently extract the information about the non-zero entries of the basis vectors without actually constructing them.

Las Vegas computation of $\sim_G$. The relation $\sim_G$ can be computed in $O(n^\omega)$ time Las Vegas if G is elementary or bipartite (see Section 5.2). As a consequence, we get a Las Vegas matching verification algorithm for these two cases. It is interesting whether a Las Vegas algorithm is possible in the general case.

Derandomization. Another interesting problem is whether we can find a substitution $s : \tilde{E} \to \mathbb{F}_p$ such that $\det \tilde{A}(G)(s) \neq 0$, deterministically. This can of course be done by first finding a perfect matching using a deterministic algorithm, but this does not give an $O(n^\omega)$ matching algorithm. More importantly, we would like a solution based on linear algebra only. Geelen [15] has some ideas, but his algorithm is very inefficient.

Derandomization of our matching algorithms is interesting in itself, but it might also lead to an NC algorithm for the maximum matching problem. The RNC algorithm for this problem, due to Mulmuley, V. Vazirani and U. Vazirani [31] is based on the Rabin-Vazirani matching algorithm.

Algebraic matching algorithm. In Section 3.6 we gave an $O(n^\omega)$ maximum matching algorithm for general graphs, based on the algebraic approach. However, this algorithm is quite complicated and heavily relies on graph-theoretic results and techniques. It would be nice to have a strictly algebraic, and possibly simpler, matching algorithm for general graphs.

The last two problems should probably be called research topics rather then open problems.

Matrix inversion via perfect matchings. In this work, we have shown that by using the fast matrix multiplication algorithm of Coppersmith and Winograd, we can find maximum matchings in time $O(n^{2.38})$. For dense graphs, this is asymptotically faster than using any of the combinatorial algorithms. However, the large constant hidden in $O(n^{2.38})$ complexity renders our algorithm useless for practical purposes.

It would really be amazing if we could go in the opposite direction as well, i.e. use the combinatorial matching algorithms to multiply boolean matrices in time $O(n^{2.5})$, with a small constant (for information theoretic reasons, there is probably no hope in case of matrices over larger rings). It is easy to prove that all we need is an $O(n^{2.5})$ combinatorial algorithm for computing the $\sim_G$ relation defined in Subsection 3.6.2. Is that possible? In fact, any improvement over the $O(n^{2.81})$ complexity of the Strassen's matrix multiplication algorithm would be very interesting.

A substitute for nested dissection. Reading Chapter 4 might give an impression that although the nested dissection algorithm leads to an $O(n^{\omega/2})$ maximum matching algorithm, it is not really suited for this kind of purposes. This deficiency of the nested dissection is particularly clear in Subsection 4.3.2 where a submatrix of $\tilde{A}(G)^{-1}$ is computed.

One would expect a framework for planar matrix computations, in which elaborate considerations of Subsection 4.3.2 are not necessary.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Longman Publishing Co., Inc., 1974.

[2] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$. *Inf. Proc. Letters*, 37:237–240, 1991.

[3] C. Berge. Two theorems in graph theory. *Proc. Nat. Acad. Sci. USA*, pages 842–844, 1957.

[4] N. Blum. A new approach to maximum matching in general graphs. In *Proc. 17th ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 586–597. Springer-Verlag, 1990.

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

[6] J. Cheriyan. Randomized $\widetilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing*, 26(6):1635–1655, 1997.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing*, pages 1–6. ACM Press, 1987.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[9] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[10] J. Egerváry. Matrixok kombinatorius tulajdonságairól. *Matematikai és Fizikai Lapok*, 38:16–28, 1931.

[11] S. Even and Kariv. An $O(n^{2.5})$ algorithm for maximum matching in general graphs. In *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–112, 1975.

[12] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, 1995.

[13] C. Fremuth-Paeger and D. Jungnickel. Balanced network flows. VIII. A revised theory of phase-ordered algorithms and the $O(\log(n^2/m)/\log n)$ bound for the nonbipartite cardinality matching problem. *Networks*, 41(3):137–142, 2003.

[14] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, 1991.

[15] J. Geelen. An algebraic matching algorithm. *Combinatorica*, 20:61–70, 2000.

[16] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[17] J. Hopcroft and R. Karp. $N^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[18] O. H. Ibarra and S. Moran. Deterministic and probabilistic algorithms for maximum bipartite matching via fast matrix multiplication. *Inf. Process. Lett.*, 13(1):12–15, 1981.

[19] H. Karloff. A Las Vegas RNC algorithm for maximum matching. *Combinatorica*, 6:387–391, 1986.

[20] R. Karp, E. Upfal, and A. Widgerson. Constructing a perfect matching is in Random NC. *Combinatorica*, 6:35–48, 1986.

[21] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 27–37, 1994.

[22] D. König. Graphok és matrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

[23] R. J. Lipton, D. J. Rose, and R. Tarjan. Generalized nested dissection. *SIAM J. Num. Anal.*, 16:346–358, 1979.

[24] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Applied Math.*, pages 177–189, 1979.

[25] L. Lovász. On determinants, matchings and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 565–574. Akademie-Verlag, 1979.

[26] L. Lovász and M. Plummer. *Matching Theory*. Akadémiai Kiadó, 1986.

[27] S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*, pages 17–27, 1980.

[28] G. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 112–117, 1989.

[29] M. Mucha and P. Sankowski. Maximum matchings in planar graphs via Gaussian elimination. In *Proceedings of the 12th Annual European Symposium on Algorithms*, pages 532–543, 2004.

[30] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255, 2004.

[31] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing*, pages 345–354. ACM Press, 1987.

[32] V. Y. Pan and J. H. Reif. Fast and efficient parallel solution of sparse linear systems. *SIAM J. Comput.*, 22(6):1227–1250, 1993.

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

[34] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[35] W. T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, 22:107–111, 1947.

[36] J. Wein. Las Vegas RNC algorithms for unary weighted perfect matching and T-join problems. *Information Processing Letters*, 40:161–167, 1991.

[37] D. B. Wilson. Determinant algorithms for random planar structures. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 258–267. Society for Industrial and Applied Mathematics, 1997.

[38] V. P. Z. Galil. Improved processor bounds for combinatorial problems in RNC. *Combinatorica*, 8:189–200, 1988.

[39] R. Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226, Berlin, 1979. Springer-Verlag.

Index