University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

Algebraic Algorithm for Maximum Matching

Antonio Marcos Shiro Arnauts Hachisuca

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Marcel K. de Carli Silva

São Paulo
2024

Resumo

Antonio Marcos Shiro Arnauts Hachisuca. **Algoritmo algébrico para emparelhamento máximo**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2024.

O problema do emparelhamento máximo, que busca encontrar o maior conjunto possível de arestas não adjacentes em um grafo, é um desafio fundamental na teoria dos grafos que tem impulsionado inovações algorítmicas por décadas. Em 2009, Nicholas J. A. Harvey alcançou um avanço significativo ao desenvolver um algoritmo probabilístico que resolve o emparelhamento máximo em tempo $O(n^\omega)$ para grafos arbitrários, onde n é o número de vértices e ω é o expoente da multiplicação de matrizes. Seu algoritmo atinge este limite ao combinar de forma engenhosa conceitos da teoria algébrica dos grafos com uma estratégia de implementação eficiente que emprega atualizações "lazy" e técnicas de divisão e conquista. Este trabalho fornece uma análise abrangente e implementação do algoritmo de Harvey.

Palavras-chave: Grafos. Emparelhamentos. Algoritimos probabilísticos.

Abstract

Antonio Marcos Shiro Arnauts Hachisuca. **Algebraic Algorithm for Maximum Matching**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2024.

The Maximum Matching problem, which seeks to find the largest possible set of non-adjacent edges in a graph, is a fundamental challenge in graph theory that has driven algorithmic innovation for decades. In 2009, Nicholas J. A. Harvey made a significant breakthrough by developing a randomized algorithm that solves maximum matching in $O(n^\omega)$ time for arbitrary graphs, where n is the number of vertices and ω is the matrix multiplication exponent. His algorithm achieves this bound by ingeniously combining concepts from algebraic graph theory with an efficient implementation strategy that employs lazy updates and divide-and-conquer techniques. This work provides a comprehensive analysis and implementation of Harvey's algorithm.

Keywords: Graphs. Matchings. Probabilistic algorithms.

List of Figures

List of Programs

Contents

Introduction

The Maximum Matching problem is a fundamental problem in graph theory. Key milestones include:

- In 1965, Edmonds [4] introduced an $O(n^2m)$ algorithm that solves this problem using graph theory techniques;
- In 1989, Rabin and Vazirani [11] developed a probabilistic $O(n^{\omega+1})$ algorithm that solves the problem through algebraic methods;
- In 2009, Harvey [6] created a probabilistic $O(n^{\omega})$ algebraic algorithm using a divide-and-conquer approach with lazy updates.

These developments show how researchers have progressively improved algorithms to solve the Maximum Matching problem, reducing computational complexity over time.

The primary objective of this work is to implement the algorithm developed by Harvey [6]. We present a C++ implementation of the algorithm, available in the open-source repository at github/antoniomsah/algebraic-max-matching. The dissertation is structured as follows:

1. Chapter 1 covers the fundamental concepts necessary for understanding the subsequent work;
2. Chapter 2 explores Tutte matrices and algebraic methods for solving perfect matching problems;
3. Chapter 3 presents a detailed explanation of Harvey's algorithm;
4. Chapter 4 shows how to extend perfect matching algorithms to maximum matching scenarios.

Chapter 1

Preliminaries

The purpose of this chapter is to introduce key concepts related to the maximum matching algorithm. The chapter covers important topics such as the definition of graph maximum matching, the Sherman-Morrison-Woodbury formula, and skew-symmetric matrices. These concepts are fundamental for understanding the algorithm's correctness and time complexity.

1.1 Graph theory

Definition 1.1.1 (Graph). *A **graph** G is a pair (V, E) such that*

*(i) V is a finite set, whose elements are called **vertices**;*
*(ii) E is set of unordered pairs of vertices, whose elements are called **edges**;*

The vertex set of a graph G is denoted as V_G or $V(G)$. The edge set of a graph G is denoted as E_G or $E(G)$. If $\{u, v\} \in E(G)$, then u and v are the **ends** of e and e **incides** in both u and v. When the context is clear, $\{u, v\}$ may be abbreviated to uv.

Definition 1.1.2 (Matching). *For a graph $G := (V, E)$, a subset $M \subseteq E$ is a **matching** of G if no two edges in M share an end. A vertex $v \in V$ is M-covered if some edge of M incides in v, and it is said that M covers v; otherwise, v is M-exposed. A matching M is:*

- ***maximal**, if there is no edge $e \in E \setminus M$ such that $M \cup \{e\}$ is a matching of G;*
- ***maximum**, if for every matching M' of G one has $|M| \geq |M'|$;*
- ***perfect**, if $|V_G| = 2|M|$, i.e., every vertex of G is M-covered.*

*The **matching number** of G, denoted as $\nu(G)$, is the size of a maximum matching of G.*

Definition 1.1.3 (Essential edges). *Let $G := (V, E)$ be a graph that has a perfect matching. An edge $e \in E$ is **essential** if the graph $(V, E \setminus \{e\})$ does not have a perfect matching. An edge is **inessential** if it is not essential.*

Now, the following problem can be introduced.

MAXIMUM MATCHING

Given a graph G, find a maximum matching of G.

1.2 Linear algebra

Definition 1.2.1 (Submatrix)**.** *Let M be a matrix, we say that M' is a **submatrix** of M if we can obtain M' by removing zero or more rows and/or columns from M.*

Let M be a matrix. For any sets of indices R and C, we write $M_{R,C}$ or $M[R, C]$ to denote the submatrix of M formed by keeping only the rows indexed by R and columns indexed by C. Furthermore, we use $M[R, *]$ or $M_{R,*}$ to represent the submatrix containing all rows indexed by R and all columns of M (resp., $M[*, C]$).

Definition 1.2.2 (Schur complement)**.** *Let M be a square matrix of the form*

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

*where D is a square matrix. Then, if D is non-singular, the matrix $A - BD^{-1}C$ is the **Schur complement** of D in M. The Schur complement has the following property:*

$$\det(M) = \det(D)\det(A - BD^{-1}C). \tag{1.1}$$

Theorem 1.2.3 (Sherman-Morrison-Woodbury formula)**.** *Suppose that M is non-singular. Then*

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

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

Proof. For (1), let

$$A := \begin{bmatrix} I & V^T \\ -U & M \end{bmatrix}.$$

Note that the Schur complement of the block M of the matrix A is $I + V^T M^{-1} U$ and that

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

Then, by (1.1), one has

$$\begin{aligned} \det(M)\det(I + V^T M^{-1} U) &= \det(A) \\ &= \det\left(\begin{bmatrix} I & 0 \\ -U & I \end{bmatrix} \begin{bmatrix} I & V^T \\ 0 & M + UV^T \end{bmatrix} \right) \\ &= \det(I)\det(M + UV^T) = \det(M + UV^T). \end{aligned}$$

Thus, (1) is proven. For (2), it suffices to verify that

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

Let $A := (I + V^T M^{-1}U)$, then

$$\begin{aligned}
&(M + UV^T)(M^{-1} - M^{-1}U(I + V^T M^{-1}U)^{-1}V^T M^{-1}) \\
&\quad = (M + UV^T)(M^{-1} - M^{-1}UA^{-1}V^T M^{-1}) \\
&\quad = (I - UA^{-1}V^T M^{-1}) + (UV^T M^{-1} - UV^T M^{-1}UA^{-1}V^T M^{-1}) \\
&\quad = (I + UV^T M^{-1}) - (UA^{-1}V^T M^{-1} + UV^T M^{-1}UA^{-1}V^T M^{-1}) \\
&\quad = (I + UV^T M^{-1}) - U(I + V^T M^{-1}U)(A^{-1}V^T M^{-1}) \\
&\quad = (I + UV^T M^{-1}) - UAA^{-1}V^T M^{-1} \\
&\quad = I + UV^T M^{-1} - UV^T M^{-1} = I.
\end{aligned}$$

□

Corollary 1.2.4 (Harvey [6, Corollary 2.1])**.** *Let M be a non-singular square matrix, let $N := M^{-1}$ and let S be a subset of rows from M. Let $\tilde{M}$ be a matrix that which is identical to M except that $\tilde{M}_{S,S} \neq M_{S,S}$ and $\Delta := \tilde{M}_{S,S} - M_{S,S}$. If $\tilde{M}$ is non-singular, then*

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

Proof. Let $k := |S|$. Let U[1] be the $n \times k$ matrix that selects the S columns from an $n \times n$ matrix, i.e. for an $n \times n$ matrix M, one has $MU = M_{*,S}$. Let $V^T := \Delta U^T$. Then, by Theorem 1.2.3,

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

□

Definition 1.2.5 (Skew-symmetric matrix)**.** *A matrix M is **skew-symmetric** if $M = -M^T$.*

Fact 1.2.6 (Inverse of a skew-symmetric matrix)**.** *Let M be a skew-symmetric matrix. If M is non-singular, then M^{-1} is also skew-symmetric.*

1.3 Matrix Algorithms

Definition 1.3.1 (Matrix multiplication exponent)**.** *The **matrix exponent multiplication exponent**, denoted by ω, is the infimum of the exponent over all matrix multiplication algorithms. That is there is a known algorithm that solves matrix multiplication in $O(n^{\omega+o(1)})$ field operations.*

[1] Every column should be a canonical basis vector from S.

Here are some milestones related to the matrix multiplication exponent throughout the years.

Year	Author(s)	Bound on omega
1969	Strassen [12]	2.8074
1990	Coppersmith and Winograd [2]	2.3755
2024	Alman et al. [1]	2.371339

The following algorithms can be done in $O(n^\omega)$:

1. **Matrix inversion**, see CLRS [13, Theorem 28.2];
2. **Matrix rank**, see [7].

Chapter 2

Perfect matchings

The initial section introduces fundamental algebraic techniques, focusing on Tutte matrices and their probabilistic representation. Then, a simple algorithm is presented, demonstrating an algebraic strategy for matching problems. The chapter ends with an enhanced version of the simple algorithm that uses rank-two matrix update techniques.

2.1 Tutte Matrix

Definition 2.1.1 (Indeterminates)**.** *An **indeterminate** is a variable that is not assigned a specific value. It represents a symbolic placeholder that never assumes a value.*

Definition 2.1.2 (Tutte Matrix)**.** *Let G be a graph and let $n := |V_G|$. For each edge $uv \in E_G$ associate an indeterminate t_{uv}. The Tutte matrix is the $n \times n$ skew-symmetric matrix such that $T_{uv} = \pm t_{uv}$ if $uv \in E_G$ and 0, otherwise. The sign is chosen such that T is skew-symmetric. The Tutte matrix of G is denoted as T_G.*

For a Tutte matrix T, its Pfaffian, denoted $\mathrm{Pf}(T)$, is a polynomial whose complete mathematical definition involves some algebraic concepts beyond our current scope. However, two properties of the Pfaffian are crucial:

1. The Pfaffian $\mathrm{Pf}(T)$ has a monomial for every perfect matching in the underlying graph of T;
2. There exists a fundamental relationship between the Pfaffian and the determinant of a Tutte matrix: $\det(T) = \mathrm{Pf}(T)^2$.

For a comprehensive treatment of this polynomial and its properties, we refer the reader to Godsil [5, Chapter 7].

Fact 2.1.3 (Tutte [14])**.** *A graph G has a perfect matching iff T_G is non-singular.*

Proof. Direct from the property $\det(T) = \mathrm{Pf}(T)^2$. □

While this property of Tutte matrices is powerful, it presents a computational challenge for algorithmic applications. The issue stems from the relationship $\det(T) = \mathrm{Pf}(T)^2$, and

that $\mathrm{Pf}(T)$ contains a monomial for each perfect matching in graph G. Since a graph may contain exponentially many perfect matchings, symbolic computation becomes infeasible.

2.1.1 Probabilistic representation of a Tutte Matrix

Fortunately, a probabilistic solution was proposed by Lovász [8]:

- Replace the non-zero entries of T with random values from a sufficiently large field.

This provides a computationally feasible approach.

Lemma 2.1.4 (Schwartz-Zippel)**.** *Let $\mathbb{F}$ be a field, and let $p(x_1, x_2, \ldots, x_n)$ be a non-zero polynomial in $\mathbb{F}[x_1, x_2, \ldots, x_n]$ of total degree d. Suppose $S \subseteq \mathbb{F}$ is a finite subset of the field $\mathbb{F}$. If the variables $x_1, x_2, \ldots, x_n$ are chosen independently and uniformly at random from S, then*

$$\Pr\left[p(x_1, x_2, \ldots, x_n) = 0\right] \leq \frac{d}{|S|}.$$

Proof. See Motwani and Raghavan [10, Theorem 7.2]. □

According to Lemma 2.1.4, selecting a sufficiently large field significantly reduces the probability of failure. Hence, the rank of a Tutte matrix with random values is preserved with **high** probability.

Selecting a sufficiently large field

Constructing such a field is straightforward. For any prime number p, the set of integers modulo p, denoted as $\mathbb{Z}_p$, forms a field of size p.

Implementation Overview

Let G be a graph. The following assumption is used for the algorithm pertaining Tutte matrices:

- For each edge $uv \in E(G)$, the matrix entry t_{uv} (since Tutte matrices are skew-symmetric, t_{vu} is also fixed) is randomly selected from the finite field $\mathbb{Z}_p$. These values are not altered throughout the subsequent algorithmic procedures and can be accessed in $O(1)$ time complexity.

In the pseudocode, the following functions related to Tutte matrices are used:

1. TutteMatrix(G): Builds a Tutte matrix of graph G with random values;
2. RemoveEdge(T, uv): Assigns $T_{uv} \leftarrow 0$ and $T_{vu} \leftarrow 0$;
3. $E(T)$: Returns every edge $uv \in E(G)$ such that $T_{uv} \neq 0$, i.e. the edges that were not removed from G.

2.2 Naive algorithm

Now, a naive algorithm can be implemented. Let $G := (V, E)$. The idea is, for each edge $e \in E$, check using Fact 2.1.3 if the graph $(V, E \setminus \{e\})$ has a perfect matching.

1. If it does not have a perfect matching, then e is essential;
2. Else, let $G = (V, E \setminus \{e\})$.

Then, after iterating through each edge in E, only the essential edges remain in G, i.e. a perfect matching. Thus, we have the following algorithm.

Program 2.1 NaiveAlgorithm

```
1   function NaiveAlgorithm(G)
2       T ← TutteMatrix(G)
3       if T is singular then   ▷ G does not have a perfect matching.
4         return ∅
5       for each uv ∈ E(G) do
6           t ← T_uv   ▷ Save the matrix entry before removing it.
7           RemoveEdge(T, uv)
8           if T is singular then   ▷ By Fact 2.1.3, edge uv is essential.
9             T_uv ← t   ▷ Reverse the change.
10            T_vu ← −t
11      return E(T)
```

Time complexity

Let $f(n)$ be the running time of NaiveAlgorithm when $|V_G| =: n$.

- **Line 2**: Since a graph has at most $\binom{n}{2}$, there are $O(n^2)$ iterations;
- **Line 4**: Checking if a matrix is singular can be done in $O(n^\omega)$ (Section 1.3).

Thus, the running time can be expressed as:

$$f(n) = O(n^2)O(n^\omega) = O(n^{\omega+2}).$$

2.3 Rank-two update algorithm

The bottleneck of the previous algorithm is the necessity to recompute the whole matrix inverse after each iteration. An improvement from the previous algorithm was achieved by using Corollary 2.3.2 to quickly check essential edges and Theorem 2.3.1 to update the inverse in $O(n^2)$ time rather than $O(n^\omega)$.

Theorem 2.3.1 (Rank-two update)**.** *Let G be a graph. Let $T := T_G$. Let $N := T^{-1}$. Let $S \subseteq \{u, v\}$ such that $u, v \in V(G)$ and $T_{S,S} \neq 0$. Let $\tilde{T}$ be a matrix which is identical to T except*

that $\tilde{T}_{S,S} = 0$. *Let* $\Delta := \tilde{T} - T$. *If* $\tilde{T}$ *is non-singular, then*

$$\tilde{T}^{-1} = N + N_{*,S} \cdot \begin{pmatrix} 1/(1 + T_{u,v}N_{u,v}) & 0 \\ 0 & 1/(1 + T_{v,u}N_{v,u}) \end{pmatrix} \cdot \Delta \cdot N_{S,*}.$$

Proof. By Corollary 1.2.4, it suffices to prove that

$$(I + \Delta N_{S,S})^{-1} = \begin{pmatrix} 1/(1 + T_{u,v}N_{u,v}) & 0 \\ 0 & 1/(1 + T_{v,u}N_{v,u}) \end{pmatrix}.$$

Then, one has

$$\begin{aligned} (I + \Delta N_{S,S}) &= I + \begin{pmatrix} 0 & -T_{u,v} \\ -T_{v,u} & 0 \end{pmatrix} \begin{pmatrix} 0 & N_{u,v} \\ N_{v,u} & 0 \end{pmatrix} \\ &= I + \begin{pmatrix} -T_{v,u}N_{u,v} & 0 \\ 0 & -T_{u,v}N_{v,u} \end{pmatrix} \\ &= I + \begin{pmatrix} T_{u,v}N_{u,v} & 0 \\ 0 & T_{v,u}N_{v,u} \end{pmatrix} && T \text{ and } N \text{ are skew-symmetric} \\ &= \begin{pmatrix} 1 + T_{u,v}N_{u,v} & 0 \\ 0 & 1 + T_{v,u}N_{v,u} \end{pmatrix}. \end{aligned}$$

Finally,

$$\begin{aligned} (I + \Delta N_{S,S})^{-1} &= \begin{pmatrix} 1 + T_{u,v}N_{u,v} & 0 \\ 0 & 1 + T_{v,u}N_{v,u} \end{pmatrix}^{-1} \\ &= \begin{pmatrix} 1/(1 + T_{u,v}N_{u,v}) & 0 \\ 0 & 1/(1 + T_{v,u}N_{v,u}) \end{pmatrix}. \end{aligned} \tag{2.1}$$

Thus, it is proven. □

Corollary 2.3.2 (Edge removal condition)**.** *Let* G *be a graph,* T *be the Tutte matrix of* G *and* $N := T^{-1}$. *An edge* $ij \in E(G)$ *is essential if and only if* $N_{i,j} = -1/T_{i,j}$.

Proof. Direct from Equation (2.1). □

The idea is similar to the simple algorithm. Let $G := (V, E)$ be a graph. For each $e \in E$, check if e is essential. However, rather than temporarily removing e from G, we use Corollary 2.3.2 to quickly decide if e is essential. If e is inessential, then we use a rank-two update instead of completely recomputing the inverse. Thus, we have the following algorithm.

Program 2.2 RANK-TWO UPDATE ALGORITHM

```
1  function RankTwoAlgorithm(G)
2      T ← TutteMatrix(G)
3      N ← T^{-1}
4      for each uv ∈ E(G) do
```

cont ⟶

```
    ⟶ cont
5           if N_{u,v} ≠ −1/T_{u,v} then   ▷ By Corollary 2.3.2, this edge is inessential.
6             S ← {u, v}
7             N ← RankTwoUpdate(S, T, N)   ▷ Theorem 2.3.1.
8             RemoveEdge(T, uv)
9       return E(T)
```

Time complexity

First, let $t(n)$ be the running time of RankTwoUpdate when $|V_G| =: n$. Note that

$$\begin{pmatrix} 1/(1+T_{u,v}N_{u,v}) & 0 \\ 0 & 1/(1+T_{v,u}N_{v,u}) \end{pmatrix}$$

is a 2×2 matrix. Consequently,

$$t(n) = O(2n^2) = O(n^2).$$

Then, let $g(n)$ be the running time of RankTwoAlgorithm when $n =: |V_G|$. Since a rank-two update is performed for each removed edge and it is necessary to compute the matrix inverse, we have the following running time.

$$g(n) = mt(n) + O(n^\omega) = mO(n^2) + O(n^\omega) = O(n^2 m) + O(n^\omega).$$

Chapter 3

Harvey's algorithm

This chapter presents the probabilistic algorithm proposed by Harvey [6] that finds a perfect matching in general graphs with time complexity $O(n^\omega)$.

3.1 Algorithm

The main bottleneck in the previous algorithm was the need to update the entire inverse matrix at each step. Harvey's algorithm addresses this limitation by employing a divide-and-conquer strategy combined with lazy updates. After each recursive step, only the necessary portions of the inverse matrix are updated. As a result, Harvey's algorithm has a time complexity of $O(n^\omega)$.

For a graph G and $n = |V(G)|$, the algorithm maintains two matrices, T and N, that are initialized as

1. $T :=$ a Tutte matrix where the entries were randomly chosen, see Section 2.1.1;
2. $N := T^{-1}$.

It relies on two recursive functions: DeleteEdgesCrossing and DeleteEdgesWithin.

3.1.1 DivideInTwo

DivideInTwo(A) is a function that divides a set A in two parts, R and S, such that $R \cup S = A$, $R \cap S = \emptyset$ and $|R| - |S| \leq 1$. This function has time complexity $O(n)$ and can be implemented through integer indexing the set.

3.1.2 DeleteEdgesCrossing

DeleteEdgesCrossing(R, S): receives two disjoint sets of vertices R and S and deletes inessential edges with an end in R and the other in S. The following invariant must be preserved:

- DeleteEdgesCrossing(R, S): initially has $N[R \cup S, R \cup S] = T^{-1}[R \cup S, R \cup S]$ and this property is restored after each call of DeleteEdgesCrossing(R_i, S_j).

To maintain this invariant the following updates are done.

Theorem 3.1.1 (Update 1[1]). *Let R, S be two disjoint sets of vertices such that $|R| = |S| = 1$. Let $N := T^{-1}$, $r \in R$ and $s \in S$. If $\{r, s\}$ is inessential, let $\tilde{T}$ be the Tutte matrix of G without edge rs, then one has*

$$\tilde{T}^{-1}_{r,s} = N_{r,s}/(1 + T_{r,s}N_{r,s})$$

and

$$\tilde{T}^{-1}_{s,r} = N_{s,r}/(1 + T_{r,s}N_{r,s}) = -\tilde{T}^{-1}_{r,s}.$$

Proof. Let $RS := R \cup S$ and let $\Delta := \tilde{T}_{RS,RS} - T_{RS,RS}$. By Corollary 1.2.4, one has:

$$\begin{aligned}
\tilde{T}^{-1}_{RS,RS} &= N_{RS,RS} - N_{RS,RS}(I + \Delta N_{RS,RS})^{-1}\Delta N_{RS,RS} \\
&= N_{RS,RS} - N_{RS,RS}\begin{bmatrix} 1 + T_{r,s}N_{r,s} & 0 \\ 0 & 1 + T_{r,s}N_{r,s} \end{bmatrix}^{-1} \Delta N_{RS,RS} && \text{by (2.1)} \\
&= N_{RS,RS} - N_{RS,RS}\begin{bmatrix} 1/(1 + T_{r,s}N_{r,s}) & 0 \\ 0 & 1/(1 + T_{r,s}N_{r,s}) \end{bmatrix} \Delta N_{RS,RS} \\
&= N_{RS,RS} + \begin{bmatrix} 0 & N_{r,s}T_{s,r}N_{r,s}/(1 + T_{r,s}N_{r,s}) \\ N_{s,r}T_{r,s}N_{s,r}/(1 + T_{r,s}N_{r,s}) & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & N_{r,s}/(1 + T_{r,s}N_{r,s}) \\ N_{s,r}/(1 + T_{r,s}N_{r,s}) & 0 \end{bmatrix}.
\end{aligned}$$

□

Theorem 3.1.2 (Update 2). *Let R, S be two disjoint set of vertices. Let $\tilde{T}$ be T after removing some (possibly zero) edges from G with an end in R_i and another in S_j. Let $N := T^{-1}$ and $\Delta := \tilde{T} - T$. Then*

$$\tilde{T}^{-1}[R \cup S, R \cup S] = N_{R\cup S,R\cup S} - N_{R\cup S,R_i\cup S_j}(I + \Delta N_{R_i\cup S_j,R_i\cup S_j})^{-1}\Delta N_{R_i\cup S_j,R\cup S}.$$

Proof. Direct from Corollary 1.2.4. Update the whole matrix with 1.2.4 and select only the desired submatrix. □

We have the following algorithm.

[1] This update is different from the one in Harvey [6].

Program 3.1 Harvey's algorithm: DeleteEdgesCrossing

```
1   function DeleteEdgesCrossing(R, S)  ▷ R and S are disjoint sets of vertices.
2       if |R| = 0 or |S| = 0 then return  ▷ There are no edges.
3
4       if |R| = 1 and |S| = 1 then  ▷ There is at most one edge.
5           Let r in R
6           Let s in S
7           if T_{r,s} ≠ 0 and N_{r,s} ≠ −1/T_{r,s} then  ▷ Corollary 2.3.2.
8               N_{r,s} ← N_{r,s}/(1 + T_{r,s}N_{r,s})  ▷ Theorem 3.1.1.
9               N_{s,r} ← −N_{r,s}
10              RemoveEdge(T, rs)
11          return
12
13      RS ← R ∪ S
14      R_1, R_2 ← DivideInTwo(R)
15      S_1, S_2 ← DivideInTwo(S)
16      for i in {1, 2} do
17          for j in {1, 2} do
18              T′, N′ ← T, N  ▷ Save current T and N states
19              DeleteEdgesCrossing(R_i, S_j)
20              Δ ← T[R_i ∪ S_j, R_i ∪ S_j] − T′[R_i ∪ S_j, R_i ∪ S_j]
21              N_{RS,RS} ← N′_{RS,RS} − N′_{RS,R_i∪S_j}(I + ΔN′_{R_i∪S_j,R_i∪S_j})^{−1}ΔN′_{R_i∪S_j,RS}  ▷ Theorem 3.1.2.
```

Time complexity

Let $f(r, s)$ be the running time for DeleteEdgesCrossing(R, S) when $|R| = r$ and $|S| = s$. Let $n = r + s$. The base cases are $O(1)$. Otherwise, a line-by-line analysis:

1. **Dividing in half (Lines 14 and 15)**: Takes $O(n)$ time;
2. **Saving the states (Line 18)**: Takes $O(n^2)$ time;
3. **Recursive call (Line 19)**: Recurrence is $f(r/2, s/2)$;
4. **Delta (Line 20)**: Matrix subtraction is $O(n^2)$;
5. **Update submatrix (Line 21)**: Takes $O(n^\omega)$.

Combining these steps, we have

$$\begin{aligned} f(r, s) &= O(1) + O(1) + O(n^2) + 4(O(n^2) + f(r/2, s/2) + O(n^\omega)) \\ &= 4O(n^2) + 4f(r/2, s/2) + 4O(n^\omega) \\ &= 4f(r/2, s/2) + 4O(n^\omega). \end{aligned}$$

Now, applying the master theorem [13, Theorem 4.1] for recursions, i.e.

$$T(n) = aT(n/b) + f(n).$$

In this case, $T(n) = t(r, s)$, $a = 4$, $b = 2$ and $f(n) = O(n^\omega)$, then if $\omega > \log_b a = 2$ the complexity is dominated by $O(n^\omega)$. From Section 1.3, the best currently known matrix

multiplication algorithm has $\omega > 2$. Thus, the complexity is dominated by $O(n^\omega)$ and

$$f(r,s) = 4f(r/2, s/2) + 4O(n^\omega) = 8O(n^\omega) = O(n^\omega). \tag{3.1}$$

3.1.3 DeleteEdgesWithin

DeleteEdgesWithin: receives a set of vertices S and deletes inessential edges that have both ends in S. The following invariant must be preserved:

- DeleteEdgesWithin(S): initially has $N[S,S] = T^{-1}[S,S]$ and this property is restored after each call of DeleteEdgesWithin and DeleteEdgesCrossing.

To maintain this invariant the following update is done.

Theorem 3.1.3 (Update 3)**.** *Let $S \subseteq V_G$, let $\tilde{T}$ be T after removing some (possibly zero) edges from G with both ends in S. Then, let $N := T^{-1}$ and $\Delta := \tilde{T} - T$, one has*

$$\tilde{T}^{-1}_{S,S} = N_{S,S} - N_{S,S_i}(I + \Delta N_{S_i,S_i})^{-1}\Delta N_{S_i,S}.$$

Proof. Direct from Corollary 1.2.4. Update the whole matrix using 1.2.4 and select only the desired submatrix. □

We have the following algorithm.

Program 3.2 Harvey's algorithm: DeleteEdgesWithin

```
1   function DeleteEdgesWithin(S)
2       if |S| = 1 return
3
4       S_1, S_2 ← DivideInTwo(S, 2)
5       for i in {1, 2} do
6           T', N' ← T, N   ▷ Save current T and N states.
7           DeleteEdgesWithin(S_i)
8           Δ ← T_{S_i,S_i} − T'_{S_i,S_i}
9           N_{S,S} ← N' − N'_{S,S_i}(I + ΔN'_{S_i,S_i})^{-1} ΔN'_{S_i,S}   ▷ Theorem 3.1.3.
10      DeleteEdgesCrossing(S_1, S_2)
```

Time complexity

Let $g(n)$ be the running time of DeleteEdgesWithin(S) when $|S| = n$. The base case is direct. Then, through a line-by-line analysis we have

1. **DivideInTwo (Line 4)**: Takes $O(n)$ times;
2. **Saving the states (Line 6)**: Takes $O(n^2)$ time;
3. **Recursive call (Line 7)**: Takes $g(n/2)$ time;
4. **Delta (Line 8)**: Takes $O(n^2)$;
5. **Update 3 (Line 9)**: Takes $O(n^\omega)$ time;
6. **DeleteEdgesCrossing (Line 10)**: Takes $f(n/2, n/2)$.

Now, combining these steps:

$$\begin{aligned} g(n) &= O(n) + 2(2O(n^2) + g(n/2) + O(n^\omega)) + f(n/2, n/2) \\ &= O(n) + 2(2O(n^2) + g(n/2) + O(n^\omega)) + O(n^\omega) && \text{by (3.1)} \\ &= 2g(n/2) + 3O(n^\omega). \end{aligned}$$

Similarly to (3.1), we have

$$g(n) = 2g(n/2) + 3O(n^\omega) = O(n^\omega). \tag{3.2}$$

3.1.4 PerfectMatching

PerfectMatching: Receives a graph G and finds a perfect matching, or returns $\emptyset$ if one does not exist. It creates a Tutte Matrix of G with random entries. Note that calling DeleteEdgesWithin($V(G)$) is equivalent to deleting every inessential edge from G. Thus, after this call, the graph only has essential edges, i.e. edges from the perfect matching. Consequently, there is the following implementation:

Program 3.3 Harvey's algorithm: Perfect Matching

```
1  function PerfectMatching(G)
2      T ← TutteMatrix(G)
3      if T is singular then return ∅   ▷ The graph has no perfect matching.
4      N ← T^{-1}
5      DeleteEdgesWithin(V)
6      return E(T)
```

Time complexity

Let $T(n)$ be the running time of PerfectMatching(G) when $|V| = n$. A line-by-line comparison:

1. **Building a TutteMatrix (Line 2)**: Takes $O(n^2)$ time;
2. **Checking if a matrix is singular (Line 3)**: Takes $O(n^\omega)$ time;
3. **Calculating a matrix inverse (Line 4)**: Takes $O(n^\omega)$ time;
4. **DeleteEdgesWithin (Line 5)**: From (3.2), takes $g(n) = O(n^\omega)$ time;
5. **Returning the matching (Line 6)**: Takes $O(n^2)$ time.

Combining these steps, we have:

$$T(n) = O(n^2) + O(n^\omega) + O(n^\omega) + O(n^\omega) + O(n^2) = O(n^\omega). \tag{3.3}$$

Probability of failure

Let δ be the probability of failure. From Lemma 2.1.4, deciding if an edge can be deleted fails with probability n/q where q is the size of the field. Since there are $\binom{n}{2}$ edges, then

$$\delta \leq \binom{n}{2} n/q < n^3/q.$$

Correctness

To prove the algorithm's correctness, it suffices that for every edge $uv \in E(G)$, when evaluating whether uv can be removed (using Corollary 2.3.2), one has $N_{uv} = T_{uv}^{-1}$. This equality is guaranteed by the invariants. Since the updates ensure that the invariants are maintained throughout the algorithm, we can conclude that the edge removal decisions are correct.

3.2 Experimental Analysis

This section compares Harvey's algorithm with two perfect matching algorithms:

1. The rank-two algorithm described in Section 2.3;
2. An implementation of the Edmonds-Blossom algorithm made by Delfino [3].

The naive algorithm was not compared due to its poor performance. For reference, its performance is at least 100 times slower than the Rank-two algorithm in all cases.

3.2.1 Methodology

The implementations were evaluated using randomly generated graph instances created with the Erdős-Rényi model. For each vertex count, 10 test graphs were generated, varying the probability of edge creation between each pair of vertices. Each test case number corresponded to a specific probability of edge creation between vertex pairs, as defined in Table 3.1. A verification system was implemented to validate the correctness of all outputs.

Test number	Probability
1	0.10
2	0.20
3	0.25
4	0.50
5	0.50
6	0.50
7	0.75
8	0.75
9	0.75
10	1.00

Table 3.1: *Edge creation probability for each Perfect Matching test case number.*

Each of the 50 test cases was executed 50 times to account for machine performance variability, resulting in 2,500 total executions.

Hardware specifications

The benchmark was executed in a computer with the following specifications:

CPU	Intel(R) Core(TM) i7-9750H
RAM	32GB

Algorithms tested

The fast matrix multiplication algorithm was **not** implemented since its threshold is too high for application purposes. Instead the trivial $O(n^3)$ algorithm was used, thus the tested algorithms have the following time complexities, where n is the number of vertices and m is the number of edges.

Algorithm	Time complexity
Rank-two algorithm	$O(n^2m + n^3)$
Harvey's algorithm	$O(n^3)$
Edmonds-Blossom algorithm	$O(n^2m)$

3.2.2 Results

In Figure 3.1 provides an overview of the algorithms performance throughout all test cases. Even though the Edmonds-Blossom worst-case time complexity is theoretically worse than Harvey's algorithm, when $m > n$ (which is true for all test cases with $n = 200$), it demonstrates superior performance in all cases. Nonetheless, Harvey's algorithm outperforms the Rank-two algorithm for bigger graphs demonstrating a considerable improvement.

A more detailed case by case analysis follows. In Figure 3.2, the Rank-two algorithm outperforms Harvey's algorithm with small n, likely due to its lower constant factor.

However, as the test sizes increase, Harvey's algorithm shows progressive improvement relative to the Rank-two algorithm. This trend becomes evident in Figure 3.3, where the performance gap narrows significantly; And, for the complete graph (e.g. the last test case), the Rank-two algorithm is outperformed by Harvey's algorithm.

In Figure 3.4, Harvey's algorithm ultimately achieves better overall performance than the Rank-two algorithm. Notably the Rank-two algorithm exhibits significant performance variability, which can be attributed to its update frequency being directly tied to the number of edges removed from the graph. In contrast, Harvey's algorithm maintains consistent performance by executing a fixed number of updates, independent of the edge count. This behavior makes Harvey's algorithm more predictable in terms of execution time, particularly for larger graphs.

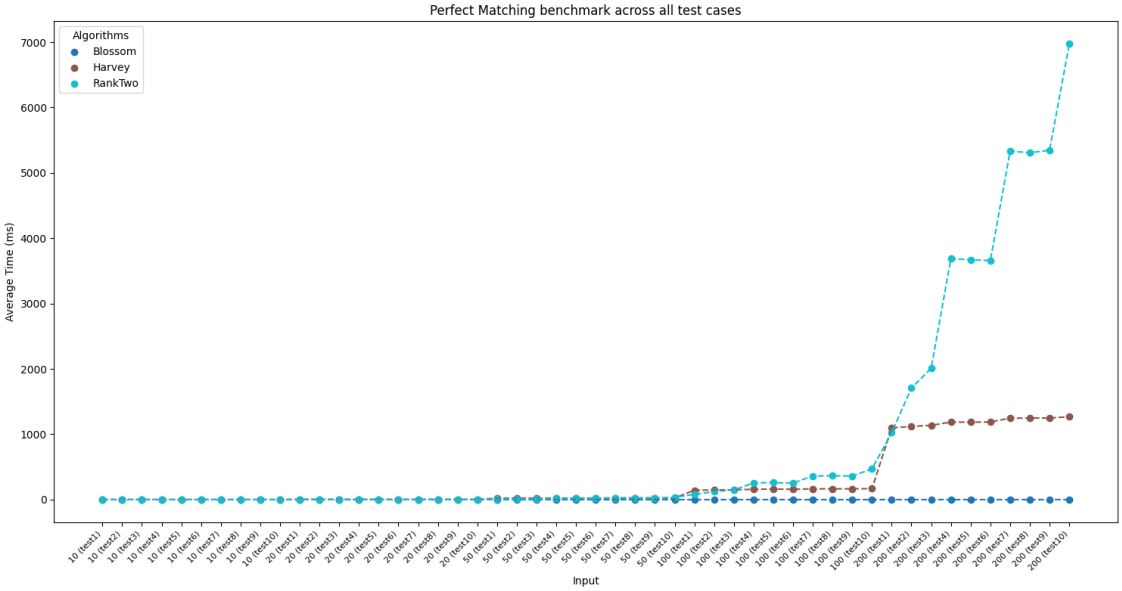

Figure 3.1: *Perfect Matching benchmark with all test cases where the input number represents the number of vertices.*

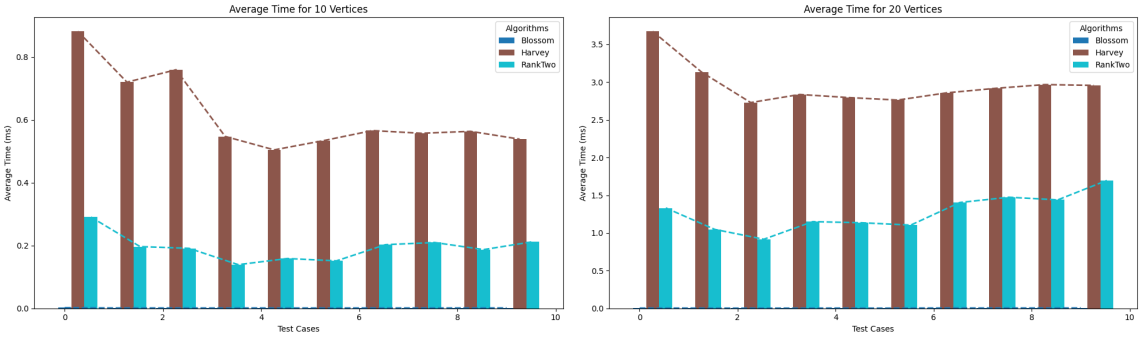

Figure 3.2: *Perfect Matching benchmark with 10 and 20 vertices.*

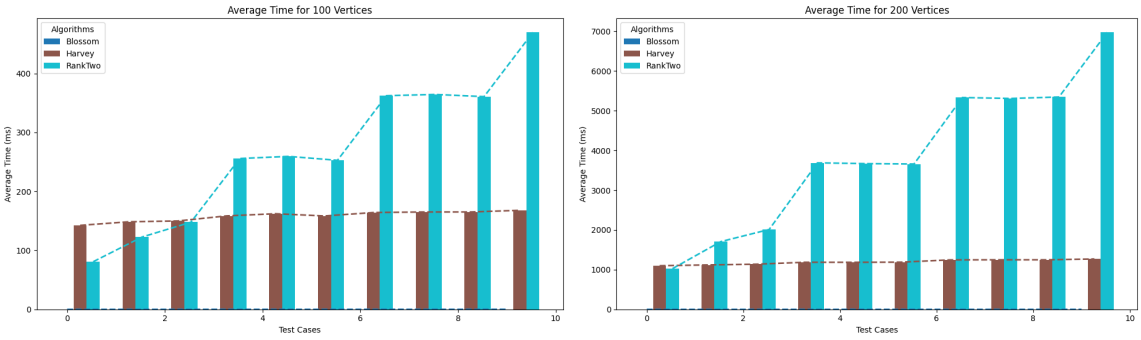

Figure 3.4: *Perfect Matching benchmark with 100 vertices and 200 vertices.*

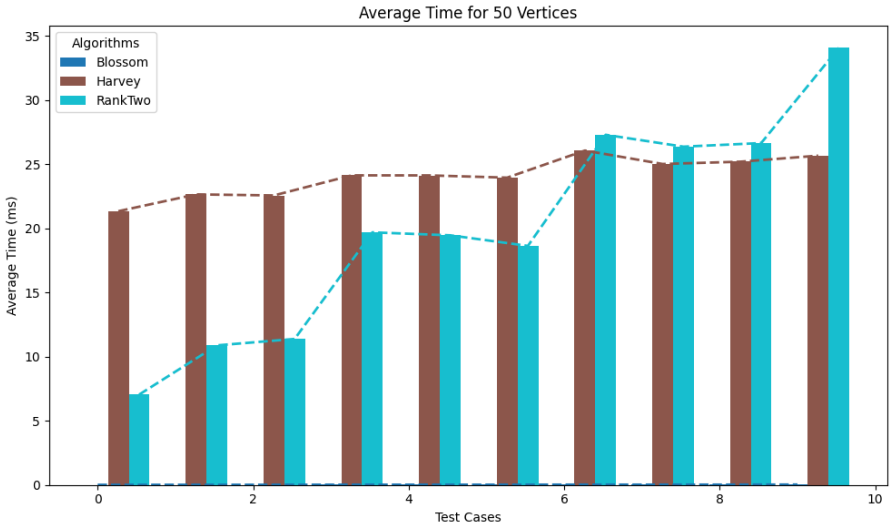

Figure 3.3: *Perfect Matching benchmark with 50 vertices.*

Chapter 4

Extension to Maximum Matching

This chapter shows how to extend the perfect matching algorithms to solve the maximum matching problem.

4.1 Maximum Matching algorithm

The extension to maximum matching is based on the following theorem.

Theorem 4.1.1 (Lovász and Plummer [9]). *Let G be a graph and let T be the Tutte Matrix of G. Then,* $\operatorname{rank}(T_G) = 2\nu(G)$.

Proof. See Rabin and Vazirani [11, p. 560]. □

According to Theorem 4.1.1, the number of unmatched vertices can be directly computed as:

$$|V(G)| - \operatorname{rank}(T).$$

To address these unmatched vertices, we can construct an augmented graph by introducing new vertices connected to all existing vertices. Specifically, for each unmatched vertex in the original graph, we add a new vertex with connections to every vertex in the original graph. This transformation ensures that every previously unmatched vertex now has an adjacent vertex, thus ensuring the existence of a perfect matching.

Program 4.1 Maximum Matching algorithm

```
1   function MaximumMatching(G)
2       T ← TutteMatrix(G)   ▷ Tutte matrix of G with random values.
3       G′ ← G
4       for i ← 0; i < |V(G)| − rank(G); i ← i + 1
5           v ← new vertex
6           V(G′) ← V(G′) ∪ {v}
7           for u ∈ V(G) do
8               E(G′) ← E(G′) ∪ {uv}
9       M′ ← PerfectMatching(G′)
10      M ← ∅
11      for uv ∈ M′ do
12          if u ∈ E(G) and v ∈ V(G) then   ▷ If this edge exists in the original graph.
13              M ← M ∪ {uv}
14      return M
```

Time complexity

Let $t(G)$ be the total running time of MaximumMatching(G) and $f(G)$ be the running time of PerfectMatching(G). Then, we have

- **Augmented Graph creation (Lines 4 to 10)**: The algorithm identifies unmatched vertices and adds corresponding new vertices. Since there are at most n unmatched vertices and each new vertex connects to n vertices, this phase requires $O(n^2)$ time;
- **Perfect Matching in the augmented graph (Line 11)**: The augmented graph has at most $2n$ vertices. By Equation (3.3), this step has a time complexity $f(G') = O((2n)^\omega) = O(2^\omega n^\omega) = O(n^\omega)$;
- **Maximum Matching recovery (Lines 13 to 17)**: Checking whether a vertex belongs to the original graph can be implemented in $O(1)$ time by using integer indexing. The overall complexity for this verification across all vertices is $O(n)$.

Combining these steps, the total time complexity is:

$$t(G) = O(n^2) + f(G') + O(n^2) = O(n^2) + O(n^\omega) + O(n^2) = O(n^\omega). \tag{4.1}$$

4.2 Experimental Analysis

This section compares Harvey's algorithm with two maximum matching algorithms:

1. The rank-two algorithm described in Section 2.3;
2. A Edmonds-Blossom algorithm implementation made by Delfino [3].

The naive algorithm was not compared due to its poor performance. As a reference, its performance is at least 100 times slower than the Rank-two algorithm in all cases.

4.2.1 Methodology

The implementations were evaluated using randomly generated graph instances created with the Erdös-Rényi model. To establish upper bounds on the matching number, the generated graphs were constrained to include odd components. For example, if the desired matching number was k, then at least $n/2 - k$ odd components were contained in the generated graph. For each vertex count, 10 test graphs were generated, varying the probability of edge creation between each pair of vertices. Each test case number corresponded to a specific probability of edge creation between vertex pairs, as defined in the Table 4.1. A verification system was implemented to validate the correctness of all outputs.

Test number	Probability
1	0.10
2	0.20
3	0.25
4	0.50
5	0.50
6	0.50
7	0.75
8	0.75
9	0.75
10	1.00

Table 4.1: *Edge creation probability for each Maximum Matching test case number.*

Each of the 100 test cases was executed 50 times to account for machine performance variability, resulting in 5,000 total executions.

Hardware specifications

The benchmark was executed in a computer with the following specifications:

CPU	Intel(R) Core(TM) i7-9750H
RAM	32GB

Algorithms tested

The fast matrix multiplication algorithm was **not** implemented since its threshold is too high for application purposes. Instead the trivial $O(n^3)$ algorithms were implemented, thus the tested algorithms have the following time complexities, where n is the number of vertices and m is the number of edges.

Algorithm	Time complexity
Rank-two algorithm	$O(n^2m + n^3)$
Harvey's algorithm	$O(n^3)$
Edmonds-Blossom algorithm	$O(n^2m)$

4.2.2 Results

Figure Figure 4.1 illustrates the algorithmic performance across test cases, where the first input number represents the number of vertices and the second represents the upper bound on the matching number. The Edmonds-Blossom algorithm demonstrated superior performance compared to the alternative approaches.

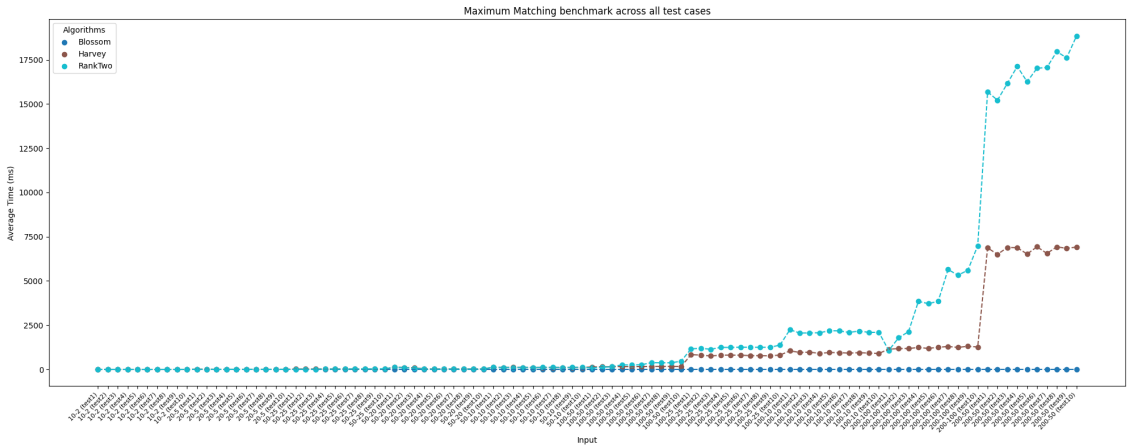

Figure 4.1: *Maximum Matching benchmark with all test cases.*

Similarly to the perfect matching algorithm, in small graphs (Figure 4.2) the Rank-two algorithm outperforms the Harvey's algorithm.

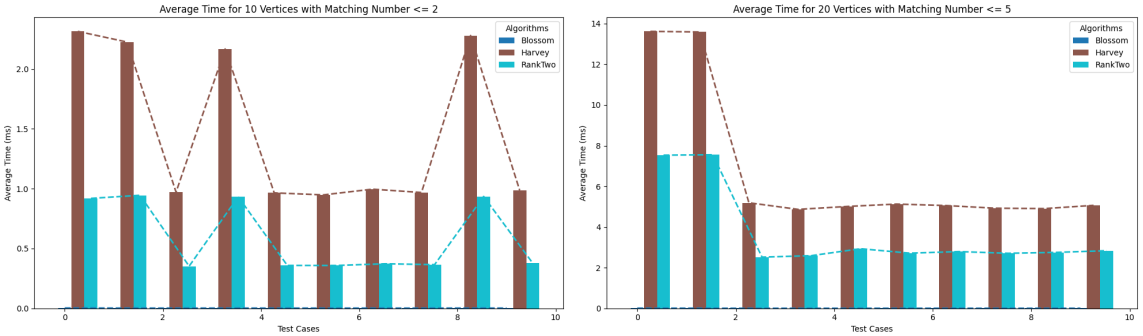

Figure 4.2: *Maximum matching benchmark with smaller graphs.*

In Figure 4.3, the impact of matching number on algorithm performance becomes evident through the augmented graph. The left plot shows both algorithms maintaining stable, similar performance, likely due to the augmented graph's almost doubling the number of vertices. Conversely, the right plot exhibits behavior reminiscent of the perfect matching algorithm, attributed to fewer added vertices in the augmented graph.

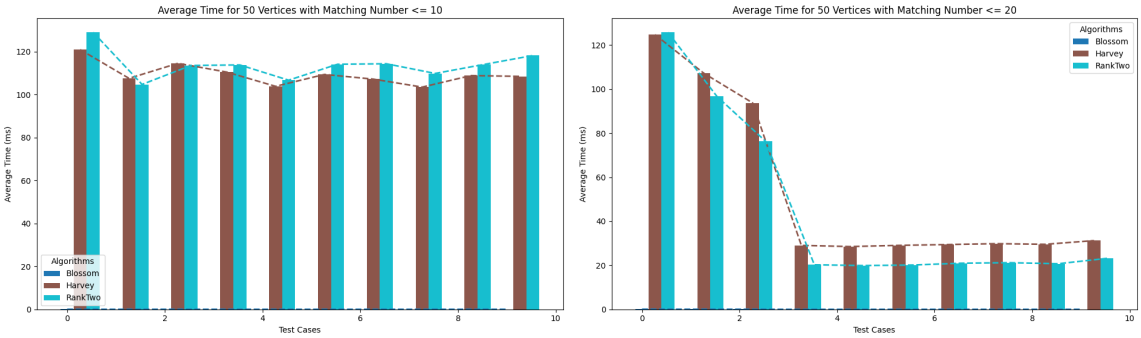

Figure 4.3: *Maximum matching benchmark with 50 vertices.*

In Figure 4.4 and Figure 4.5, Harvey's algorithm outperforms the Rank-two algorithm. The performance trend observed in Figure 4.3 recurs: as the graph's matching number increases, algorithm performance varies significantly due to augmented graph characteristics.

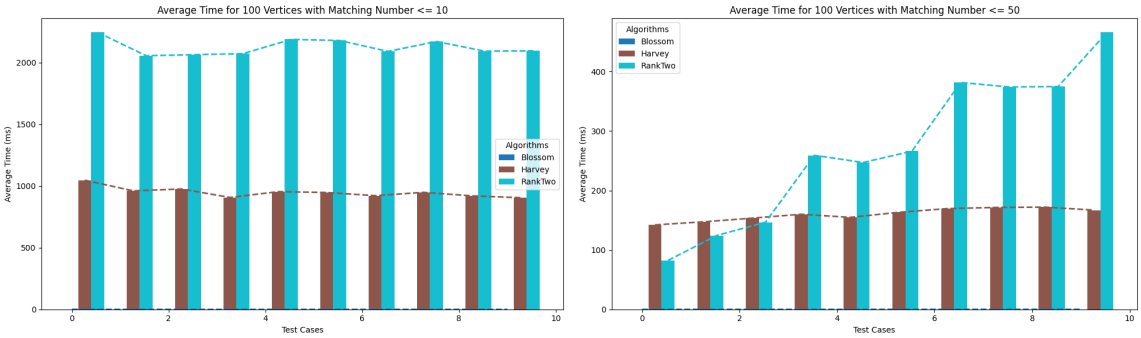

Figure 4.4: *Maximum matching benchmark with 100 vertices.*

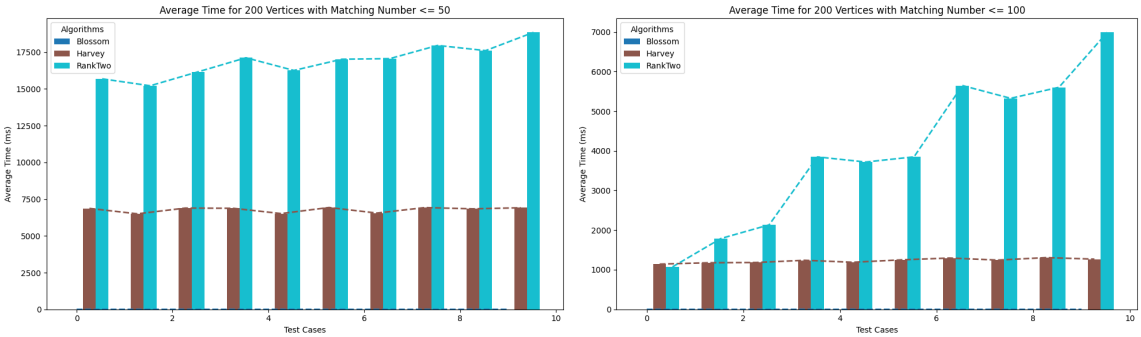

Figure 4.5: *Maximum matching benchmark with 200 vertices.*

Chapter 5

Conclusion

In this paper, we implemented the randomized maximum matching algorithm introduced by Harvey [6]. The algorithm's approach leverages algebraic graph theory, contrasting with traditional methods like the Edmonds-Blossom [4] algorithm. While Harvey's algorithm achieves a superior theoretical time complexity of $O(n^{\omega})$ (where n represents the number of vertices and ω is the matrix multiplication exponent), our implementation analysis in Section 3.2.2 and Section 4.2.2 reveals significant practical limitations. The high constant factor in the complexity makes the algorithm less efficient for smaller graphs. Theoretically, a performance threshold exists where this approach surpasses the Edmonds-Blossom algorithm, but this threshold was neither found nor tested.

Nonetheless, when considering the implementation aspects, Harvey's algorithm offers notable advantages. Setting aside the mathematical foundations behind the updates (e.g. Theorem 3.1.1), the algorithm provides a significantly more intuitive and straightforward implementation compared to the Edmonds-Blossom algorithm. This simplicity makes it particularly valuable for applications involving larger graphs or scenarios where absolute computational efficiency is not the primary concern.

References

[1] J. Alman et al. *More Asymmetry Yields Faster Matrix Multiplication.* 2024. arXiv: 2404.16349 (cit. on p. 5).

[2] D. Coppersmith and S. Winograd. "Matrix multiplication via arithmetic progressions". In: *Journal of Symbolic Computation* 9.3 (1990). Computational algebraic complexity editorial, pp. 251–280. ISSN: 0747-7171 (cit. on p. 5).

[3] G. Delfino. *Emparelhamento em grafos: Algoritmos e aplicações.* 2017 (cit. on pp. 16, 21).

[4] J. Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467 (cit. on pp. 1, 25).

[5] C. D. Godsil. *Algebraic Combinatorics.* Chapman & Hall, 1993 (cit. on p. 6).

[6] N. J. A. Harvey. "Algebraic Algorithms for Matching and Matroid Problems". In: *SIAM Journal on Computing* 39.2 (2009), pp. 679–702 (cit. on pp. 1, 4, 11, 12, 25).

[7] O. H. Ibarra, S. Moran, and R. Hui. "A generalization of the fast LUP matrix decomposition algorithm and applications". In: *Journal of Algorithms* 3.1 (1982), pp. 45–56. ISSN: 0196-6774 (cit. on p. 5).

[8] L. Lovász. "On determinants, matchings and random algorithms". In: vol. 79. Jan. 1979, pp. 565–574 (cit. on p. 7).

[9] L. Lovász and M. D. Plummer. *Matching Theory.* Annals of Discrete Mathematics. North Holland, 1986. ISBN: 9780080872322 (cit. on p. 20).

[10] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995 (cit. on p. 7).

[11] M. O. Rabin and V. V. Vazirani. "Maximum matchings in general graphs through randomization". In: *Journal of Algorithms* 10.4 (1989), pp. 557–567. ISSN: 0196-6774 (cit. on pp. 1, 20).

[12] V. Strassen. "Gaussian elimination is not optimal". In: *Numer. Math. (Heidelb.)* 13.4 (Aug. 1969), pp. 354–356 (cit. on p. 5).

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms, Third Edition.* 3rd. The MIT Press, 2009. ISBN: 0262033844 (cit. on pp. 5, 13).

[14] W. T. Tutte. "The Factorization of Linear Graphs". In: *Journal of the London Mathematical Society* s1-22.2 (1947), pp. 107–111 (cit. on p. 6).