Formalizing Combinatorial Matrix Theory

FORMALIZING COMBINATORIAL MATRIX THEORY

BY

FERNÁNDEZ, ARIEL GERMÁN,

A THESIS

SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

McMaster University

DOCTOR OF PHILOSOPHY (2013) McMaster University

(Computer Science) Hamilton, Ontario, Canada

TITLE:	Formalizing Combinatorial Matrix Theory
AUTHOR:	Fernández, Ariel Germán
SUPERVISOR:	Dr. Soltys-Kulinicz, Michael
NUMBER OF PAGES:	xii, 257

A mi papá

Abstract

In this thesis we are concerned with the complexity of formalizing reasoning in Combinatorial Matrix Theory (CMT). We are interested in the strength of the bounded arithmetic theories necessary in order to prove the fundamental results of this field. Bounded Arithmetic can be seen as the uniform counterpart of Propositional Proof Complexity.

Perhaps the most famous and fundamental theorem in CMT is the König's Min-Max Theorem (KMM) which arises naturally in all areas of combinatorial algorithms. As far as we know, in this thesis we give the first feasible proof of KMM. Our results show that Min-Max reasoning can be formalized with uniform Extended Frege.

We show, by introducing new proof techniques, that the first order theory **LA** with induction restricted to Σ_1^B formulas—i.e., restricted to bounded existential matrix quantification—is sufficient to formalize a large portion of CMT, in particular KMM. Σ_1^B-**LA** corresponds to polynomial time reasoning, also known as $\exists$**LA**.

While we consider matrices over $\{0,1\}$, the underlying ring is $\mathbb{Z}$, since we require that ΣA compute the number of 1s in the matrix A (which for a 0-1 matrix is simply the sum of all entries—meaning ΣA). Thus, over $\mathbb{Z}$, **LA** translates to $\mathbf{TC}^0$-Frege, while, as mentioned before, $\exists$**LA** translates into Extended Frege.

In order to prove KMM in $\exists$**LA**, we need to restrict induction to Σ_1^B formulas.

The main technical contribution is presented in Claim 4.3.4, Section 4.3.3. Basically, we introduce a polynomial time procedure, whose proof of correctness can be shown with $\exists\mathbf{LA}$, that works as follow: given a matrix of size $e \times f$ such that $e \leq f$, where the minimum cover is of size e, our procedure computes a maximum selection of size e, row by row.

Furthermore, we show that Menger's Theorem, Hall's Theorem, and Dilworth's Theorem—theorems related to KMM—can also be proven feasibly; in fact, all these theorems are equivalent to KMM, and the equivalence can be shown in **LA**. We believe that this captures the proof complexity of Min-Max reasoning rather completely.

We also present a new Permutation-Based algorithm for computing a Minimum Vertex Cover from a Maximum Matching in a bipartite graph. Our algorithm is linear-time and computationally very simple: it permutes the rows and columns of the matrix representation of the bipartite graph in order to extract the vertex cover from a maximum matching in a recursive fashion. Our Permutation-Based algorithm uses properties of KMM Theorem and it is interesting for providing a new permutation—and CMT—perspective on a well-known problem.

Acknowledgements

I would like to thank my supervisor Michael Soltys-Kulinicz for fruitful inspiration, careful guidance, enthusiasm, patient, and support; it is thanks to Michael that this thesis has been written, thank you for introducing me to the beautiful land of Complexity Theory and Logic.

I want to thank many people for encouragement and support during my studies, my "mamá", my lovely Noelia, my brothers, "the Russian", and my chosen brothers who always are in my heart.

Contents

List of Figures

Chapter 1

Introduction

> "A major goal of my field is to sort out which kinds of computer problems can be solved efficiently, and which are intractable. Impossibility proofs can help in constructing useful algorithms in the same way that the conservation of energy principle helps in the design of machines: Don't waste your time trying to build a perpetual motion machine."
>
> Stephen A. Cook

Our main goal is to formalize concepts from Combinatorial Matrix Theory using a Proof Complexity approach.

We show a feasible framework to analyze and formalize concepts in Combinatorial Matrix Theory—concepts which are a cornerstone in different fields of Discrete Mathematics and Computer Science. We take a proof-complexity approach to formalizing Min-Max type of reasoning within our feasible framework. Also, we will see some examples—inside of our framework—of combinatorial-graph theoretical theorems such as Menger's Connectivity Theorem, Hall's System of Distinct Representative Theorem, and Dilworth's Decomposition Theorem for Partially Ordered Sets.

1.1 Motivation

There are two fundamental pillars with respect to the motivation of this thesis: *Combinatorial Matrix Theory* and *Bounded Reverse Mathematics*. Inside each of these vast fields, we show what motivates us.

From Combinatorial Matrix Theory

Combinatorial Matrix Theory is a branch of mathematics that combines Graph Theory, Combinatorics and Linear Algebra; it is concerned with the use of matrix theory and linear algebra to prove combinatorial theorems.

Combinatorial Matrix Theory studies patterns of entries in a matrix rather than values, using graphs or digraphs to describe patterns. In some applications, only the sign of the entry—or whether it is nonzero—is known, not the numerical value, and in other cases, some entries are missing. In some sense, it considers matrices as two dimensional strings.

From the computational-algorithmic point of view, we were motivated with the Combinatorial Matrix Theory field by showing that its application in Theoretical Computer Science field, for instance, yields not only a new Permutation-Based Algorithmic approach to a well-known problem (e.g., *Minimum Vertex Cover Problem*, see Chapter 3) but also several advantages like simplicity, more understanding of the structural concepts behind the Permutation-Based Algorithm, all this, maintaining a good performance in the running time, as we show in Chapter 3.

The algorithmic problem, named *Vertex Cover Problem*, roughly speaking, says that "Vertex Cover (VC) on input $\langle G, k\rangle$ graph G and parameter k asks if there is a cover C of G of size at most k, where a cover C is a subset of vertices in V_G such that each edge in E_G has at least one end point in C."

In a bipartite case, there is a close link between VC and matching; a VC can be obtained from a maximum matching by selecting one end point from each edge in the matching.

There is—in the finite case—no feasible algorithm which means that there is no algorithm that goes through the graph and, as it finds each edge, decides whether to put it into the required matching, and if so, which vertex goes into the desired cover.

In order to understand more the motivations related to this particular problem, let us see some historical events. In 1916, in two nearly identical papers—one in German [Kön16b], the other in Hungarian [Kön16a]—König proved that every doubly stochastic matrix[1] with non-negative entries must have a non-zero term in its determinant. In the same papers König also proved that every regular[2] bipartite graph has a perfect matching. In the late 1950's Dulmage and Mendelsohn published papers ([DM58a, DM58b]) in which they worked out a canonical decomposition theory for bipartite graphs in terms of maximum matchings and minimum vertex covers. In 1972 the problem of finding a minimum vertex cover was proven to be a typical example of an **NP**-hard problem; in fact, it was one of Karp's 21 original **NP**-complete problems [Kar72]. (For more details on **NP** problems see [GJ79]). In the 1980's excellent surveys of Min-Max results appeared, such as [Sch82] and [LP86]. The original paper presenting the HK-Algorithm is [JEH73].

On one hand, it is well-known that for general graphs, the maximum matching problem can be solved in polynomial time by Edmond's blossom shrinking algorithm [Edm65], and for bipartite graphs also in polynomial time by Hopcroft-Karp

[1] A *doubly stochastic matrix* of order n is a non-negative $n \times n$ matrix A in which all rows and columns sums are equal to 1.

[2] A graph is *regular* is every vertex has the same degree, where degree of a vertex v is the number of edges incident with v.

algorithm [JEH73].

On the other hand, the minimum vertex cover problem, as we mentioned before, for general graphs is **NP**-complete, but surprisingly, for bipartite graphs, it is polynomial time. In this last case, König's Min-Max Theorem makes the difference, because it relates matchings and covers allowing minimum vertex cover and maximum matching to be computed feasibly.

From Bounded Reverse Mathematics

Studying the complexity of mathematical practice, some structures are more complicated than others, some constructions more complicated than others, and some proofs more complicated than others. Understanding how to measure this complexity is interesting. From a computational viewpoint, it is important to know what part of mathematics can be done by mechanical algorithms, and, even for the part that can not be done mechanically, we want to know how constructive are the objects we deal with.

The main goal of *Bounded Reverse Mathematics* is to find the weakest theory capable for proving a given theorem—in our case we "use" as "a source of theorems" the mathematical field of Combinatorial Matrix Theory—and in general these can be proved in weak theories. The adjective *Bounded* refers to bounds on quantified variables.

The point of Proof Complexity is to see which combinatorial theorems have feasible proofs. Then, Proof Complexity takes the **P vs. NP** problem from computation and "translates" it into **NP vs. co-NP** in proofs.

More precisely, Proof Complexity is an area of Mathematics and Theoretical Computer Science that studies the length of proofs in propositional logic. It is

an area of study that is fundamentally connected both to major open problems of Computational Complexity Theory and practical properties of Automated Theorem Provers [BP98].

A propositional formula γ is a *tautology* if γ is true under all truth value assignments. Let $TAUT$ be the set of all tautologies. A *propositional proof system* is a polytime predicate $P \subseteq \Sigma^* \times TAUT$ such that:

$$\gamma \in TAUT \iff \exists x P(x, \gamma)$$

P is *poly-bounded* if there exists a polynomial p such that:

$$\gamma \in TAUT \iff \exists x(|x| \leq p(|\gamma|) \ \wedge \ P(x, \gamma))$$

The existence of a poly-bounded proof system is related to the fundamental question:

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

In 1979 S. Cook and R. Reckhow [CR79] proved that NP=co-NP if and only if there is a poly-bounded propositional proof system for tautologies. On the other hand, if P = NP then NP = co-NP. Thus, if there is no poly-bounded proof system, then NP $\neq$ co-NP, and that in turn would imply that P $\neq$ NP.

A fundamental notion that appears throughout this thesis is that of *feasible proof* (and *feasible computation*, or *polynomial time computation*). Feasible proofs were introduced by Cook in [Coo75], and they formalize the idea of tractable reasoning; a theorem can be proven feasibly, if all the computations involved in the proof are polynomial time computations, and the induction can be "unwound" feasibly. (For

more details about why we want to find feasible proofs see Section 4.1.1). Combinatorial theorems of interest in computer science often have polynomial time proofs, and appear throughout all corners of Combinatorial Matrix Theory field, examples of such theorems are König's Min-Max Theorem, Dilworth's Theorem, Hall's Theorem, Menger's Theorem, etc. (For more details see Chapter 4).

From the complexity-theoretic point of view, the idea is to find the smallest complexity class such that a theorem can be proved using concepts in that class.

In this thesis we concentrate our study into the **LA**-Theory (see [Sol01]). The main reason is that **LA** was designed for reasoning in matrices.

From a Proof Complexity

The *Proof Complexity* (PC) of *Combinatorial Matrix Theory* (CMT) is related to the *Proof Complexity* of linear algebra, and hence we are going to use **LA**-Theory in order to formalize reasoning in CMT.

The PC has two aspects, the *uniform* concerns the power of logical theories required to prove a given assertion, and the *nonuniform* aspect which concerns the power of propositional proof systems required to yield polynomial size proofs of a tautology family representing the assertion. Here we are concerned with the *uniform* case.

We concentrate on theorems involving concepts of smaller complexity especially below the polynomial hierarchy. More specifically, we classify these theorems based on the computational complexity of concepts needed to prove them. This is a fundamental issue of PC, which it will shed light on complexity classes, by providing proofs of low complexity.

The *Proof Complexity of Linear Algebra*, that is, the **LA**-Theory, roughly speaking, is the complexity of concepts needed to prove the basic properties of linear algebra operations.

More precisely, **LA**-Theory is a field-independent logical theory for expressing and proving matrix properties. **LA** proves all the ring properties of matrices (i.e., commutativity of matrix addition, associativity of matrix products, etc.).

While **LA** is strong enough to prove all the ring properties of matrices, its propositional proof complexity is low, that is, all the theorems of **LA** over the standard field of two elements, translate into families of propositional tautologies with polynomial size bounded-depth Frege proofs, with "$\oplus$" gates of unbounded fan-in, that is, $\mathbf{AC}^0[2]$-Frege, in general, **LA**-Theory over fields $\mathbb{Z}_p$: polynomial Bounded Depth Frege with `MOD` p gates translate into the complexity class $\mathbf{AC}^0[p]$ (see [Sol01] for this result).
While we consider matrices over $\{0,1\}$, the underlying ring is $\mathbb{Z}$, since we required that ΣA compute the number of 1s in the matrix A (which for a 0-1 matrix is simply the sum of all entries—meaning ΣA). Thus, over $\mathbb{Z}$, **LA** translate to $\mathbf{TC}^0$-Frege, [SC04, §6.5].

So, we investigate a formalization of Min-Max theorems in the **LA**-Theory, with a detailed proof complexity analysis, with the main goal of "extracting" the weakest fragment of **LA** that can prove feasible, for instance, König Min-Max Theorem. To be more specific, sometimes we use a conservative extension named $\exists$**LA** or $\forall$**LA** which incorporates bounded existential (universal) matrix quantifiers to the quantifier-free **LA** formulas, sometimes, we refer to these conservative extensions

of **LA** like a **Bounded Matrix Algebra**[3] Theory, bounded because of the bounds in matrix quantifiers and Matrix Algebra because, to be more specific, **LA**-Theory is a Matrix Algebra Theory. The theory ∀**LA** is the extension of **LA** that also includes induction axioms for ∀**LA** formulas, and ∃**LA** extension is similar. Besides, these theories translate into Extended Frege.

We use the approach—the Proof Complexity view—outlined in [CN10] which also constitutes the main reference to the field of Proof Complexity for our propose. The reader interested in general treatises in Proof Complexity should check [Kra95, Bus98], and the references contained in these.

The following note has the propose of summarizing what is Reverse Mathematics about.

Inside computability theory, *Reverse mathematics* analyzes the complexity of mathematical theorems in terms of the complexity of the constructions needed for their proofs.

The questions of which axioms and rules are necessary to do mathematics is of great importance in Foundations of Mathematics and is the main question behind Simpson and Friedman's program of Reverse Mathematics. To analyze this question formally it is necessary to fix a logical system. Reverse Mathematics deals with the system of Second Order Arithmetic. Second Order Arithmetic, though much weaker than set theory, is rich enough to be able to express an important fragment of classical mathematics.

The idea of Reverse Mathematics goes as follows. We start by fixing a basic system of axioms as a base. The most commonly used base system is called RCA_0,

[3]In particular we use this name in the mind-map depicted at the end of this section.

that roughly speaking says that the natural number form an ordered semi-group and that a computable set exists. More formally, RCA_0 is the most fundamental subsystem[4] of $GF(2)$, named for the *Recursive Comprehension Axiom*. This restrict the comprehension scheme to Δ^0_1-formulas, i.e., those which define recursive sets[5]. The subscript 0 in the name indicate that RCA_0 only has limited induction.

In our case, we use the axioms of **LA**-Theory, that is, all the axioms for equality (for indices, field elements, and matrices), the axioms for indices which are the usual axioms of Robinson's Arithmetic in **LA** together with axioms defining `div`, `rem`, and `cond`; the axioms for matrices (including those that define the behaviour of constructed matrices, as well as the axioms defining Σ function), and finally, the axioms for field elements are the usual field axioms.

Also we use the axiom convention: *All substitution instances of axioms are also axioms.* Thus, our axioms are really axiom schemas.

Recall that, $\Sigma^B_0 = \Pi^B_0$ formulas are those which correspond to the set of matrix-quantifier-free bounded formulas, for instance, bounded index-type element formula. Σ^B_1 formulas are those which when presented in prenex form, contain a single block of bounded existential matrix quantifiers—Π^B_1 are defined similarly, except the block of quantifiers is universal.

In general, Σ^B_i is the set of formulas which, when presented in prenex form, start with a block of bounded existential matrix quantifiers, followed by a block of bounded universal matrix quantifiers, with i such alternating blocks. The set Π^B_i is defined similarly, except it starts with a block of universal quantifiers.

[4] A *subsystem* if $GF(2)$ is a formal system S in the language of Second Order Arithmetic where each axiom $\phi \in S$ is a theorem of $GF(2)$.

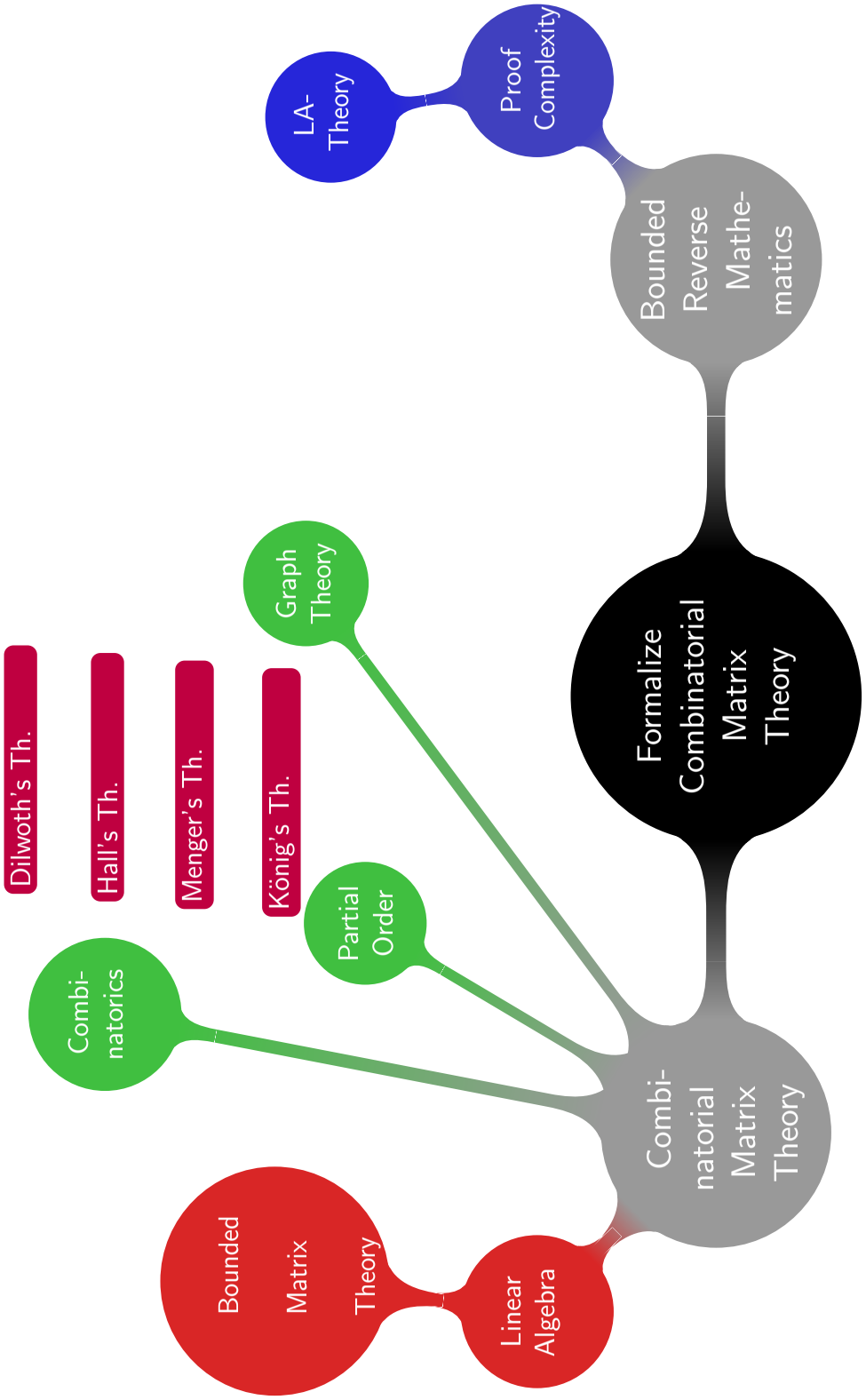
[5] A set $A \in \mathbb{N}$ is *recursive* if and only if its characteristic function $f : \mathbb{N} \to \{0, 1\}$ is total and computable.

Finally, we restrict the induction rule to Σ_i^B-formulas (eventually Π_i^B-formulas), getting the Σ_i^B-IND rule, respectively Π_i^B-IND rule. (For more details about **LA** see Section 4.1.2 , and Appendix A.5).

Now, given a theorem of "ordinary" mathematics (we work mainly with Combinatorial Matrix Theory theorems), the question we ask is what axioms and rules do we need to add to the base system to prove this theorem. It is often the case in Reverse Mathematics that we can show that a certain set of axioms and rules are necessary to prove a theorem by showing, using the base system that the axioms and rules follow from the theorem. Because of this idea, this program is called Reverse Mathematics. (For more details see [Sim99, Sim92]).

Motivation Mind-Map

The following mind-map summarize the fundamentals sources of motivations and their relation within this thesis.


Proof Complexity
LA-Theory
Bounded Reverse Mathe-matics
Formalize Combinatorial Matrix Theory
Graph Theory
Dilwoth's Th.
Hall's Th.
Menger's Th.
König's Th.
Partial Order
Combi-natorics
Combi-natorial Matrix Theory
Bounded Matrix Theory
Linear Algebra

1.2 Contributions

In this section we present the contributions of this thesis in order of major importance.

The main contributions of this thesis are concentrated in Chapter 4 in which we present a feasible proof of König's Min-Max Theorem, by consequence an important series of proofs of several main theorems in different areas, like Combinatorics, Graph Theory, Order Theory, etc. We present feasible proofs of Dilworth's Decomposition Theorem for partial order sets (see Section 4.3.5.3), a feasible proof of Hall's System of Distinct Representatives Theorem (see Section 4.3.5.2), and a feasible proof of Menger's Connectivity Theorem (see Section 4.3.5.1), a particular case of the well-known Max-Flow Min-Cut Theorem.

To be more precise, in Chapter 4 we show that the well-known König's Min-Max Theorem (KMM), a fundamental result in Combinatorial Matrix Theory, can be proven in the First Order Theory **LA** with induction restricted to Σ_1^B formulas. The standard proofs of KMM which requires Π_2^B induction (see Appendix A.1), and hence does not yield feasible proofs (see Section 4.1.1)—while our new approach does. In order to prove KMM in $\exists$**LA**, we need to restrict induction to Σ_1^B formulas. The main technical contribution is presented in Claim 4.3.4 Section 4.3.3. Basically, we introduce a polynomial time procedure, whose proof of correctness can be shown with $\exists$**LA**, that works as follow: given a matrix of size $e \times f$ such that $e \leq f$, where the minimum cover is of size e, our procedure computes a maximum selection of size e, row by row.

LA is a weak theory that essentially captures the ring properties of matrices; however, equipped with Σ_1^B induction **LA** is capable of proving KMM, and a host of fundamentals combinatorial-graph theory theorems such as Menger's, Hall's and

Dilworth's Theorems. Therefore, our result formalizes Min-Max type of reasoning within a feasible framework.

More formally, in Chapter 4 we show the following two theorems :

Theorem $\exists\mathbf{LA} \vdash$ KMM.

Theorem *The theory* **LA** *proves the equivalence of* KMM*, Menger's, Hall's and Dilworth's Theorems.*

Besides, in Chapter 4 we show the standard KMM Theorem is stated as an implication (see Equation (4.9)), and hence it makes no assertions about the actual existence of a minimal covering or maximal selection of 1s, let alone how to compute them. It only says that if they do exist, they are equal. However, the proof of Lemma 4.3.2 suggests an algorithm for computing both.

Note that computing a minimal cover can be accomplished in polytime with the well-known Karp-Hopcroft (KH-Algorithm)—sometimes called HK-Algorithm, see [JEH73]—as follows: First use the KH-Algorithm to compute a "maximal matching," which in this case is simply a maximal selection of 1s (when we view A—in the natural way —as the adjacency matrix of a bipartite graph). Then in Chapter 3 we present an algorithm to convert, in linear time, a maximal selection into a minimal cover.

Certainly the correctness of the algorithms mentioned in the above paragraph can be shown in $\exists\mathbf{LA}$ (as it captures polytime reasoning—see [Sol01]), and so it follows that we can prove in $\exists\mathbf{LA}$ the existence of a minimal cover and maximum selection.

Therefore, $\exists\mathbf{LA}$ can prove something stronger than (Equation 4.9). Namely, it can not only show that *if* we have a minimal cover and a maximal selection, then they have the same size, but rather, that there *always exists* a minimal cover and maximal selection, and the two are of equal size.

However, instead of doing the heavy lifting necessary to formalize the correctness of HK-Algorithm and [SF12] in $\exists\mathbf{LA}$, we present a new simple polytime algorithm for computing minimal covers based on the proof of Lemma 4.3.2. Note that using a similar argument we show the existence of the dual polytime algorithm for maximal selection too.

On the other hand, in Chapter 3 we present a new linear-time Permutation-Based Algorithm for computing minimum vertex covers from maximum matching. More precisely, we present our new algorithm that on input $G = (V, E)$ and a maximum matching M_G, constructs a vertex cover for G in time $|V|^2$, and therefore substantially faster than the natural reduction from search to decision when the number of edges, $|E|$, is large.

In other words, the contribution is a new Permutation-Based Algorithm for computing a Minimum Vertex Cover from a Maximum Matching in a bipartite graph. Our algorithm is linear-times and computationally very simple: it permutes the rows and columns of the matrix representation of the bipartite graph in order to extract the vertex cover from the matching in a recursive fashion. Our algorithm uses properties of König's Min-Max Theorem and it is interesting for providing a new permutation—and Combinatorial Matrix Theory—perspective on a well-known problem.

In Section 3.4 we study in more detail the computational complexity of our Permutation-Based Algorithm, inside of this section we show that we make use of the concept of cycle decomposition of permutations to use linear representation of permutation matrices, and to operate on them.

1.3 Summary of results

In this section we summarize—again in order of major importance—the main results of this thesis in a table format.

Table 1.1: Summary of results by Chapter[Section]

Chapter[Section]	**Summary**
4[4.3.3]	We present a feasible proof of König's Min-Max Theorem. More formally, we prove $\exists\mathbf{LA} \vdash$ KMM (Theorem 4.3.1).

Continued on next page

Table 1.1 – *Continued from previous page*

Chapter[Section]	**Summary**
4[4.3.4]	The standard König's Min-Max Theorem is stated as an implication (see Equation 4.9), and hence it makes no assertions about the actual existence of a minimal covering or maximal selection of 1s, let alone how to compute them. It only says that if they do exist, they are equal. However, the proof of Lemma 4.3.2 suggests an algorithm for computing both. We present such new simple polytime algorithm for computing minimal covers and maximum matching based on the proof of Lemma 4.3.2. Besides, we show that ∃**LA** can prove something stronger than Equation 4.9. Namely, it can not only show that *if* we have a minimal cover and a maximal selection, then they have the same size, but rather, that there *always exists* a minimal cover and maximal selection, and the two are of equal size. In short, we emphasize that there is a difference between the following two statements: • *if* minimum cover and maximum selection exists, then they are equal, which is how it is KMM stated in Equation 4.9. • minimum cover and maximum selection exists and are equal, which is what we proved.

Continued on next page

Table 1.1 – *Continued from previous page*

Chapter[Section]	**Summary**
4[4.3.5]	We prove that the theory **LA** proves the equivalence of KMM, Menger's, Hall's and Dilworth's Theorems. To be more specific, we prove the following important lemmas: • $\mathbf{LA} \cup \text{Menger} \vdash \text{KMM}$ (Lemma 4.3.3), and $\mathbf{LA} \cup \text{KMM} \vdash \text{Menger}$ (Lemma 4.3.4), • $\mathbf{LA} \cup \text{Hall} \vdash \text{KMM}$ (Lemma 4.3.6), and $\mathbf{LA} \cup \text{KMM} \vdash \text{Hall}$ (Lemma 4.3.5), • $\mathbf{LA} \cup \text{KMM} \vdash \text{Dilworth}$ (Lemma 4.3.7), and $\mathbf{LA} \cup \text{Dilworth} \vdash \text{KMM}$ (Lemma 4.3.9). In other words: • We formalize concepts in **LA**-Theory, and • all are equivalents in **LA**.

Continued on next page

Table 1.1 – *Continued from previous page*

Chapter[Section]	**Summary**
3[Sec. 3.3]	We present a new linear-time Permutation-Based Algorithm for computing Minimum Vertex Covers from Maximum Matchings. Expressed in the following theorem (Theorem 3.3.1): **Theorem** *Given a bipartite graph* $G = (V = V_1 \cup V_2, E)$*, we obtain a procedure for computing a minimum vertex cover from a maximum matching that runs in time* $|\langle A, M_A \rangle| = |V|^2$*. That is, given a matrix and the corresponding maximum matching, we can compute the vertex cover in linear time (in the size of the input).* With the follow characteristics: • It's interesting for providing a *new permutation perspective* on a well-known problem *Minimum Vertex Cover Problem*. • It's *linear-times* and *computationally very simple*. • It works *directly* on a given maximum matching M. • It *doesn't recompute* the bipartite graph G. • It's *much faster* when $\|E\| >> \|\sqrt{\|V\|}\|$. • It uses a *simple and more accurate data Structure*.

Continued on next page

Table 1.1 – *Continued from previous page*

Chapter[Section]	**Summary**
3[Sec. 3.4.1.2]	We use of the concept of cycle decomposition of permutations to use linear representation of permutation matrices, and operate on them.
3 [Sec. 3.4.1.3]	Tight Complexity Analysis of our Permutation-Based Algorithm, with $R(n) \leq n^2$ where $R(n)$ is the maximum number of steps in the worst-case that our procedure takes on matrix of size n. (See Claim 3.4.2)
Appendix[A.1]	We give a Π_2^B-Inductive proof of König's Min-Max Theorem in **LA**.
Appendix[A.2]	Give a more detailed **LA**-proof of Lemma 4.3.1 Section 4.1
Appendix[A.3]	Give a detailed partial correctness and termination proof of our Permutation-Based Algorithm (Chapter 3)

Chapter 2

Background

In this chapter we shall see two major algorithms to solve the problem of finding a maximum matching, they are the well-known approach called *Hungarian algorithm*, and the most effective approach known named *Hopcroft-Karp algorithm*.

Given a bipartite graph $G = (V = V_1 \cup V_2, E)$, i.e., a graph where $E \subseteq V_1 \times V_2$, and let $|E| = m$ and $|V| = n$, both algorithm are polynomial time, the first one with a running time of $O(mn)$, and the second one improving the running time of the first one by $O(m\sqrt{n})$.

The main goal of this chapter is show these different approaches to the maximum matching problem, in order to have enough background to continue with the exposition of our Permutation-Based Algorithm in Chapter 3.

We finish this chapter with a slightly formal and small introduction to what is Combinatorial Matrix Theory, which not only is a very rich mathematical field by "its ingredients"—graph theory, linear algebra, and combinatorics—but also it is a rich source of algorithmic concepts and very useful theorems to Computer Science.

2.1 Definitions

In this section we are going to review a few definitions that are used in the analysis of the Hungarian algorithm (Section 2.2), the Hopcroft-Karp algorithm (Section 2.3), and throughout the rest of the thesis.

Sometimes, in order to make more self-contained each chapter or section, we shall review a few definitions, and add another.

All the definitions, lemmas, and theorems that we present in this section, and some classical result that we shall show in this chapter come most from [We01], and [Zwi09]. For more detail turn to [ADH98, CLRS09, Als99].

We begin with the basic terminology,

Definition 2.1.1 *A* graph *G is a triple consisting of a* vertex set *V_G or simply V when the context is clear, an* edge set *E_G or simply E, and a relation that associates with each edge two vertices—no necessary distinct—called its* endpoints*.*

Every graph mentioned in this thesis is *finite*—vertex and edge sets finite—unless explicitly constructed otherwise. When in an edge the endpoints are equals, the edge is called *loop*. A graph that has no multiple edges and it has no loops is a *simple graph*. When u and v are the endpoints of an edge, they are *adjacent* and *neighbours*.

Some useful concepts and terminology about the structure of graphs are expressed in the following important definition:

Definition 2.1.2 *The complement $\overline{G}$ of a simple graph G is the simple graph with vertex set V_G defined by $(u, v) \in E_{\overline{G}} \iff (u, v) \notin E_G$. A* clique *in a graph is a set of pairwise adjacent vertices. An* independent set *in a graph is a set if pairwise nonadjacent vertices.*

Definition 2.1.3 *A* directed graph *or* digraph *G is a triple consisting of a* vertex set *V_G or simply V when the context is clear, an* edge set *E_G or simply E, and a function assigning each edge an ordered pair of vertices. The first vertex of the ordered pair is the* tail *of the edge, and the second is the* head*; together, they are the* endpoints*. We say that an edge is an edge* from *its tail* to *its head.*

The previous concept—Definition 2.1.2—is used, for instance, in the core of Dilworth's Theorem.

Definition 2.1.4 *Let $G = (V, E)$ be a graph, we say that G is a bipartite graph if $V = V_1 \cup V_2$ and $E \subseteq V_1 \times V_2$, with $V_1 \cap V_2 = 0$.*

Sometimes, according to the context, we will use the equivalent classical definition. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph, where V_1 and V_2 are set of vertices of G and E a set of edges of G, with $|V_1| + |V_2| = |V| = n$ and $|E| = m$.

The follows are some definitions which will be useful, particularly, in the context of Menger's Theorem.

Definition 2.1.5 *A* path *of length n from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence $v_1, v_2, \ldots, v_n$ of vertices such that $u = v_1, u' = v_n$, and $\forall i \in \{1, 2, \ldots, n-1\}$ $(v_i, v_{i+1}) \in E$. A path is* simple *if all vertices in the path are distinct. In a directed graph, a path $v_1, v_2, \ldots, v_n$ form a* cycle *is $v_1 = v_n$ and the path contains at least one edge. Te cycle is* simple *if, in addition, $v_1, v_2, \ldots, v_n$ are distinct. A self-loop is a cycle of length 1.*

Definition 2.1.6 *A* subgraph *of a graph G is a graph H such that $V_H \subseteq V_G$ and $E_H \subseteq E_G$ and the assignment of endpoints to edges in H is the same as in G. Then we say G* contains *H, meaning $H \subseteq G$. A graph G is* connected *is each pair of vertices in*

G belongs to a path, otherwise, G is disconnected. *A* maximal *connected subgraph of G is a subgraph that is connected and is not contained in any other connected subgraph of G, it is called* component *of G.*

We are not using the cycle definition explicitly throughout the thesis, but what we do is to use it implicitly because sometimes we are going to mention a special class of bipartite graphs named *tree*—a connected, simple and acyclic graph. A union of disjoint trees is called *forest*.

Now we define two of the most useful representation of graphs.

Definition 2.1.7 *Let G be a graph with vertex set* $V_G = \{v_1, \ldots, v_n\}$ *and edge set* $E_G = \{e_1, \ldots, e_m\}$. *The* adjacency matrix *of G, denoted* A_G, *is the* $n \times n$ *matrix in which each entry* a_{ij} *is the number of edges in G with endpoints* $\{v_i, v_j\}$. *The* incident matrix W_G *is the* $n \times m$ *matrix in which entry* $w_{ij} = 1$ *if* v_i *is an endpoint of* e_j, *otherwise* $w_{ij} = 0$. *The* degree *of vertex v is the number of incident edges.*

Note that an adjacency matrix is defined by a vertex ordering. Every adjacency matrix of an undirected graph is *symmetric*. An adjacency matrix of a simple graph G has entries 0,1—if the edges have no associated cost—with 0s on the main diagonal. The degree of v is the sum of the entries in the row for v in either A_G or W_G.

We finish this section introducing an important problem called *Maximum Matching Problem*, for which we shall define some concepts and lemmas to be more clear the context of the problem. Before continue, note the difference between "maximum" and "maximal", as adjectives, *maximum* means "maximum-sized", and *maximal* means "no larger one contains this one". So, every maximum set is a maximal set, but maximal set need not have maximum size. When describing numbers rather than

containment, the meanings are the same, for instance, maximum vertex degree equals to maximal vertex degree. Dual aspect has same definition, that is, *minimum* and *minimal* concepts.

Let $G = (V, E)$ be an undirected graph. A set $M \subseteq E$ is a *matching* if no two edges in M share a common vertex. A vertex v is *matched* by M if it is part of (is contained) in an edge of M, and *unmatched* otherwise.

Definition 2.1.8 *Let G be a graph and let M be a matching in G. A path P is an alternating path with respect to M if an only if among every two consecutive edges along the path, exactly one belongs to M.*

Definition 2.1.9 *Let A and B be sets. We define their symmetric difference by $A \oplus B = (A - B) \cup (B - A)$.*

For the proofs of the follow lemmas and theorems consult [Zwi09], and the classical textbooks of algorithms, for instance [CLRS09, AHU74, AHU83].

Lemma 2.1.1 *If M is a matching and P is an alternating path with respect to M, where each endpoint of P is either unmatched by M or matched by the edge of P touching it, then $P \oplus M$ is also a matching.*

PROOF: See, for instance, [Zwi09, pg. 2]. □

Definition 2.1.10 *An augmenting path P with respect to a matching M is an alternating path that starts and ends in unmatched vertices.*

Observe that letting G be an undirected graph and let M_1 and M_2 be matchings in G, then, the subgraph $(V, M_1 \oplus M_2)$ is constituted by isolated vertices, alternating paths with respect to both M_1 and M_2.

The following lemmas and concluding theorem clarify the relation among matching, alternating paths, and augmenting paths.

Lemma 2.1.2 *Let G be an undirected graph and let M and M' be matchings in G such that $|M'| = |M| + l$, where $l \geq 1$. Then, there are at least l vertex disjoint augmenting paths in G with respect to M. At least one of these augmenting paths is of length at most $(n/l) - 1$.*

PROOF: See, for instance, [Zwi09, pg. 3]. □

Theorem 2.1.1 *([Ber57]) Let G be an undirected graph and let M be a matching in G. Then, M is a maximum matching in G if an only if there are no augmenting paths with respect to M in G.*

PROOF: See, for instance, [Zwi09, pg. 3]. □

We finish this section announcing the well-known problem called *the Maximum Matching Problem*:

Let $G = (V_1 \cup V_2, E)$ be a bipartite graph, let M a matching in G. The problem is to find a maximum matching in G.

The problem arises in many applications, particularly in areas of communication and scheduling. The problem is interesting in its own right, and it is indispensable as a building block in the design of more complex algorithms. In short, the problem is often used as a subroutine in the implementation of many practical algorithms.

We shall see in the next two sections two different algorithms to solve this important problem in polynomial time.

2.2 Hungarian Algorithm

We give a brief overview of the historical works of matching problem ([LP86, ADH98]). Immediately following World War II, computer scientists focused attention on the development of algorithms, and in particular, a fundamental algorithmic question has been with us since the earliest days of matching theory: *how do you find a maximum matching?* Its importance is perhaps belied by its simplicity of its statement.

The rudiment for finding maximum matching in a bipartite graph had already appeared in the works of König—one in German [Kön16b], the other in Hungarian [Kön16a]—König proved that in a bipartite graph G the size of a largest matching is equal to the size of a smallest set of points which together touch every line in G. In the middle 1950's Kuhn [Kuh55] and M. Hall [Hal56] presented the first formal procedures for finding a maximum matching in a bipartite graph. It seems to have been Kuhn whom at this time first used the phrase "Hungarian Method" to distinguish algorithms of this type. So, in the next section we show in detail how the "Hungarian algorithm" works, mainly, we will make use of [Als99] to discuss it.

Recall a classical theorem—Theorem 2.1.1—which states a relation between maximum matching and augmenting paths. More precisely, let G be an undirected graph and let M be a matching in G, then there are no augmenting paths with respect to (w.r.t.) M in G if an only if M is a maximum matching. We can see that implicitly there is an algorithm for finding a maximum matching which proceed like follows: start with an initial matching M, possibly the empty one, and as long as there is an augmenting path P w.r.t. M, augment M using P (that is invert the condition of matched to unmatched and vice-versa of the edges of P) and repeat.

Hence, we need a procedure for find augmenting paths, if they exist, so, in this

section we try a *natural*[1] approach, called *Hungarian Method or Hungarian algorithm*, (which will succeed only in bipartite graphs) for finding an augmenting path with respect to a matching, say M if one exists. So, in order to analyze how Hungarian algorithm works, and make this section self-contained, we review some definitions.

Let $G = (V_1 \cup V_2, E)$ be a bipartite graph, where V_1 and V_2 are set of vertices of G and E a set of edges of G, with $|V_1| + |V_2| = |V| = n$ and $|E| = m$. A *matching* M in a graph G is a subset $M \subseteq E$ such that no two edges meet at the same vertex. An edge $e \in E$ is *matched* if it is in M, otherwise it is *unmatched*. Similarly, a vertex $v \in V = V_1 \cup V_2$ is *matched* if it is incident to a matched edge, otherwise is *unmatched*.

Notation: We denote vertices that are in V_i with the superscript v^i, where $i \in \{1, 2\}$, for example, a vertex, say u, in V_1 is denoted by a v^1-vertex, say u^1.

2.2.1 Hungarian Trees[2]

Starting from level 0 of the under construction tree, we are going to label vertices at the even levels of a particular kind of tree (an *alternating path tree*, defined below), as *even* and vertices at odd levels as *odd*. Vertices that belong to a tree are labelled while those that are not in a tree are considered unlabelled. Initially all vertices are unlabelled.

So, we choose an unlabelled and unmatched v^1-vertex named the root of a new tree, say r, and declare it unexplored and label it *even*. If there are no unmatched vertices then we obtained a maximum matching[3]. Otherwise, from r, we are going to construct a tree in which each path from the root to the leaf is an alternating path

[1] Natural in the sense that was the first attempt to solve the problem.

[2] Actually, we are constructing a forest of *Hungarian trees*, i.e., a *Hungarian Forest*, that is, undirected, acyclic and possibly disconnected graph.

[3] A *maximum matching* is a "maximum-sized" matching

(for more details see Section 2.1) with respect to a matching M. This tree is called *alternating path tree* denoted by T.

We construct T as follows, as long as there is an unexplored *even* vertex in the tree, at the beginning only the root, we choose one such v^1-vertex, say u, and examine all the edges adjacent to it[4], that is $(u, v) \in E$, where v is an v^2-vertex.

For each $(u, v) \in E$:

- if v is *unmatched*, that is, if we are going to add an unmatched vertex other than the root, to an alternating path tree, then we are in present of an *augmenting path*.

- if v is *matched* and *unlabelled*, we let v' be the v^1-vertex matched to v, that is, $(v', v) \in M$. So, we add v and v', and declare v' to be *unexplored*, and label v odd and v' even. Note that if v is unlabelled then v' is also unlabelled, because v' is in a more depth level than v, and we do not explored in order to label it.

- if v is already labelled *odd*, we do nothing, because we found an alternative odd length alternating path from an unmatched vertex to v. Note that v may be part of a previously constructed tree, because may be we explored it in the existent alternative path.

- if v is already labelled *even*, it cannot belong to a previous constructed tree, because the edge (u, v) would have been explored from u, and we are in the case that v is unexplored. So, v belong to the same tree as u, but this cannot happen because we are in the presence of an odd cycle in a bipartite graph. Hence, that this last case cannot happen in our algorithm.

[4]We assume that such edge exists, otherwise we have a isolate node and we can ignore without affecting our goal.

When all the edges of the v^1-vertex u are examined, we will declared it to be explored.

Repeat the above procedure and extend the tree until either:

- an unmatched v^2-vertex other than the root is found (an augmenting path), or
- there are no more unexplored *even* vertices, in other words, T cannot be extended any more, i.e., T is *blocked*, note that no vertex is added to T more than once.

Now, if T is blocked then T is named a *Hungarian Tree*.

In the next step, we start from another unmatched v^1-vertex, if some one exists, and repeat the above procedure, creating a new alternating path tree. The algorithm terminates when there is no more unmatched v^1-vertices.

First of all, if our tree T is a Hungarian tree, then it is blocked, therefore each alternating path from the root finishes at some even vertex, and the only vertex in T which is unmatched is its root r.

Besides[5], if (u, v) is an edge with u a v^1-vertex such that $u \in T$ and v a v^2-vertex such that $v \notin T$, then u must be labelled odd, otherwise, u is connected to an unmatched vertex (but the only unmatched vertex is the root) or T is extendable through u (but this means that T is not a Hungarian tree).

It follows that no vertex in a Hungarian tree can occur in an augmenting path[6], because, suppose that p is an alternating path that shares at least one vertex with T, there are two ways to share at least one vertex, "entering" T, or "leaving" T, in the first one must be to an odd vertex, the second one, must be also to an odd vertex.

[5]When we say that vertex doesn't belong to the tree, we are doing reference to the actual tree that it is under construction.

[6]Remember that in an augmenting path we found a unmatched v^2-vertex, say v, which is part of the edge of the form (u, v) such that $u \in T$ and $v \notin T$.

Because, like we mention before, our Hungarian tree T starts in an even vertex, the root, and finish from the root at some even vertex, so, we "enter" or "leave" T, to an odd vertex. In both situations go to a contradiction because that means that p is not an alternating path.

Therefore, if in the process of searching for an augmenting path, we found a Hungarian tree we can remove it permanently without have effect in the process.

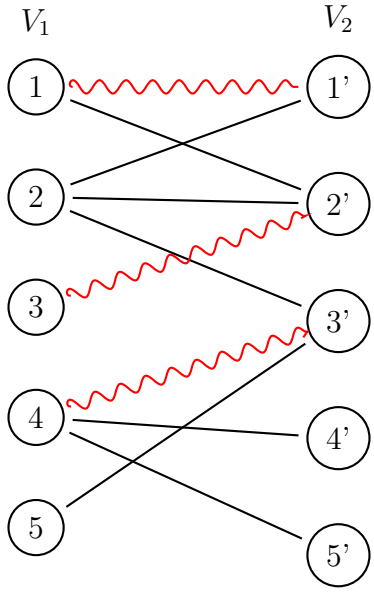
Before continue with the complexity analysis, we mention some assumptions which are that the Breadth-First Search (BFS) procedure assumes that the input graph G is represented using adjacency list, and, of course, it maintains several additional data structures with each vertex in the graph, e.g., distances from the root, the predecessor of a vertex, etc. Also, it is assume that BFS procedure uses a FIFO (First-In,First-Out) Queue to manage the set of discovered vertices.

The running time of the algorithm is computed as follows: the creation of each alternating tree costs $O(|E|)$ times which comes from total time scanning the adjacency list using BFS.[7] Since at most $|V_1| = O(n)$ trees are created, the total running time of the Hungarian algorithm is $O(|V_1||E|) = O(n \cdot m) = O(n^3)$.

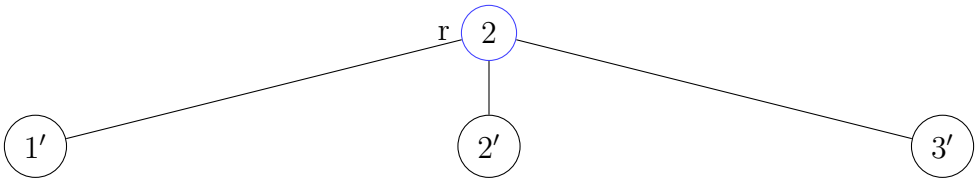
2.2.2 Example

Consider the following bipartite graph $G = (V_1 \cup V_2, E)$, such that $V_1 = \{1, 2, 3, 4, 5\}$ and $V_2 = \{1', 2', 3', 4', 5'\}$, and $E \subseteq V_1 \times V_2$ set of edges represented in the following figure, and consider the edges that belong to the matching M denoted by meandering edges.

[7] Note that no vertex is added to the tree more than once, therefore the total time devoted o queue operations is $O(|V|)$. Thus the well known total running time of BFS is $O(|V| + |E|)$, i.e., it runs on linear time in the size of the adjacency list of G.



We pick some unmatched v^1-vertex, in our example there are two, vertex 2 and 5. We choose 2 like the root of our alternating path tree, and we declare it unexplored and label it even. Afterwards, we add all edges adjacent to the root, i.e., $(u, v) \in E$, where u is a v^1-vertex, at beginning $r = u$, and v is a v^2-vertex. Then we have:



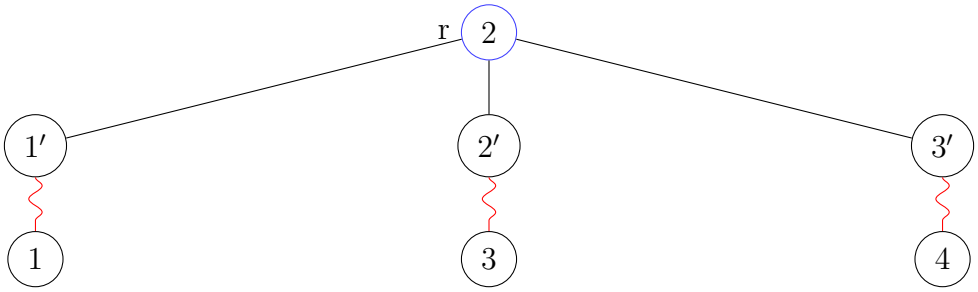
So, the next step will be explored each of $(u, v) \in E$, then we examined all vertices that touch u, that is, $1', 2'$, and $3'$.

- For $(u, 1')$ we have that $1'$ is unlabelled and matched, therefore we add to the tree, say T, the edge that correspond to the matching, i.e., $(1, 1') \in M$. Label 1 even and $1'$ odd, respectively, and declare 1 to be unexplored.

- For $(u, 2')$ we have that $2'$ is unlabelled and matched, therefore we add the edge $(3, 2') \in M$ to T, and label 3 even and $2'$ odd, respectively, and declare 3 unexplored.

- For $(u, 3')$ we proceed similar, adding $(4, 3') \in M$ to T, and labelling 4 even and $3'$ odd, respectively, and declare 4 unexplored.

After examine all this vertices, we declare r to be explored. The following picture show this instance:



Continue with the unexplored even vertices, in our case are $\{1, 3, 4\}$, we choose one such vertex and examine all the edges touching it.

- For 1 we have:

  - $(1, 1')$ where $1'$ is odd and matched, so we do nothing, this edge is exactly the same that was added before but in the other direction.

  - $(1, 2')$, where $2'$ is odd and matched, so we do nothing because, we are in present of an alternative odd length alternating path from an unmatched vertex (i.e., 2) to $2'$. Finally, we declare 1 to be explored.

- For 3 we have:

  - $(3, 2')$, where $2'$ is odd and matched, again we are in present of the same edges in the opposite direction, and we do nothing. Finally, we declare 3 to be explored.

- For 4 we have:

  - $(4, 3')$, where $3'$ is odd and matched, so we do nothing. Idem like previous case.
  - $(4, 4')$, where $4'$ is unlabelled and unmatched, so, here we found an augmenting path. Finally, we label 4 even.
  - $(4, 5')$ observe that this edge was not analyzed by the algorithm, because it has found an augmenting path, therefore, on the one hand, the vertex $5'$ remain unexplored and unlabelled,and it does not belong to the tree, on the other hand, with respect to the vertex $4'$ obviously we do need to explore it because a new tree is going to be created.

This is illustrated in the Figure 2.1.

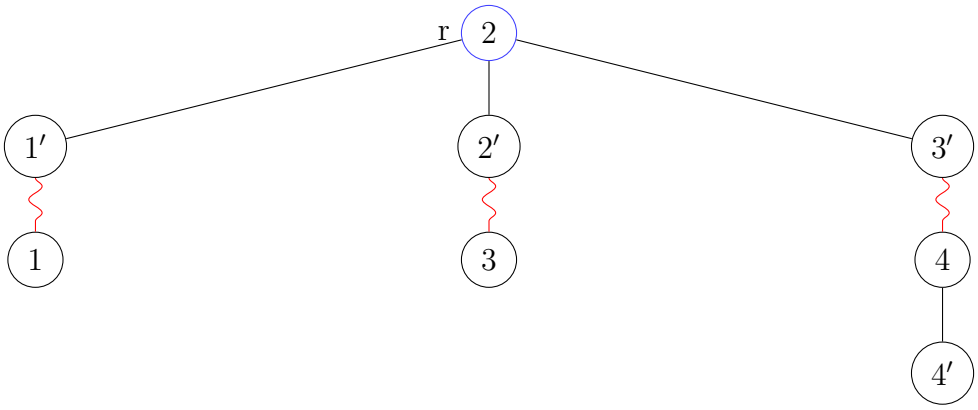


Figure 2.1: In this alternating path tree we found the augmenting path, say p, such that $p = (2, 3'), (3', 4), (4, 4')$ in which we encounter an unmatched v^2-vertex, i.e., $v = 4'$. So, we get an increasing of the current matching by p.

At this point we need to rearrange G, such that we alternate the roles of the edges in p (see Figure 2.1), i.e., matched to unmatched and vice-versa, getting (Figure 2.2):

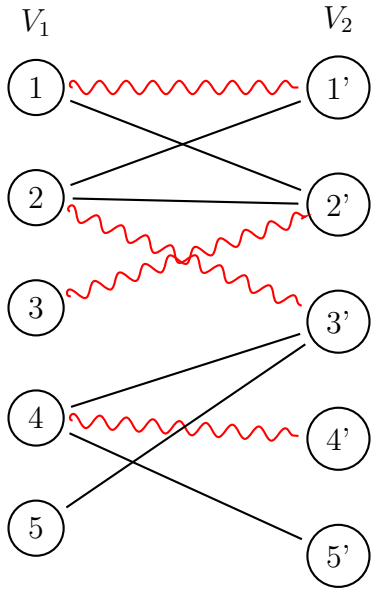


Figure 2.2: Maximum Matching found with Hungarian algorithm.

So, we start with a new tree, in this case choosing the next unmatched v^1-vertex, which is 5. We choose 5 to be the root of our new alternating path tree, and we declare it unexplored and label it even. Afterwards, we add all edges adjacent to the root, i.e., $(u, v) \in E$, where u is a v^1-vertex.

So, we proceed like before, we explore each of $(u, v) \in E$, then we examine all vertices that touch u, in this case we have only one, i.e., $3'$. For $(u, 3')$ we have that $3'$ is unlabelled and matched, so we proceed to add the matched edge to which $3'$ belongs to, that is, $(2, 3') \in M$, and label $3'$ odd and 2 even and unexplored.

Afterwards, we declare $5'$ to be explored and continue with the next even and unexplored v^1-vertex, in our case, 2. Then we explore that vertices that touch 2.

- For $(2, 1')$, where $1'$ is matched and unlabelled, again, we add the matched

edge $(1, 1') \in M$ to the tree and label 1 even and $1'$ odd, and declare 1 to be unexplored.

- For $(2, 2')$, where $2'$ is matched and unlabelled, so, like before, we add $(3, 2') \in M$ to the tree and label 3 even and $2'$ odd, and declare 3 to be unexplored.

- For $(2, 3')$, is just the same edge that was added to the tree, but exploring it in opposite direction, so, $3'$ is odd and matched, then we do nothing.

Finally, we declare 2 explored and continue with the even and unexplored vertices, which are 1 and 3.

But here we found that, from 3 the only edge that touch it is $(3, 2')$ which it was in the tree, and correspond to exploring the edge in the opposite direction, and $2'$ is matched and odd, so we do nothing.

And, with respect to 1, we have an identical situation with the edge $(1, 1')$, so we do nothing, and there is another edge $(1, 2')$ which correspond to an alternative odd length alternating path from the unmatched vertex , i.e., 5 to $2'$, so we do nothing, and also we have that $2'$ is odd and matched, so we do nothing either. Finally, 3 and 1 became explored.

This situation is depicted in the Figure 2.3.

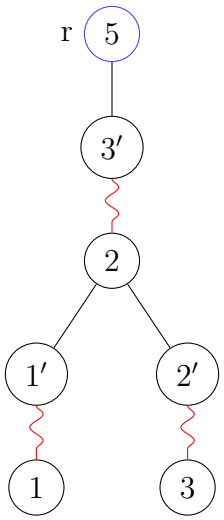


Figure 2.3: The tree corresponds to a Hungarian Tree, because the tree becomes blocked.

Repeating the procedure we get that there are no more unmatched v^1-vertices, therefore the algorithm terminates and the matching of the Figure 2.2 is maximum

2.3 Hopcroft Karp Algorithm

In this section we study the well-known polynomial time algorithm for find a maximum matching in a bipartite graph introduced by John Hopcroft and Richard Karp in 1973 [JEH73]. In the previous section we gave an $O(n^3)$-time Algorithm for finding a maximum matching in bipartite graphs, in this section we show a classical result that improves to $O(n^{5/2})$.

Instead of building the alternating trees (Section 2.2) one at a time, we can use Breadth-First Search (BFS) to simultaneously build alternating trees from all unmatched vertices of V_1. This allows us to find *shortest* augmenting paths. Furthermore, we can find a *maximal* collection of shortest augmenting paths.

The Hopcroft-Karp algorithm [JEH73] work in *phases*. In each phase it constructs a maximal collection of vertex disjoint shortest augmenting paths and uses them to augment the matching.

In this section we shall restrict our attention to finding a maximum matching in bipartite graphs, that is, graphs in which the vertex set can be partitioned into $V = V_1 \cup V_2$, where V_1 and V_2 are disjoint and all the edges in E go between V_1 and V_2, note that this direction is arbitrary because we are working with an undirected graph. Arbitrarily, we will use V_1 to be the set from which we are going to take the unmatched vertices.

We start describing a faster algorithm due to J. Hopcroft and R. Karp, for finding maximum matching in a bipartite graph. The algorithm runs in $O\left(E\sqrt{V}\right)$ time. Given an undirected bipartite graph $G = (V = V_1 \cup V_2, E)$ where all the edges have exactly one endpoint in V_1, let M be a matching in G.

We say that a simple path, i.e., a path with all edges in the path are distinct, P in G is an *augmenting path with respect to* M if it starts at an unmatched vertex in V_1, ends at an unmatched vertex in V_2, and its edges belong alternately to E/M and M. Here paths are treating as a sequence of edges, rather than as a sequence of vertices.

A shortest augmenting path with respect to a matching M is an augmenting path with a minimum number of edges.

The general structure of the algorithm is the following [8]:

Algorithm 1: Hopcroft-Karp (HK)-Algorithm

input : Undirected Bipartite Graph G

output : Maximum Matching M in G

```
1 M ← ∅;                                  /* Initialized with the empty set */
2 repeat
3 |   let P = {P_1, P_2, ..., P_k} be a maximal set of vertex-disjoint shortest
  |   augmenting paths w.r.t M;
4 |   M = M ⊕ (P_1 ∪ P_2 ∪ ··· ∪ P_k);
5 until P == ∅;
6 return M;
```

Before continue with the correctness proof of HK-Algorithm, let us review how it works. The goal is to find a maximum matching in a bipartite graph G. Using Breadth-First Search (BFS) "simultaneously" [9] from all the unmatched vertices of V_1. This allows us to find many paths of the same length with one examination of the set of edges, in particular, find shortest augmenting paths.

Also, Hopcroft and Karp proved that subsequent augmentations—denoted in the algorithm by the symmetric difference—must use larger paths, so the search can be grouped in "phases" finding paths of the same lengths. Then, in particular the HK-Algorithm find a maximal collection of shortest augmenting paths.

The Algorithm works in phases, in each phase the Algorithm construct a maximal collection of vertex-disjoint shortest augmenting paths and uses them to augment the matching. Combining this ideas of simultaneously BFS, and that subsequent

[8] See Definition 2.1.9.

[9] This means that simultaneously search in all the alternating trees, i.e., in an alternating forest.

augmentations requires larger paths, Hopcroft and Karp show that a few phases (in order of $\sqrt{|V|}$) are needed to get a maximum matching in time $O(n^{5/2})$.

Now, we proceed with the Correctness of the Algorithm.

PROOF: **Correctness of HK-Algorithm**

Let M be the result of running HK-Algorithm on a graph G. And let M' be a bigger matching $|M'| \geq |M|$. Then, we proceed as follow

1 - Pick a left vertex $v_1 \in M'$ that is not in M, it exists because M' is bigger.

2 - Let $(v_1, u_1) \in M'$ for some right vertex u_1.

3 - If $u_1 \notin M$, then we found two vertices one in V_1 and other in V_2, but then we found an augmenting pat with respect to M, and so contradiction.

4- If $u_1 \in M$, and so there is a v_2 on the left such that (v_2, u_1) is an edge in M. Then we have two more cases:

a- If $v_2 \notin M'$ then we found $v_1 \in M'$ but not in M, and $v_2 \in M$ but not in M', or

b- If $v_2 \in M'$ then it connects to some u_2 on the right, and repeat the procedure.

5- Eventually, we have a situation like in 4, sooner or later, since otherwise we create an augmenting path w.r.t. M, and get a contradiction.

This proves that HK-Algorithm outputs—if it terminates—a maximum matching.

On the other hand, we have that $|M \oplus P| = |M| + k$, where P is a set of vertex-disjoint augmenting path w.r.t. M, give us a measure of increment, and hence termination.

We show first that if M is a matching and P an augmenting path w.r.t. M then $|M \oplus P| = |M| + 1$.

Consider the following matching $M = \{(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)\}$. Now, consider the following augmenting path $P = \{(a_1, b_1), (b_1, a_2), (a_2, b_2), \ldots, (a_{l+1}, b_{l+1})\}$ such that $(a_i, b_i) \notin M$ and $(b_i, a_{i+1}) \in M$ with $i \in \{1, \ldots, l+1\}$, this is because, for instance, $a_1 \neq u_i$ since u is an unmatched vertex, and $b_1 = v_j$ for some j; and let $|P| = 2l + 1$ for $l \in \mathbb{N}$. So, for $l = 0$ we have $P = \{(a_1, b_1) \notin M\}$, where $l = 1$ we have $P = \{(a_1, b_1) \notin M, (b_1, a_2) \in M, (a_2, b_2) \notin M\}$, and so on until $|P|$, so we have exactly l of these edges which are in M. Then, $|M \oplus P| = |M| - l + (l + 1) = |M| + 1$.

Note that the l factor comes from those edges that belong to $P \cap M$, and the $l + 1$ factor comes from those edges that belong to P but not to M.

Now, if $P_1, P_2, \ldots, P_k$ are vertex-disjoint augmenting paths w.r.t. M, and let $|P_k| = 2l_k + 1$ for $[k]$ the length of each one. Without lost of generality, let $P = \{(a_1, b_1), (b_1, a_2), \ldots, (a_{l+1}, b_{l+1})\}$ for $l \in \mathbb{N}$, and $|P| = 2l + 1$, now we keep (a_1, b_1) and a_1 was free (unmatched) vertex, and b_1 becomes free because we remove (b_1, a_2), afterwards we keep (a_2, b_2), and a_2 is free because $(a_2, b_2) \notin M$, and we remove (b_2, a_3) matching b_2 free as well. Proceeding in this way we get that $|P| = k$ then $M \oplus (P_1, P_2, \ldots, P_k)$ is a matching with cardinality $|M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)| = |M| + k$. Note that l and P are used for the augmenting path result of the k-union. This complete the correctness of HK-Algorithm. □

Therefore, the Hopcroft-Karp algorithm find a maximum matching, if someone exists, such that the termination of the algorithm yields when no more augmenting paths exists. At that point the matching is maximum.

We finish this section enunciating some important properties that are used in

the kernel of the Hopcroft-Karp algorithm, for their proofs consult [Zwi09], and the classical textbooks of algorithms, for instance [CLRS09, AHU74, AHU83].

Claim 2.3.1 *Let P be a shortest augmenting path with respect to M and let P' be an augmenting path with respect to $M \oplus P$. Then $|P'| \geq |P| + 2|P \cap P'|$.*

PROOF: See, for instance, [Zwi09, pg. 6]. □

Claim 2.3.2 *Let G be a bipartite graph and let M be a matching in G. Assume that $P = \{P_1, P_2, \ldots, P_k\}$ is a maximal collection of disjoint shortest augmenting paths with respect to M. Let $M' = M \oplus P_1 \oplus P_2 \oplus \cdots \oplus P_k$ be an augmenting path with respect to M'. Then, $|P'| > |P_1| = |P_2| = \cdots = |P_k|$.*

PROOF: See, for instance, [Zwi09, pg. 6]. □

Theorem 2.3.1 *The Hopcroft-Karp algorithm finds a maximum matching in a bipartite graph after at most $2\sqrt{n}$ phases.*

PROOF: See, for instance, [Zwi09, pg. 6]. □

2.4 Combinatorial Matrix Theory

In this section we intend to show the basic concepts behind the mathematical field called Combinatorial Matrix Theory. The main reference to this field is the book of Richard Brualdi and Herbert J. Ryser entitled *Combinatorial Matrix Theory* [BR91]. Citing Brualdi[10], he wrote *"Combinatorial Matrix Theory is concerned with the use of matrix theory and linear algebra in proving combinatorial theorems and in describing*

[10]Richard A. Brualdi, together with Herb Ryser and R. Craigen, among others, are the pioneers in the field called *Combinatorial Matrix Theory*.

and classifying combinatorial constructions, and it is also concerned with the use of combinatorial ideas and reasoning in the finer analysis of matrices and with intrinsic combinatorial properties of matrix arrays"

Throughout all this thesis we will see different concepts from Combinatorial Matrix Theory, but, in particular we center our attention on a cornerstone result due to König Dénes, named König's Min-Max Theorem, which has its application in many different areas of Mathematics and Computer Science. Combinatorial Matrix Theory is a branch of Mathematics that combines Graph Theory, Combinatorics and Linear Algebra. The Matrix Theory is concerned with combinatorial properties including, for instance, permanents.

Relationship between matrices and graphs:

- knowledge about one of the graphs that can be associated with a matrix is used to illuminate matrix properties and to get better information about the matrix, and
- linear algebraic properties of one of the matrices associated with a graph is used to get useful combinatorial information about the graph.

On one hand, *Combinatorics* is the study of configurations resulting from the discrete combinations of objects and the structure and relationships within, and between, systems of discrete elements.

On the other hand, *Matrix theory* is the study of matrices, which are rectangular arrays of numbers (or other things, like variables, expressions, or elements of arbitrary algebraic systems).

The distinction between Combinatorics and Matrix Theory is sometimes confusing since a matrix can often be viewed as a combinatorial object, called graph.

Combinatorial matrix theory deals with matrices whose entries, or whose submatrix structure, satisfy some combinatorial restraints. Typically, one might study matrices whose elements are 0 or 1, i.e., 0-1 matrices; we also often deal with $(0, 1, -1)$-matrices, $(1, -1)$-matrices or matrices whose elements are taken from a set

$$\{x_1, \ldots, x_n, -x_1, \ldots, -x_n\}$$

where $x_1, \ldots, x_n$ are commuting variables. There are many other variations. We then ask—and try to answer—questions about such matrices that deal with existence structure, classification, relationships, and properties precipitated by the combinatorial restraints.

For example, given two lists of positive integers, does a 0-1 matrix exist whose row sums form the first list and whose column sums form the second list? That is, give a simple condition from which we can determine the answer, for any such pair of lists.

We see now one of the most important theorem in Combinatorial Matrix Theory.

2.4.1 König's Min-Max Theorem

The next it is the fundamental Min-Max Theorem of König [Kön16b, Kön36], this theorem has a long history and many ramifications, some of which are described in Section 2.4.2 where we present different applications.

Given an $m \times n$ matrix A with entries over the Galois field of two elements $GF(2) = \{0, 1\}$, let S be a set of pairs

$$\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$$

where $A_{i_p j_p} = 1$ for every $p \in [k]$, and all the i_p's, as well as all the j_p's, are distinct. In other words, S is a set of positions in the matrix A containing 1s, and no two of those 1s are on the same row or the same column, i.e., no two of them are on the same *line*. Given A, the maximum possible size of such a set S is called the *term rank* of A. Notice that if A_G is the adjacency matrix of a bipartite graph G, then the term rank is in fact the size of the maximum matching in G. Recall that a bipartite graph $G = (V = V_1 \cup V_2, E)$, i.e., a graph where $E \subseteq V_1 \times V_2$, and a matching M is a subset of E consisting of a "pairing" of the vertices of G in such a way that no two edges of M meet at the same vertex.

On the other hand, given a matrix A of size $m \times n$ with entries over the Galois field of two elements $GF(2)$, a set C of lines (i.e., a collection of rows and columns of A) is called a *cover* if every 1 in A is in at least one row or column of C. Then, given a bipartite graph G, the size of the minimum VC corresponds to the minimum cover of A_G. Recall that a *vertex cover* (VC) is a subset C of $V = V_1 \cup V_2$ such that each edge in E has at least one end-point in C.

Theorem 2.4.1 (König's Min-Max) *Let A be a matrix of size $m \times n$ with entries over the Galois field of two elements $GF(2)$. The minimal number of lines in A, that cover all of the 1s in A, is equal to the maximal number of 1s in A, no two of the 1s on a line.*

A detailed Combinatorial Matrix Theory proof of König's Min-Max Theorem is presented in Section 4.2, and an $\exists$**LA**-proof of this Theorem is presented in Section 4.3. Note that König's Min-Max Theorem deals exclusively with properties of 0-1 matrix that remain invariant under arbitrary permutations of lines of the matrix.

2.4.2 Some applications of König's Min-Max Theorem

In this section we are going to present some important applications of König's Min-Max Theorem related to different fields of Discrete Mathematics and Theoretical Computer Science. The importance of this section is to provide some background to Chapter 4 Section 4.3, in which we prove that all the following applications rely on the same Min-Max principle, i.e., the König's Min-Max Theorem. Therefore, some applications are the following:

- *Application on Systems of Distinct Representatives.*

  Let $S_1, S_2, \ldots, S_n$ be n subsets of a given set M. Let D be a set of n elements of M, $D = \{a_1, a_2, \ldots, a_n\}$, such that the elements a_i are all distinct and such that $a_i \in S_i$ for each $i = 1, 2, \ldots, n$. Then D is said to be a *system of distinct representative, or S.D.R.* for the subsets $S_1, S_2, \ldots, S_n$.

  If the sets $S_1, S_2, \ldots, S_n$ have a S.D.R., then any k of the sets must contain between them at least k elements. The converse proposition is the combinatorial Theorem of P. Hall. Let $S_1, S_2, \ldots, S_n$ denote n subsets of a set M. For each $k = 1, 2, \ldots, n$, suppose that every k of these sets contain between them at least k distinct elements of M. Then there exists a S.D.R. for these subsets. (For more details see [Hal87, EW49, HV50]).

- *Applications to the Theory of Partial Order Sets.*

  Let $\mathcal{P}$ be a *finite partially ordered set* or *poset*. We say that $a, b \in \mathcal{P}$ are *comparable elements* if either $a < b$ or $b < a$. A subset C of $\mathcal{P}$ is a *chain* if any two distinct elements of C are comparable. A subset S of $\mathcal{P}$ is an *anti-chain* (also called an *independent set*) if no two elements of S are comparable.

We want to partition a poset into chains; a poset with an anti-chain of size k cannot be partitioned into fewer than k chains, because any two elements of the anti-chain must be in a different partition. Dilworth's Theorem states that the maximum size of an anti-chain equals the minimum number of chains needed to partition $\mathcal{P}$. (For more on Dilworth's Theorem see [Dil50, Per63]).

- *Applications to the Theory of Connectivity.*

  Given a graph $G = (V, E)$, an *x, y-path* in G is a sequence of distinct vertices $v_1, v_2, \ldots, v_n$ such that $x = v_1$ and $y = v_n$ and for all $1 \leq i < n, (v_i, v_{i+1}) \in E$. The vertices $\{v_2, \ldots, v_{n-1}\}$ are called *internal vertices*; we say that two x, y-paths are *internally disjoint* if they do not have internal vertices in common.

  Given two distinct vertices $x, y \in V$, we say that $S \subseteq E$ is an *x, y-cut* if there is no path from x to y in the graph $G' = (V, E - S)$. Let $\kappa(x, y)$ represent the size of the smallest x, y-cut, and let $\lambda(x, y)$ represent the size of the largest set of pairwise internally disjoint x, y-paths.

  Menger's Theorem states that for any graph $G = (V, E)$, if $x, y \in V$ and $(x, y) \notin E$, then the minimum size of an x, y-cut equals the maximum number of pairwise internally disjoint x, y-paths. That is, $\kappa(x, y) = \lambda(x, y)$. For more details on Menger's Theorem turn to [Meng27, Go00, Pym96].

  Menger's Theorem, first proved in 1927, turns out to be one of the many consequences of a fundamental theorem about flows in directed graphs, the well known *Max-Flow Min-Cut Theorem*, that is, Menger's Theorem is Min-Cut Max-Flow Theorem where all edges have capacity 1.

- *More applications.*

Finally, there exists more applications of König's Min-Max Theorem, we mention the following applications here, not only because these applications rely on the same Min-Max principle, but also by their importance on Computer Science field; we have application to the *Theory of Permanents* which it is a important field not only in Mathematics but also in Computer Science; another application is to the *Theory of Doubly Stochastic Matrices*, those matrices have the characteristic that are non-negative $n \times n$ matrices in which all lines (rows and columns) sums are equal to 1. For more details about applications of König's Mini- Max Theorem see [BR91].

Chapter 3

Permutation-Based Algorithm

In this chapter we present a new Permutation-Based Algorithm for computing a Minimum Vertex Cover from a Maximum Matching in a bipartite graph. Sometimes, we refer to the Permutation-Based Algorithm as "our algorithm". So, our algorithm is linear-time and computationally very simple: it permutes the rows and columns of the matrix representation of the bipartite graph in order to extract the vertex cover from a maximum matching in a recursive fashion. Besides, our algorithm uses properties of König's Min-Max Theorem and it is interesting for providing a new permutation perspective on a well-known problem.

König's famous Min-Max Theorem ([Kön16b, Kön36]) assures that the existence of a maximum matching of size ρ is equivalent to the existence of a minimum vertex cover of size ρ. On the other hand, the Hopcroft-Karp algorithm computes maximum matchings in bipartite graphs in polynomial time.

These two facts yield a natural way of computing vertex covers using standard reductions from search to decision problems that work in time $O(|V|^{\frac{3}{2}}|E|)$. (See Algorithm 1).

On the other hand, in this chapter we show how rich is the field of Combinatorial Matrix Theory from an algorithmic point of view, insomuch in our algorithm we make use, purely, of combinatorial matrix properties to compute the Minimum Vertex Cover.

Unless otherwise specified, we assume that our matrices are over $GF(2)$, that is, the Galois field of two elements $\{0, 1\}$.

Sometimes, we shall apply function $f : M_{m\times n}(GF(2)) \longrightarrow \mathbb{Z}$, that is, function from 0-1 matrices of size $m \times n$ to integers. We require the integers as one of our fundamental operations will be counting the number of 1s in a 0-1 matrix, i.e., compute $f := \Sigma A$, the sum of all the entries of A. In fact, we will just do Boolean Matrices since we are interested in combinatorial properties more than arithmetical ones.

It does not matter if we view matrices as (1) $GF(2)$, (2) $\{0, 1\}$ over $\mathbb{Z}$, (3) Boolean matrices. Since we are only concerned with patterns of 0s and 1s, any of these 3 formalisms will do—however, given $\Sigma(A)$ equals the number of 1s in A, $\{0, 1\}$ over $\mathbb{Z}$ is the most natural view.

3.1 Introduction

Suppose that we are given a bipartite graph $G = (V = V_1 \cup V_2, E)$, i.e., a graph where $E \subseteq V_1 \times V_2$. Let A_G be the adjacency matrix of G, of size $|V_1| \times |V_2|$, and with entries over the Galois field of two elements $GF(2) = \{0, 1\}$. Thus, $(i, j) \in E$ if and only if $(A_G)_{ij} = 1$. A matching $\mathcal{M}$ is a subset of E consisting of a "pairing" of the vertices of G in such a way that no two edges of $\mathcal{M}$ meet at the same vertex. Again, we can represent a matching as a set of pairs of nodes of V, i.e., $\mathcal{M} \subseteq E$, or as an adjacency matrix. A matching is maximum if $|\mathcal{M}|$ is as large as possible. We talk of

bipartite graphs and their adjacency matrix representations interchangeably. We use script $\mathcal{M}$ in order to distinguish it—set representation—from its matrix representation denoted with M_A, or just M.

It is well known that given a general graph, its maximum matching can be computed with Edmond's blossom shrinking Algorithm in polynomial time [Edm65]. For bipartite graphs we can compute a maximum matching slightly faster using the Hopcroft-Karp algorithm (HK-Algorithm)—see Chapter 2 Section 2.3. On the other hand, while minimum vertex covers can be computed in polytime for bipartite graph, for general graphs it is an **NP**-hard problem.

Let $M_A = \mathrm{HK}(A)$ be the output of running the HK-Algorithm on A, i.e., M_A is the adjacency matrix of a maximum matching produced by the HK-Algorithm.

The dual of a maximum matching for a bipartite graph is the graph's Minimum Vertex Cover. Given a bipartite graph $G = (V = V_1 \cup V_2, E)$, a *vertex cover* (VC) is a subset C of $V = V_1 \cup V_2$ such that each edge in E has at least one end-point in C. A VC C is minimum if $|C|$ is as small as possible. In this chapter we present an algorithm that on input $\langle A, M_A \rangle$ produces the minimum VC of A. Our algorithm relies on König's famous Min-Max theorem—for a combinatorial matrix theory proof of this theorem see, for example, Chapter 4 Section 4.2 or [BR91], and for a Π_2^B-Inductive proof in **LA**-Theory see Appendix A.1, recall that Π_2^B-Induction does not yield feasible proofs. We find it useful to give two equivalent formulations of König's Min-Max Theorem.

Theorem 3.1.1 (König's Min-Max version I) *Given a bipartite graph G, if ρ_G is the size of the maximum matching of G, and ρ'_G is the size of the minimum VC of G, then $\rho_G = \rho'_G$.*

Given an $m \times n$ matrix A with entries over $GF(2)$, let S be a set of pairs

$$\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$$

where $A_{i_p j_p} = 1$ for every $p \in [k]$, and all the i_p's, as well as all the j_p's, are distinct. In other words, S is a set of positions in the matrix A containing 1s, and no two of those 1s are on the same row or the same column, i.e., no two of them are on the same *line*. Given A, the maximum possible size of such a set S is called the *term rank* of A. Notice that if A_G is the adjacency matrix of a bipartite graph G, then the term rank is in fact the size of the maximum matching in G.

On the other hand, given a matrix A of size $m \times n$ with entries over $GF(2)$, a set C of lines (i.e., a collection of rows and columns of A) is called a *cover* if every 1 in A is in at least one row or column of C. Then, given a bipartite graph G, the size of the minimum VC corresponds to the minimum cover of A_G.

Theorem 3.1.2 (König's Min-Max version II) *Let A be a matrix of size $m \times n$ with entries over $GF(2)$. The minimum number of lines in A, that cover all of the 1s in A, is equal to the maximum number of 1s in A, no two of the 1s on a line.*

Therefore, we can use the HK-Algorithm to also compute the size of the minimum vertex cover of G: let $M_{A_G} = \mathrm{HK}(A_G)$, and ρ_G is given by the number of 1s in M_{A_G} (let's denote that by $|M_{A_G}|$). Thus, we know from the Min-Max theorem that the size of the smallest VC of G is also given by $|M_{A_G}| = \rho_G = \rho'_G$. Once we know the size of the smallest VC, we can also compute the actual VC using a reduction from a search to a decision problem; see the Algorithm 1. Our Permutation-Based Algorithm is faster, especially when there are many edges in G.

As a "Proof Complexity" aside, note that König's Min-Max Theorem has many other equivalent formulations and we later show that these equivalences can be proven in low complexity (Chapter 4); note that this means that our proof of equivalences is such that in each step it uses only feasible—polytime reasoning.

Some of these "reformulations" are as follows: *Menger's Theorem* (Section 4.3.5.1), counting disjoint paths; *Hall's Theorem* (Section 4.3.5.2), giving necessary and sufficient conditions for the existence of a "System of Distinct Representatives" of a collection of sets (SDR); *Dilworth's Theorem* (Section 4.3.5.3), counting the number of disjoint chains in a poset. (For more details see Chapter 4, Section 4.3.5).

3.1.1 Background and context

As is well known, vertex cover (VC) is an **NP**-complete problem for general graphs, and a polynomial time for bipartite graphs. We mentioned that the HK-Algorithm computes a Maximum Matching for a given bipartite graph in polynomial time—specifically, in time $O(\sqrt{|V|}|E|)$, where $G = (V = V_1 \cup V_2, E)$. And by König's Min-Max Theorem, we know that a bipartite graph G has a maximum matching of size k if and only if it has a minimum VC of size k. Putting all those elements together, we can use the HK-Algorithm for maximum matchings in order to compute a minimum vertex cover in a bipartite graph.

We use the following notation in algorithms, the end of line is denote by ";", the $Z \leftarrow \emptyset$ denote initialization statement, and sometime, we use an alternative notation $Z = 0$ denoting assignment statement, also, we use $Z == 0$ to denote testing equality condition statement. Finally, we use $|\cdot|$ to denote the cardinality of a set.

Algorithm 1: On input $G = (V = V_1 \cup V_2, E)$, the algorithm, by repeatedly invoking the Hopcroft-Karp algorithm, computes a minimum vertex cover C.

input : $G = (V = V_1 \cup V_2, E)$

output : minimum Vertex Cover C

```
1  C ← ∅;
2  k ← |HK(G)|;                    /* k is the size of minimum VC of G */
3  for every node u ∈ V = V1 ∪ V2 do
4      if u ∈ Vi then
5          Create new G' = (V', E') from G by:
6          V' ← {u'1, u'2} ∪ V3−i;
7          E' ← {(u, u'1), (u, u'2)} ∪ E;
8      k' ← HK (G' = (V', E'));
9      if k == k' then
10         C ← {u} ∪ C;
11         delete from G all edges adjacent on u;
12         delete from G all singleton nodes;
13     k ← k − 1;
```

Observe that in lines 6 and 7, we add two new nodes u'_1, u'_2 to V_{3-i} to obtain V', and add two new edges (u, u'_1), (u, u'_2) to E to obtain E', getting in that way the new graph G'.

Lemma 3.1.1 *The procedure described in the Algorithm 1 works correctly and runs in time* $O(|V|^{\frac{3}{2}}|E|)$.

PROOF: If we add two new edges (u, u'_1) and (u, u'_2) to G—and obtain G'—the

"cheapest" way to cover those two new edges is with their common vertex u. That is, by adding the gadget $(u, u'_1), (u, u'_2)$, we force u to be part of a minimum cover of the resulting graph.

If there was a cover of G that included u, then the same cover works for G'; essentially, we cover the two new edges "for free." This corresponds to the case $k = k'$, where we know that u was part of a cover, and so we add it to C, we delete from G the edges adjacent on u and we delete all singleton nodes.

If, on the other hand, $k < k'$, then no minimum cover of G contained u, and thus we needed to add u to the cover of G' in order to take care of the two new edges; in this case we do not put u in C.

We repeat the same procedure on G (with the edges adjacent on u removed in case $k = k'$, and G unaltered otherwise) on the next vertex in $V = V_1 \cup V_2$, each time running the HK-Algorithm, giving the stated running time bound of $O(\sqrt{|V|}|E||V|)$. Note that it is immaterial in which order we examine the nodes of G; any ordering works. □

Note that the reduction described in Lemma 3.1.1 would not work over general graphs (i.e., not necessarily bipartite). First, for the obvious technical reason that requires u'_1, u'_2 to be added to " the other vertex set," i.e., to V_{3-i} if $u \in V_i$, where $i = 1, 2$. But, more importantly, say that instead of the Hopcroft-Karp algorithm we invoke Edmond's blossom shrinking algorithm [Edm65] that works over general graphs. Could we then modify the Algorithm 1 somehow to make it work over general graphs? The answer is: "not in polynomial time, unless **P=NP**". The reduction given by the Algorithm 1 relies deeply on the graph being bipartite.

It is not difficult to see (and we show it in Lemma 3.2.1 below) that given a

maximum matching $\mathcal{M}$ of a bipartite G, the minimum VC C can be constructed by taking, for each $e \in \mathcal{M}$, one end point node. Recall that we use script $\mathcal{M}$ in order to distinguish its set representation from its matrix representation denoted with M. Of course, not all selections of end-points works, but at least one selection of end-points works. Our algorithm in Section 3.3 works directly on a given maximum matching, not rerunning the HK-Algorithm. Our algorithm is therefore much faster when the number of edges is large (in the order of $|E| >> \sqrt{|V|}$), as its running time is $O(|V|^2)$.

3.2 Preliminaries to our algorithm

We start with terminology for denoting lines: given a matrix A, we can denote the lines as $r_1, r_2, \ldots, r_m$ and $c_1, c_2, \ldots, c_n$ where A is $m \times n$, and the r's denote the rows and the c's denote the columns. It will also be advantageous to denote lines with the following terminology: $l^o_{(i,j)}$ where $o \in \{0, 1\}$, and $l^o_{(i,j)}$ denotes a line going through entry i, j, where $i \in [m]$ and $j \in [n]$, and

$$o = \begin{cases} 0 & l^o_{(i,j)} \text{ is vertical, i.e., } l^o_{(i,j)} = c_j \\ 1 & l^o_{(i,j)} \text{ is horizontal, i.e., } l^o_{(i,j)} = r_i \end{cases}$$

Then, a cover is a set of lines $C_A^{\vec{o},\vec{i},\vec{j}} = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\}$, with orientation $\vec{o} = o_1 o_2 \ldots o_k$, and $\vec{i} = i_1, i_2, \ldots, i_k$, $\vec{j} = j_1, j_2, \ldots, j_k$, and it is such that any 1 in A is covered by one of these lines; i.e., if there is a 1 in position (i, j) of the matrix A, then there exists a $p \in [k]$, such that $l^{o_p}_{(i_p,j_p)} \in C_A^{\vec{o},\vec{i},\vec{j}}$ and

$$[i = i_p \wedge o_p = 0] \vee [j = j_p \wedge o_p = 1].$$

Thus, as was pointed out in the last paragraph of the previous section, if M_A is a maximum matching, the Min-Max Theorem tells us that there exists an $\vec{o} \in \{0,1\}^k$ such that $C_A^{\vec{o},\vec{i},\vec{j}}$, where $k = |M_A|$. Recall that M_A represents a maximum matching as a matrix with entries over $GF(2)$, and that a 1 in position (i,j) means that (i,j) is an edge in the matching (i.e., $i \in V_1$ and $j \in V_2$ are "paired"). But in terms of "Matrix Combinatorics" this means that the 1s in M_A are positioned in such a way that no two 1s are on the same line (vertical or horizontal). Thus, we know that whatever $\vec{o}$ is, $C_A^{\vec{o},\vec{i},\vec{j}}$ is such that it must have lines through all the 1s of M_A; further, any such line cannot cover more than a single 1, and since we know that the size of $C_A^{\vec{o},\vec{i},\vec{j}}$ is the size of M_A, each 1 of M_A claims exactly one line.

Lemma 3.2.1 *Suppose that $G = (V = V_1 \cup V_2, E)$ is a bipartite graph, and let A be its adjacency matrix, and M_A a maximum matching. Suppose*

$$M_A = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$$

i.e., M_A is a list of all the positions of M_A with a 1 in them ($k = |M_A|$). Then, it must be the case that

$$C_A^{\vec{o},\vec{i},\vec{j}} = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\}.$$

is a cover for some $\vec{o} \in \{0,1\}^k$.

PROOF: We know that for all $p \in [k]$, $A_{(i_p j_p)} = 1$, and so our cover must contain, for every $p \in [k]$, either r_{i_p} or c_{j_p}. By the Min-Max Theorem, there is a cover of size k, and so, by the pigeonhole principle, we can say something stronger: our cover *must consist*, for every $p \in [k]$, of either r_{i_p} or c_{j_p}.

But that is the same as saying that our cover *must consist*, for every $p \in [k]$, of $l^{o_p}_{(i_p,j_p)}$, for $o_p = 0$ or $o_p = 1$. The Lemma follows from that. □

Recapitulating, given $\langle A, M_A \rangle$, in order to compute $C_A^{\vec{o},\vec{i},\vec{j}}$, all we need to compute are the orientations: $\vec{o} = o_1 o_2 \dots o_k$, since the (i_p, j_p)'s are imposed by M_A.

Our algorithm works with permutations, and we adopt the following notation to describe permutations; as bijections:

$$\pi : [m] \longrightarrow [m]$$
$$\tau : [n] \longrightarrow [n]$$

and permutation matrices; that is, P_π and Q_τ permute the rows and columns, respectively, of M_A. The matrix P_π is obtained from the identity matrix by exchanging the rows according to π, and the matrix Q_τ is obtained from the identity matrix by exchanging the rows according to τ. Then the relationship is as follows: $(P_\pi M_A Q_\tau)_{ij} = (M_A)_{\pi^{-1}(i)\tau^{-1}(j)}$.

Our permutations P, Q work in such a way that they place the 1s on the main diagonal in the original order of the rows; that is, if the 1s of M_A where in positions:

$$(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)$$

with $i_1 < i_2 < \cdots < i_k$, then our permutations π, τ are given as follows:

$$\begin{array}{rcl@{\qquad\qquad}rcl} i_1 & \mapsto & 1 & j_1 & \mapsto & 1 \\ i_2 & \mapsto & 2 & j_2 & \mapsto & 2 \\ & \vdots & & & \vdots & \\ i_k & \mapsto & k & j_k & \mapsto & k \end{array}$$

We call such permutations *order preserving* (according to rows).

Observe that we have the following invariant.

Lemma 3.2.2 *Suppose that π, τ are order preserving permutations. Then, if*

$$C_A^{\vec{o},\vec{i},\vec{j}} = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\}$$

is a covering of A, then

$$C_{P_\pi A Q_\tau}^{\vec{o},\pi(\vec{i}),\tau(\vec{j})} = \{l^{o_1}_{(\pi(i_1),\tau(j_1))}, l^{o_2}_{(\pi(i_2),\tau(j_2))}, \ldots, l^{o_k}_{(\pi(i_k),\tau(j_k))}\}$$

is a covering of $P_\pi A Q_\tau$.

Proof: Suppose that $C_A^{\vec{o},\vec{i},\vec{j}} = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\}$ is indeed a covering of A. Consider any entry (p,q) of $P_\pi A Q_\tau$, i.e., $(P_\pi A Q_\tau)_{pq} = A_{\pi^{-1}(p)\tau^{-1}(q)}$. If $A_{\pi^{-1}(p)\tau^{-1}(q)} = 1$, then either $r_{\pi^{-1}(p)} \in C_A^{\vec{o},\vec{i},\vec{j}}$ or $c_{\tau^{-1}(q)} \in C_A^{\vec{o},\vec{i},\vec{j}}$. This last statement means that there exists an $a \in [k]$ such that at least one of the following two statements is true:

- $l^{o_a}_{(i_a,j_a)} \in C_A^{\vec{o},\vec{i},\vec{j}}$ where $i_a = \pi^{-1}(p) \wedge o_a = 1$, or
- $l^{o_a}_{(i_a,j_a)} \in C_A^{\vec{o},\vec{i},\vec{j}}$ where $j_a = \tau^{-1}(q) \wedge o_a = 0$,

which in turn means that at least one of the following is true

- $l^{o_a}_{(\pi(i_a),\tau(j_a))} \in C_{P_\pi A Q_\tau}^{\vec{o},\pi(\vec{i}),\tau(\vec{j})}$ where $\pi(i_a) = \pi(\pi^{-1}(p)) \wedge o_a = 1$, or
- $l^{o_a}_{(\pi(i_a),\tau(j_a))} \in C_{P_\pi A Q_\tau}^{\vec{o},\pi(\vec{i}),\tau(\vec{j})}$ where $\tau(j_a) = \tau(\tau^{-1}(q)) \wedge o_a = 0$,

and as π, τ are permutations, they are bijections, and so $\pi(\pi^{-1}(p)) = p$ and $\tau(\tau^{-1}(q)) = q$, and so restating once again we obtain:

- $l^1_{(p,\tau(j_a))} \in C_{P_\pi A Q_\tau}^{\vec{o},\pi(\vec{i}),\tau(\vec{j})}$, or

- $l^0_{(\pi(i_a),q)} \in C^{\vec{o},\pi(\vec{i}),\tau(\vec{j})}_{P_\pi A Q_\tau}$.

In either case, this means that there is a line covering entry (p, q) of $P_\pi A Q_\tau$ if that entry is a 1. Hence, $C^{\vec{o},\pi(\vec{i}),\tau(\vec{j})}_{P_\pi A Q_\tau}$ is indeed a covering for $P_\pi A Q_\tau$. □

It is this lemma that ensures that our algorithm, while permuting matrices, in an order preserving way, it computes the correct orientations of the original matrix. However, permuting a matrix, even if the permutations are *not* order preserving, does not destroy its covering properties; we only must account for the permutation in the orientations. We state it in the following corollary.

Corollary 3.2.1 *A general permutation of the 1s in M_A, where we place them on the diagonal of a contiguous block as in Lemma 3.2.1, but we also* reorder *them according to some permutation μ still yields a corresponding covering where we account for μ as follows:*

$$C^{\mu(\vec{o}),\pi(\vec{i}),\tau(\vec{j})}_{R_\mu P_\pi A Q_\tau R_\mu} = \{l^{o_{\mu(1)}}_{(\mu(\pi(i_1)),\mu(\tau(j_1)))}, l^{o_{\mu(2)}}_{(\mu(\pi(i_2)),\mu(\tau(j_2)))}, \ldots, l^{o_{\mu(k)}}_{(\mu(\pi(i_k)),\mu(\tau(j_k)))}\}.$$

The point of these rather technical corollary is to show that we can preserve coverings under permutations; that is, we can permute M_A at will in our algorithm, and recover the lines easily. (For more details see Section A.3.1.4).

3.3 Permutation-Based Algorithm

On input $\langle A, M_A\rangle$, our algorithm computes $C^{\vec{o},\vec{i},\vec{j}}_A$. More precisely, as was shown in Lemma 3.2.1, given M_A we know *a priori* that

$$C^{\vec{o},\vec{i},\vec{j}}_A = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\},$$

where the (i_p, j_p) are the non-zero entries of M_A. Hence, all that we need to compute in our algorithm is the orientation vector $\vec{o} = o_1 o_2 \dots o_k$. The analogy of this in the graph theoretic setting is that given a bipartite graph G, and a maximum matching M, the minimum vertex cover can be selected from M by choosing for each edge $e \in M$, one of its end-points; a particular choice of end-points corresponds to a particular orientation. We now present the algorithm for selecting a particular orientation that works.

Input: A, M_A, $m \times n$ matrices, with entries over $GF(2)$, where M_A is a maximum matching for A, and A is the adjacency matrix of a bipartite graph G:

Step 1 If $k = |M_A| = \min\{m, n\}$, then

- $\{r_1, r_2, \dots, r_m\}$ is a cover if $m \leq n$; i.e., return $\vec{o} = 1^m$ and exit
- $\{c_1, c_2, \dots, c_n\}$ is a cover if $m > n$; i.e., return $\vec{o} = 0^n$ and exit

Step 2 Else, $k = |M_A| < \min\{m, n\}$, and we let P, Q be two permutation matrices (P is $m \times m$ and Q is $n \times n$) such that P is order preserving, $\text{diag}(E) = \text{diag}(I_k)$, where:

$$PAQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0_{(m-k)\times(n-k)} \end{array}\right] = \left[\begin{array}{ccc|c} 1 & & & \\ & 1 & & \\ & & \ddots & A_1 \\ & & & 1 \\ \hline & A_2 & & 0 \end{array}\right]$$

That is, the 1s corresponding to M_A are all permuted to be on the diagonal of the upper-left $k \times k$ quadrant; call this quadrant E. The first thing to observe is

that the lower-right $(m-k) \times (n-k)$ quadrant consists entirely of zeros. This assertion is a consequence of König's Min-Max Theorem: all the lines in $C_A^{\vec{o},\vec{i},\vec{j}}$ pass through a 1 in E; none of these lines can possibly touch this lower-right quadrant, so it must be full of zeros.

Step 2a Suppose that $A_1 = 0 \vee A_2 = 0$ (note that A_1 is $k \times (n-k)$ while A_2 is $(m-k) \times k$).

- If $A_1 = 0$ then return $\vec{o} = 0^k$ and exit
- If $A_2 = 0$ then return $\vec{o} = 1^k$ and exit

Step 2b Else, $A_1 \neq 0 \wedge A_2 \neq 0$. The 1s on the diagonal of E are grouped into two sets of sizes k_1 and k_2, respectively, with $k = k_1 + k_2$.

The first group, the black 1s, have the property that both row i of A_1 and column i of A_2 have no 1s in them. That is, the white portion of the upper-right quadrant in Figure 4.2, and the white portion of the lower-left quadrant in Figure 4.2, contain only 0s.

This situation, as represented in Figure 4.2, is simplified for the sake of clarity: the black 1s and the green 1s are depicted as two separate groups, but in general they are interspersed. We could block them together to be as in Figure 4.2, but that would require in general a permutation that is not order preserving (see Section A.3.1.4); this could still be done by Corollary 3.2.1, but it introduces a technical overhead, as the orientations would no longer match (but we could recover the original orientations by inverting the permutation).

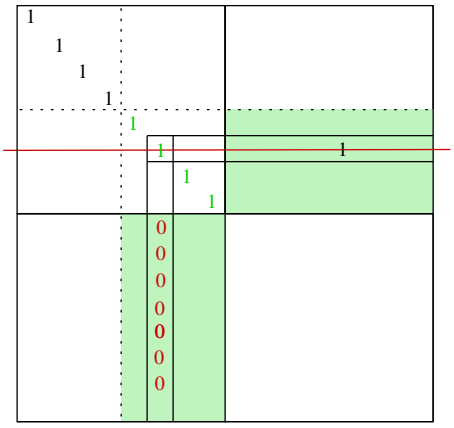


Figure 3.1: Step 2b

Note that under the assumption $A_1 \neq 0 \wedge A_2 \neq 0$, we know that $k_2 > 0$, so we know that not all 1s in the upper-left quadrant are black; but it may well be the case that none are black, i.e., all the 1s are green, which would correspond to $k = k_2$.

We now compute the orientations of the lines going through the green 1s. Note that each green 1 can claim at most one line (by Lemma 3.2.1).

For each green 1 in position (j, j):

- If there is a 1 in row j of A_1, then we let $o_j = 1$.
- Else, we let $o_j = 0$.

We know that it is not possible for both row j of A_1 to have a 1, and column j of the A_2 to have a 1, since that would require two lines through (j, j), which is not possible by Lemma 3.2.1.

Further, the square that encloses the green 1s must be successfully covered by the above scheme: we have no choice as to the orientation of the lines covering

the green 1s, and by the Min-Max Theorem a successful covering exists, and thus the covering imposed by the green portions of A_1 and A_2 must necessarily work for the square enclosing the green 1s. Again, this argument is formalized in Lemma 3.2.1.

We must now compute the orientations of the lines covering the black 1s.

We do so recursively, by repeating the procedure from **Step 2** with the matrix obtained from the $k \times k$ upper-left quadrant of $P_\pi A Q_\tau$, i.e., the upper-left quadrant of the matrix in Figure 3.2, but with the following modifications:

> the square enclosing the green 1s, represented in dark green, is zeroed out—as by the above argument and Lemma 3.2.1, it is successfully covered with the lines oriented by the green 1s —and we also place zeros wherever those lines crossed the rest of the quadrant. In Figure 3.2, we place zeros in the upper-left quadrant wherever the horizontal red lines crosses.

End of algorithm

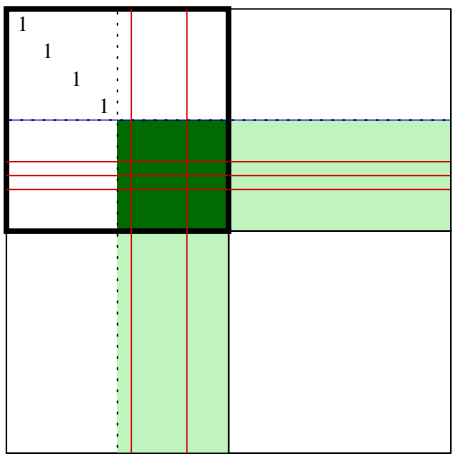


Figure 3.2: Repeat Step 2 with the upper-left quadrant, emphasized with a thicker border, with the "green square" zeroed out, as well as the entries under the red lines, arising from the cover of the "green square," zeroed out.

Conceptually, the algorithm is rather simple. The technical complication is the permutations.

We are computing the orientation of a covering for a permuted version of A; then, we must "extract" the correct orientation for the original version of A. This is what introduces a certain technical overhead. On the other hand, these permutations help to maintain a simple data structure (reconfigurations of the $|V_1| \times |V_2|$ matrix) that is essential for the computation.

As far as the complexity of this procedure, the loop is the repetition of **Step 2**; more precisely, we need to compute the orientations of the black 1s. As was mentioned above, if there are no green 1s, then the procedure terminates (outputting all horizontal or or vertical orientations, according to which one of A_1 or A_2 is all zero). Thus, if $k_2 = 0$, we are done.

Otherwise, $k_2 > 0$, and the number of black 1s decreases by at least 1. Thus this loop, in the worst case, can repeat at most

$$k = |M_A| \leq \min\{m, n\} = \min\{|V_1|, |V_2|\}$$

many times. Inside each loop, we scan the entries of the matrix looking for zeros, which can be done in time proportional to the size, i.e., $m \times n$, and we compute the permutations π, τ directly from M_A (from the positions of its 1s, so they are particularly easy to compute if M_A is given as a list of positions of 1s) again in time $(|V_1| + |V_2|)$. (For more precise treatment see next Section 3.4).

We finish with a short discussion of how to compute the permutation matrices P and Q that arise in the algorithm (for more details see Section A.3.1.1 and Section A.3.1.2). These matrices are computed from the permutations π and τ, or rather, they are the natural matrix representations of π and τ. Let n be the size of A, i.e., A is $n \times n$. Let $i_1, i_2, \ldots, i_k$ be the non-zero rows of M_A such that $i_1 < i_2 < \ldots < i_k$. Let $\{j_1, j_2, \ldots, j_{n-k}\}$ be the remaining rows, also ordered; thus, row j_p has only zeros in it.

The important thing, as explained in the algorithm, is that π and τ have to be order preserving, and in order to maintain this property, we initialize $r = 1$ and $q = 1$, and two integer arrays i, j of size n. Now for every $p = 1 \ldots n$, if row p of M_A is not zero, we let $q = q + 1$ and let $i[q] = p$. On the other hand, if row p of M_A is zero, we let $j[r] = p$, and let $r = r + 1$.

We now construct π from the two arrays i and j which encode the following mapping:

$$\begin{aligned} i[1] &\mapsto 1 \\ i[2] &\mapsto 2 \\ &\vdots \\ i[k] &\mapsto k \\ j[1] &\mapsto k+1 \\ j[2] &\mapsto k+2 \\ &\vdots \\ j[n-k] &\mapsto n \end{aligned}$$

From π we construct P as follows: P has 1s in positions

$$(1, i[1]), (2, i[2]), \ldots, (k, i[k]), (k+1, j[1]), (k+2, j[2]), \ldots, (n, j[n-k])$$

and zeros everywhere else. The permutation matrix Q is constructed in a similar manner from τ.

We summarize it in the following theorem.

Theorem 3.3.1 *Given a bipartite graph $G = (V = V_1 \cup V_2, E)$, we obtain a procedure for computing a Minimum Vertex Cover from a Maximum Matching that runs in time $|\langle A, M_A \rangle| = |V|^2$. That is, given a 0-1 matrix and the corresponding maximum matching, we can compute the vertex cover in linear time (in the size of the input).*

Corollary 3.3.1 *The correctness—partial correctness and termination—of our algorithm can be formalized in* $\exists$**LA**-*Theory (or perhaps even* **LA**-*Theory).*

PROOF: See Appendix A.3. □

3.4 Complexity Analysis

3.4.1 Complexity

In this section we investigate more closely the complexity of our algorithm, i.e., our Permutation-Based Algorithm (Section 3.3). We divide this section into two parts, the first one consist of the high level complexity analysis description of our algorithm, and the second one is a more tight complexity analysis of our algorithm.

Recall that our algorithm works from a perfect matching already in place, from these complexity analyze we get that the complexity of our algorithm is at most $O(n^2)$, where $n = |V|$, i.e., it is the cardinality of the set of vertices of a given bipartite graph G.

Our complexity measure is informal—like we will show—we measure number of steps of our algorithm. Finally, in Section 3.4.2 we give a very short discussion about different data structures concerning to the implementation of our algorithm.

3.4.1.1 Introduction

Let $G = (V = V_1 \cup V_2, E)$ be an undirected bipartite graph. Let $|V_1| = |V_2| = n$ be the number of vertices and let $|E| \subseteq V_1 \times V_2$ such that $|E| \leq n^2$.

Also, we use the matrix representation of the G by its adjacency matrix A_G, and we know that A_G is symmetric with respect to the main diagonal. As a "data

structure" aside, note that using this structure on matrices we are going to need to represent our matrices $\frac{1}{2}n\,(n+1)$ bits. We will do use of A_G and A indistinguishably when the context be clear. Also, from the correctness of HK-Algorithm—Chapter 2 Section 2.3—we know that the matching $\mathcal{M}$ is maximum and it is, for instance, of size k, that is $|\mathcal{M}| = k$. Recall that we use script $\mathcal{M}$ in order to distinguish its set representation from its matrix representation denoted with M_A, or just M.

On the other hand, our algorithm works using two matrices with entries over $GF(2)$, but we are going to do a padding process before that our Permutation-Based Algorithm use them. So, the two matrices are:

1- The matrix representation of the maximum matching $\mathcal{M}$ plus a padding with zeros rows and/or columns, if necessary, that is, for example, suppose that the maximum matching output of HK-Algorithm is of size 2, say $\mathcal{M} = \{(1, 2'), (3, 4')\}$ but our bipartite graph is $G = (V_1 \cup V_2, E)$ given by $V_1 = \{1, 2, 3, 4\}$ and $V_2 = \{1', 2', 3', 4'\}$ and $E = \{(1, 2'), (2, 4'), (3, 4'), (4, 2')\}$, so our matrix representation of $\mathcal{M}$ will be:

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

where rows two and four were padding with zeros. Note that we use prime in the second components of the element of the matching just to be consistence with the set of vertices of G.

2- The adjacency matrix representation of G and padding with zeros rows and/or

columns, if it is necessary.

$$A_G = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

where was not necessary to pad it to zero.

In this process of padding matrices, we use n be the maximum value between the two set of vertices i.e., $\max\{|V_1|, |V_2|\}$. So, after padding the matrices , we defined arbitrarily, the size of both matrices such that $|M| = k \times k$ where k correspond to the maximum index[1] of the maximal matching output of HK-Algorithm, and $|A_G| = n \times n$, where n is defined below. Hence, M and A_G are going to be square matrices which will make more simple the calculations. So, we begin to analyze the time consuming by the our algorithm dividing it by its steps. But before, let us start defining our "elementary steps" or "atomic operations" with a cost $O(1)$.

Definition 3.4.1 *The following operation are named atomic operations and have cost* $O(1)$*:*

- *write a new value in any entry* (i, j) *of a 0-1 matrix, and*
- *check if an entry* (i, j) *of a 0-1 matrix is* 0 *or* 1*.*

So, suppose that we would check if an arbitrary line—row or column—p of a

[1]Remember that $\mathcal{M} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ i.e., $\mathcal{M}$ is a list of all the positions of M with a 1 in them ($k = |\mathcal{M}|$).

matrix A with entries over $GF(2)$, is zero we define the following $\mathcal{L}_{\mathbf{LA}}$ predicate P,

$$\mathsf{P}(A,p) := \lambda_{ij}\langle 1, c(A), e(p,j,A)\rangle = \lambda_{ij}\langle 1, c(A), 0\rangle$$

where the 0 on the right side of the equality, i.e., $\lambda_{ij}\langle 1, c(A), 0\rangle$ correspond to an element of $\mathbb{Z}$, the ring of integers. Besides, $\mathsf{P}(A,p)$ in terms of atomic operations is:

$$c(A)\text{-many atomic oprations} \begin{cases} \text{for } i = 1, 2, \ldots, c(A) \\ \quad e(p,i,A) == 0 \ \} \text{ is a single atomic operation} \end{cases}$$

Recall that $\mathsf{P}(A,p)$ costs $O(n)$, where n is the length of the line.

3.4.1.2 High Level Complexity Analysis

In this section we investigate in a high level, the complexity of our Permutation-Based Algorithm. Remember that another thing to consider is that our input instances are of size, for example, input of size$(n) = n^2$. So, when we talk about input size n, we really mean parameterized inputs of size n^2.

We are going to analyze each algorithmic step through the *worst case*, which for us is when at least two green 1s "appear" in every recursive call that the Permutation-Based Algorithm does. This comes from the following claim:

Claim 3.4.1 *Let $k = |M|$ be the size of the matrix representation of the maximum matching M, let $G = (V_1 \cup V_2, E)$ be a bipartite graph, let $A_1^{k\times(|V_1|-k)}$ and $A_2^{(|V_2|-k)\times k}$ be submatrices of PA_GQ. If $A_1 \neq 0 \wedge A_2 \neq 0$, then must exist at least two green 1s on the main diagonal of PA_GQ matrix.*

Proof: We know by assumption that $A_1 \neq 0$, so, must be an arbitrary nonzero entry

on A_1 denoted by $A_1(i,j) \neq 0$, similarly on A_2 must exists an arbitrary nonzero entry denoted by $A_2(s,t) \neq 0$. Therefore, we are in the situation of PA_GQ depicted in Figure 3.3.

$$PA_GQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0_{(n-k)\times(n-k)} \end{array}\right] =$$

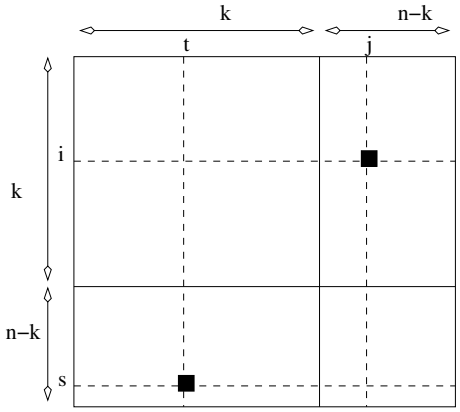


Figure 3.3: Two green 1s condition. The black boxes represent the nonzero entries of A_1 and A_2

Hence, we know that at least exist two green ones inside the $k \times k$ upper left corner of PA_GQ, because otherwise we are in the following case: suppose that exist only one green 1 inside the $k \times k$ upper left corner, then that means that $i = t$ but this is not possible, because it means that exist an entry with a 1 on it such that require two lines to be cover which is a contradiction by König's Min-Max Theorem.

Therefore we have that $i \neq t$, and our Claim follows. □

Note that we are not adding any restrictions on j and s because, by construction of PA_GQ, s and j never cross it inside of the $k \times k$ upper left corner. That is, the situation described below is valid for any s, and j.

Recall that is in the recursion call of our algorithm that computes the orientation

of the lines over the black 1s. Therefore, in order to investigate more closely the recursive step, we analyze the following cases:

Case 1: if there are *only greens 1's* on the k elements on the main diagonal then there is not recursive step because the fact that all ones are green means that our algorithm has determined before the recursive call all the k lines orientation hence it terminates.

Case 2: if there are only black 1s there is no recursion step because our algorithm is in the **Step 2a**, that is, $(A_1 = 0 \vee A_2 = 0)$, therefore recursion doesn't happens.

Case 3: according to Case 1 and Case 2, our algorithm make use of recursion steps *only* when there are a mix of green 1s and black 1s among the k first elements on the main diagonal. But, by Claim 3.4.1, we know that there are at least two green 1s among the black 1s on the main diagonal, therefore our algorithm enters in a recursive step when that happens. For example, an instance of a Case 3 is depicted in Figure 3.4.

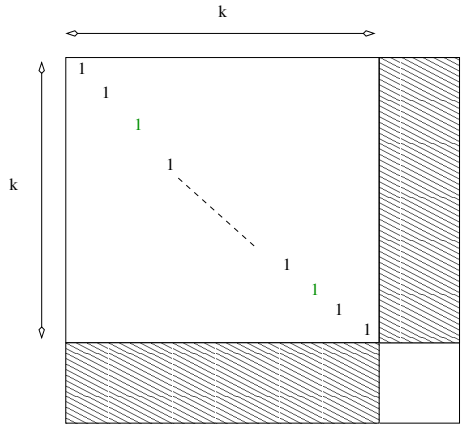


Figure 3.4: Recursive Situation. At least two green 1s and the remainder black 1s. The shared blocks denote nonzero blocks.

So, our recursive procedure will continue until reach to the following situation Figure 3.5, always in the context of the worst case scenario.

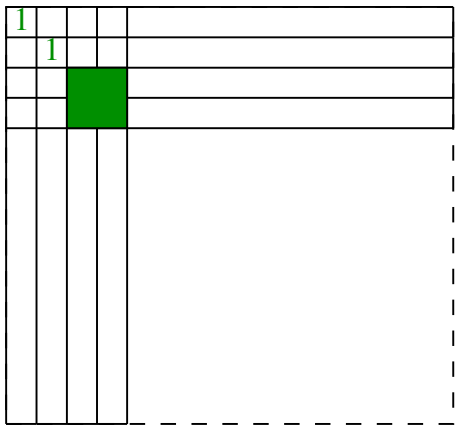


Figure 3.5: Last Step, we have a 4×4 matrix with two green 1s on the diagonal of the 2×2 upper left corner

Therefore, our Permuted-Based Algorithm in the worst case scenario makes $k/2$ recursive calls, where $k = |M|$. For each recursive call we start working on a matrix of size $n \times n$, where $n = |V|$ the cardinality of the set of vertices of G. The entire matrix is schematized in the Figure 3.6.

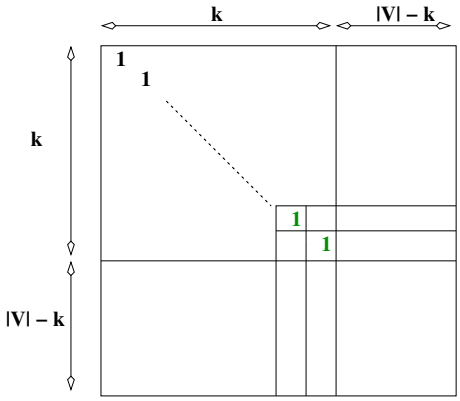


Figure 3.6: First Recursive call

At every recursive step our matrix decreases at least by two rows and two columns which corresponded to the green 1s. This can be expressed formally by the following recursive formula:

$$\begin{aligned} R(n) &= \underbrace{O(n^2)}_{(1)} + \underbrace{O(n^2)}_{(2)} + \underbrace{R(n-2)}_{(3)} \qquad (*) \\ &= c_1 n^2 + c_2 n^2 + R(n-2) \\ &= cn^2 + R(n-2) \\ &\approx n^2 + (n-2)^2 + (n-4)^2 + \cdots + 1 \\ &= \frac{n(n-1)(2n+1)}{6} \\ &= O(n^3) \end{aligned} \tag{3.1}$$

In $(*)$ term (3) correspond to the recursive call discussed previously. Also in $(*)$ the term (1) correspond to the complexity extracted from **Step 1** of our algorithm, which is $(k = |M_A| = min\{m, n\})$, **Step 2** which is $(k = |M_A| < min\{m, n\})$, and **Step 2a** which is $(A_1 = 0 \vee A_2 = 0)$, such that, like we are in the worst case scenario, **Step 1** and **Step 2a** are zero and the entire complexity factor $O(n^2)$ comes from **Step 2** which is divided in the follow two main tasks.

Before continue with the description of each task performed by **Step 2** let us make some observations: first of all, at the beginning, we need to compute permutations matrices P and Q and yields PA_GQ where A_G is the adjacency matrix of a bipartite graph G, then we need to multiply two matrices use $O(n^2)$ steps. So, we are going to explain how we will manage multiplication of permutations by using, for example, the μ permutation which split black 1s and green 1s on the main diagonal. (See for more details about computing permutations Appendix A.3). Secondly, the same

process of multiplication mentioned below is used to get any multiplication computed by our algorithm.

Therefore, the tasks computed in **Step 2** of our algorithm are:

Task 1: we make use of the structure of the permutation matrices involved in **Step 2** in order to store them with exactly n bits, using in memory a linear representation, we show this by the follow example:

For instance, let $R^{n\times n}$, or just R, be the permutation matrix associated to the μ permutation—like a bijection—which is used to split green 1s and back 1s, hence, let μ defined by $\mu = (i_1, i_2, i_3, \ldots, i_n)$ where i_l represent the (l, i_l) entry where R has a 1, that is,

$$R(l, i_l) = 1 \iff i_l \text{ represent the } (l, i_l)\text{-entry for } i_l \in \{i_1, i_2, i_3, \ldots, i_n\}$$

Hence, we are using only $|\mu| = n$ bits.

Note that in the process of Compute P,Q (Appendix A.3 Section A.3.1.1, Section A.3.1.2), and finally yield PA_GQ, we require $O(n^2)$ steps using the cycle decomposition—see below—and more important it is that our algorithm does this process *only once* at the beginning. In short, the linear representation of the permutation involved in **Step 2**, and the computation of P, Q and their product "once" at the beginning are two key concept of Task 1.

Task 2: using the fact that we are representing our permutations like

$$\mu = \begin{pmatrix} 1 & 2 & \ldots & n \\ i_1 & i_2 & \ldots & i_n \end{pmatrix}$$

and doing use of a *cycle decomposition* (for more on permutation and cycle decomposition see Appendix A.4) we will use only $O(n^2)$ steps to do the products involved in the splitting process of green 1s and black 1s. We do that in the following way:

Give μ like defined above, we use a cycle decomposition of μ such that an n-cycle will be

$$(i_1, i_n)(i_2, \ldots, i_{n-1})$$

where the commas can be omitted and the simple cycle $(i_1 i_n)$ represent the permutation that send $1 \mapsto n$ and $n \mapsto 1$, i.e., a transposition, and the order between permutation and the cycle ordering within a permutation does not matter, so $(i_1 i_n) = (i_n i_1)$.

- This kind of representation correspond to a product of *disjoint cycles*—two cycles are disjoint if they do not have any common elements. For instance, consider the permutation π on the set of integers $J_3 = \{1, 2, 3\}$ where the sub-index represent the cardinality of the set, given by $\pi(1) = 3$, $\pi(2) = 2$, $\pi(3) = 1$, then the cycle decomposition of π is: $\pi = (13)(2)$, that is, a product of two cycles (13) that maps $1 \mapsto 3$ and $3 \mapsto 1$ and cycle (2) that maps 2 to itself, we can ignore cycles of size one, so $\pi = (13)$.

We can called this process *swap decomposition* meaning writing n-cycles like a products of cycles. We are using n swapping in order to swap two arbitrary lines—rows/columns—such that, every swap has a cost of 4 atomic operations, because, again, we give an example, suppose that we want to swap position a and position b, see Figure 3.7.

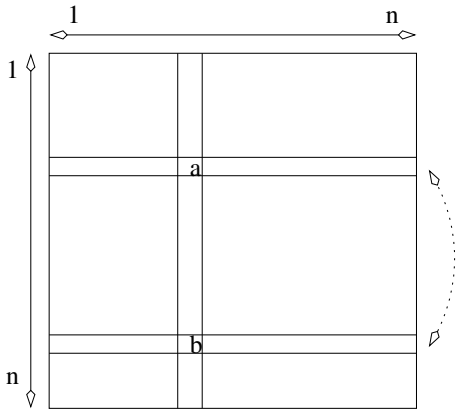


Figure 3.7: Swapping lines

First, we check if position a has a 0 or a 1, afterwards we check if position b has a 0 or a 1, then in case of be necessary swap the values, we proceed to write a new value in a as a new value in b (see Definition 3.4.1).

Therefore, putting all together we use 4 atomic operations for each position of our arbitrary row/column. Then we have $4n$ elementary steps/atomic operations in order to swap two lines.

Finally, in the worst case scenario we use $4n^2$ steps to swap the n lines of an n square matrix, getting then the required $O(n^2)$ steps from **Step 2**.

Note that we are using of auxiliary position of memory to make possible this swap, but this increase the constant factor, so, does not affect the calculus of the running time of **Step 2**.

In Equation 3.1 in $(*)$ the term (2) correspond to the **Step 2b** which is when both submatrix A_1, A_2 are not zeros, i.e., $(A_1 \neq 0 \wedge\ A_2 \neq 0)$, to be more specific correspond– inside this **Step 2b**–to the tasks that fill the lines with zeros in which was a green

one and fill of zeros the lower right corner, in every recursive call. Also, and more important–according to the contribution to the complexity of **Step 2b**–is a factor of $O(n^2)$ steps that come from the task of splitting green 1s from black 1s on the main diagonal in every recursive call and updating the orientation vector $\vec{O}$ according to the *new* distribution of the diagonal elements; a more detailed complexity analysis of this step can be found in the next section—Tight Complexity Analysis.

It is clear that we are overestimating the cost of this process because we have a reduction of two rows and two columns in every recursive call because we are in the worst case scenario.

3.4.1.3 Tight Complexity Analysis

We start this section remembering the Theorem 3.3.1 showed in this chapter Section 3.3, that is,

Theorem *(Theorem 3.3.1) Given a bipartite graph $G = (V = V_1 \cup V_2, E)$, we obtain a procedure for computing a Minimum Vertex Cover from a Maximum Matching that runs in time $|\langle A, M_A \rangle| = |V|^2$. That is, given a 0-1 matrix and the corresponding maximum matching, we can compute the vertex cover in linear time (in the size of the input).*

We will define a more tight formula of the recursion relation—Equation 3.1—found it in Section 3.4.1.2.

Let $R(n)$ be the maximum number of steps, in the worst-case, that our Permutation-Based Algorithm takes on a matrix of size n. Recall that a "step" is a single "atomic/elementary operation" defined in Definition 3.4.1, and our *worst case* is when at least two green 1s "appear" in every recursive call that the Permutation-Based

Algorithm does.

As we apply the permutation matrices P, Q only at the beginning of the procedure, the cost of these permutations is $O(n^2)$—as can be seen in Section 3.4.1.2. Once P, Q are computed, we deal with PA_GQ where the 1's associated with the matching are given in the main diagonal, that is, the diagonal of submatrix E.

Let us make the following remark about some cases which we call *entangle cases*—see at the end of this section: if there are many green 1's, the procedure has fewer recursive calls; if there are few green 1's, the procedure has more recursive calls. Suppose that k is the number of back 1's, and $n-k$ is the number of green 1's. Then, we can see that the worst-case analysis yields the following bound on the number of atomic steps:

$$R(n+1) = \max_{1<k<n} \{k(n+1-k) + R(k)\} \tag{3.2}$$

First note that there must be at least one black 1, for otherwise, there are no more steps. There must also be at least two green 1's (see Claim 3.4.1), if there were only one green 1, then either A_1 or A_2 would be all zero, which would also terminate the algorithm. Finally, no green 1's would imply termination as well. Hence the maximum is computed over $1 < k < n$ for size $n+1$.

Note that in the recursive Equation 3.2, k represents the number of black 1's, and so the corresponding sizes of A_1 and A_2 are given by $k \times (n+1-k)$ and $(n+1-k) \times k$, and hence the term $k(n+1-k)$—we are ignoring constants in Equation 3.2; it should really be $2k(n+1-k)$, but its does not change the order of $R(n)$. The recursive step is repeated on the black 1's, and hence we add $R(k)$ in Equation 3.2.

Now, we proceed to a more formal definition and we prove by induction on n that $R(n) \leq n^2$, under the assumption that $R(0) = R(1) = 1$. That is,

Definition 3.4.2 *Let n be input size of our Permutation-Based Algorithm defined in Section 3.3, let k be the size of the maximum matching. Then, let $R(n)$ be defined recursively by:*

$$R(0) = R(1) = 1$$

$$R(n) = \max_{1<k<n} \{k(n-k) + R(k)\}$$

Graphically (Figure 3.8),

$$PA_GQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0_{(n-k)\times(n-k)} \end{array}\right] =$$

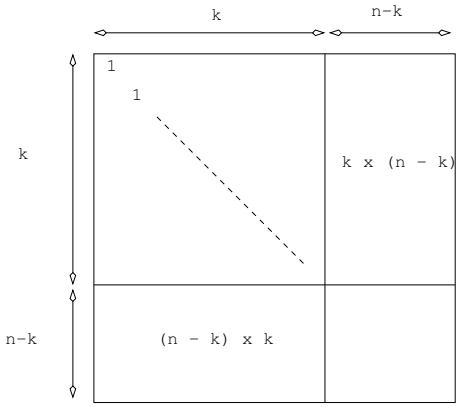


Figure 3.8: Recursive instance

From Definition 3.4.2 we have that the term $k \times (n-k)$ comes from the fact that we need to explore $(n-k)_j$ rows *and* $(n-k)_j$ columns in order to determine if the j position on the main diagonal is green or black. Hence, we have a contribution time complexity of $O(k(n-k))$ omitting the constant. The other factor of Definition 3.4.2, i.e., $R(k)$, correspond to recursive call over the k by k upper left corner.

Keep in mind that our input instances are of size input size$(n) = n^2$. So, when we talk about input of size n, we really mean parameterized inputs of size n^2.

Claim 3.4.2 *Let n be input size of our Permutation-Based Algorithm defined in Section 3.3, let R be defined like Definition 3.4.2, then $R(n) \leq n^2$.*

PROOF: By induction on n we get the following cases:

Basis Case: $n = 0$ and $n = 1$ by definition of R they holds, therefore there is nothing to prove.

Inductive Step: Suppose that R holds for n. Let us prove it for $n + 1$.

$$R(n+1) = \max_{1<k<n} \{k(n+1-k) + R(k)\} \qquad \text{by Def.}$$

by the I.H. we have that $R(k) \leq k^2$ since $k < n$,

$$\begin{aligned}
&\leq \max_{1<k<n} \{k(n+1-k) + k^2\} \\
&= \max_{1<k<n} \{kn + k - k^2 + k^2\} \\
&= \max_{1<k<n} \{kn + k\} \\
&\leq n \cdot n + n \\
&= n^2 + n \\
&\leq n^2 + 2n + 1 \\
&= (n+1)^2
\end{aligned}$$

□

Therefore, we conclude saying that the running time complexity cost of our Permutation-Based Algorithm is **linear** in the input size.

Now, we proceed to explain in more detail the entangle cases that we mention before. Without loss of generality, let Figure 3.8 be the instance depicted for analyze the following both cases.

Case a: assume $k >> \alpha$, i.e., let k be bigger than some particular factor α, such that the *operations* of splitting black 1s and green 1s and prepare the matrix to the next recursive call—where "prepare" means the task described in this chapter Figure 3.2—that our algorithm does on each recursive call are **expensive** (with respect to the number of atomic operations) but with the advantage that there are **few** of such operations, this means that A_1 and A_2 are of small size in relation to the factor α, that is, bigger α smaller $(n - k)$.

In short, a big k imply expensive few elementary/atomic operations and small dimension of A_1 and A_2.

Case b: assume that $k << \alpha$, i.e., k smaller than some factor α such that the *operations*—the same that Case a—that our algorithm does in each recursive step are **cheap** but with the disadvantage that there are **many** of such operations, which means that A_1 and A_2 are of big size in relation to the factor α.

So, in this case, k is small which imply cheap many atomic/elementary operations and big dimension of A_1 and A_2.

3.4.2 Data Structure

The purpose of this section is to review some very basic data structures used with Graph Algorithms. This section will be just a remark about "possible" data structure to be used in our Permutation-Based Algorithm, possible in the sense that we express

which is a "good" of many data structure, and good in the sense that we consider Combinatorial Matrix Theory aspect of matrices used in our algorithm, i.e., matrices denoting bipartite graphs, matching matrices, permutation matrices, 0-1 matrices with König's Property, i.e., satisfying the König Min-Max Theorem.

3.4.2.1 Issues of implementation

In this section we are going to show some usual Data Structures (DS) used to represent graphs with its advantages and disadvantages. More information can be found in [CLRS09, Ch. 22]. Another very good references are, for instance, [Meh84, AHU74, AHU83, Gon84].

We start with the one of the two major used representations about graphs which is *Adjacency List*, next we show a variation of the this one, which is called *Adjacency Hash*, and finally, we show the other major used representation named *Adjacency Matrix*.

- *Adjacency List Representation:*

  This representation is preferred to represent *sparse* graphs , that is, those for which $|E| << |V|^2$. The adjacency list representation of $G = (V, E)$ consist of an array Adj of $|V|$ many lists, one for each vertex in V. For each $u \in V$ the adjacency list, denoted $Adj[u]$, contains all the vertices v adjacent to u that is, vertices such that there exists an edge $(u, v) \in E$.

  Now, we have two cases to be consider about the graph, those are, if G is a *directed*, the sum of the lengths of all adjacency lists is $|E|$, but if G is an undirected graph, the sum is $2|E|$. For both cases the adjacency list representation has the property that require $\Theta(|V| + |E|)$ amount of memory.

The main disadvantage is that this kind of data structure is not a fast way to determine if a given edge is present if the graph. In order to solve that problem we can use the Adjacency Matrix Representation—see below–but at the cost of using asymptotically more memory.

- *Adjacency Hash Representation:*

  The *adjacency hash* representation of a graph $G = (V, E)$ consist of an array Adj of $|V|$ many hash tables, i.e, which a hash table on each vertex in V. Then each entry of $Adj[u]$ is a hash table containing the vertices v for which $(u, v) \in E$.

  So, let us look some properties about this DS. Suppose that for all edge look-up are equally likely, the the expected time to determine whether an edge is in the graph is $O(1)$ since just going to the hash table, say t, i.e., $t = Adj[u]$ and get v.

  One thing to be consider it is that this DS will require more space than the linked list version—see previous representation, making this point one of its disadvantage.

  On the other hand, another point to be consider is the type of hash function that we choose in order to have the best performance, for instance, Division Method, Multiplication Method, Universal Hashing, etc., and more important how this hash functions deals with collision , for instance, using Open Addressing schema we can manage the collision using several approach like linear or Quadratic Probing, Double Hashing, etc, which represent the principal disadvantage of this DS.

  For example, suppose that in our hash table each vertex to the list of adjacency vertices have the same key. If the distribution of vertices is sufficiently uniform,

i.e., each vertex appears with equal probability into any of the $|V|$ slots, then the average cost of a lookup depends on the average number of vertices per each list (that is, the Load Factor, say α), so the $\alpha = n/m$ where m is the size of hash table and n number of vertices in the table.

So, the expected time to determine whether an edge is in the graph is $O(n/m)$ having more space than Adjacency List approach and more query time than Adjacency Matrix. If our hash table supports dynamic resizing then we would need extra time to move the elements between the old and new hash tables and if not we would need $O(n)$ space for each hash table in order to have $O(1)$ query time which results in $O(n^2)$ space. Also we have just checked expected query time, and in worst case we may have query time just like Adjacency List, i.e., $O(deg(u))$ so it seems better to use Adjacency Matrix in order to have deterministic $O(1)$ query time and $O(n^2)$ space.

- *Adjacency Matrix Representation:*

  First of all, this DE is what we decided to use it. So, this data structure representation is preferred to represent *dense* graphs—those for which $|E| \approx |V|^2$—or when we need to be able to answer fast if there is an edge connecting two given vertices.

  For this representation we assume that the vertices are numbered from $1, 2, \ldots, |V|$ in some arbitrary order. Then the adjacency matrix representation of G consist

of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 0 & \text{if } (i,j) \in E, \\ 1 & \text{otherwise.} \end{cases}$$

The adjacency matrix of a graph requires $\Theta(|V|^2)$ memory, independently of the number of edges in G. Let $u \in V$ be a vertex of G we denote by $d_G(u)$ the degree of u—the number of incident edges—and by $N_G(u)$ the open neighbourhood of v which is the set of vertices adjacent to v.

So, we summarized, mainly, the advantage and disadvantage of the two most important representation by showing some typical operations over graphs which are, first, answer the question about $(u, v) \in E$? where with Adjacency List takes $O(\min\{d_G(u), d_G(v)\})$ while using Adjacency Matrix Representation we have $O(1)$.

On the other hand, we have that using Adjacency List we have $O(d_G(u))$ again $O(|V|)$ in the process of go over $N_G(u)$. Lastly, when we go over E, we have $O(|V| + |E|)$ using adjacency list representation while with matrix representation we get $O(|E|)$.

We have seen several representations of graphs, but there are subtle differences that can be tuned to get a maximum performance considering our particular application on our algorithm. We can use the Adjacency Matrix representation if our algorithm works over a dense graphs, or we can use Adjacency List if our graph change frequently, finally, some variants of adjacency list like, Adjacency Hash are faster that linked list in some particular cases, like we saw before.

We use the Adjacency Matrix Representation for our DE, and the two main reasons are that it is fast to answer the question if a given edge $(u, v) \in E$, and secondly, like our algorithm works with an unweighted graph, there is an additional advantage in

storage for the adjacency matrix which come from that rather than using one word of computer memory for each matrix entry, it uses only one bit per entry.

Chapter 4

A Proof Complexity approach to König's Min-Max Theorem

4.1 Feasible proofs in Linear Algebra

The *Proof Complexity* (PC) of *Combinatorial Matrix Theory* (CMT) is related to the *Proof Complexity* of linear algebra, and hence we are going to use **LA**-Theory in order to formalize reasoning in CMT.

The PC of linear algebra, roughly speaking is the complexity of concepts needed to prove the basic properties of linear algebra operations. The PC has two aspects, the *uniform* concerns the power of logical theories required to prove a given assertion, and the *nonuniform* aspect which concerns the power of propositional proof systems required to yield polynomial size proofs of a tautology family representing the assertion. Here we are concerned with the *uniform* case.

4.1.1 Why do we want to find feasible proofs?

As was mentioned at the beginning of this thesis, the main goal of *Bounded Reverse Mathematics* is to find the weakest theory capable for proving a given theorem—in our case we "use" as "a source of theorems" the mathematical field of Combinatorial Matrix Theory—and in general these can be proved in weak theories. The adjective *Bounded* refers to bounds on the sizes of quantified matrices, that is, $\forall X \leq n$ means all matrices X with at most n rows and columns.

Rather, the point of Proof Complexity is to see which combinatorial theorems have feasible proofs. Then, Proof Complexity takes the **P vs. NP** problem from computation and "translates" it into **NP vs. co-NP** in proofs.

More precisely, Proof Complexity is an area of Mathematics and Theoretical Computer Science that studies the length of proofs in propositional logic. It is an area of study that is fundamentally connected both to major open problem of computational complexity theory and practical properties of automated theorem provers [BP98]. For more details see Chapter 1 Section 1.1.

In this section we justify our Proof Complexity approach to König's Min-Max Theorem, with reference to Krajíček in [Kra10].

A fundamental notion that appears throughout this thesis is that of *feasible proof* (and *feasible computation*, or *polynomial time computation*). Feasible proofs were introduced by Cook in [Coo75], and they formalize the idea of tractable reasoning; a theorem can be proven feasibly, if all the computations involved in the proof are polynomial time computations, and the induction can be unwound feasibly. So, we understand feasible proofs as proofs that use only polynomial time concepts.

We shall discuss some situations in which it is possible to extract from a proof some

feasible computational information about the theorem being proved. This includes extracting feasible algorithms, deterministic or interactive, for instance, for witnessing an existential quantifier.

Universal Theories

Let $\mathcal{L}$ be a language that has a function symbol corresponding to every polynomial time algorithm, say as represented by clocked polynomial time Turing machines[1]. We shall assume that polynomial time relations are represented by their characteristic functions and hence the only relation symbol is the equality, which we regard as a logical symbol.

Every function symbol from $\mathcal{L}$ has a canonical interpretation on the set of natural numbers $\mathbb{N}$ which we identify with the set of all binary words $\{0,1\}^*$; the resulting $\mathcal{L}$-structure will be called the standard model. Let T be the universal theory[2] of the standard model.

Witnessing existential formula

Assume

$$T \vdash \exists y A(x,y)$$

where A is an open formula, i.e., is formed by combining atomic formulas using logical connectives. Herbrand's theorem implies that there are terms $t_1(x),\ldots,t_k(x)$ such that

$$T \vdash \bigvee_i^k A\left(x, t_i(x)\right)$$

[1] A polynomial clock $C_{(a,b)}$ is a total TM that behaves as follows: for binary input x of length $|x|$ it computes $|x|^a + b$, a, b positive integers, and stops the operation of the coupled machine $M_m(x)$ after $|x|^a + b$ cycles, if it has not stopped yet.

[2] A theory is a set of formulas, if the formulas have no existential quantifiers, the theory is called Universal.

As the class of polynomial time functions is closed under definitions by cases distinguished by a polynomial time property, it follows that there is one term $t(x)$ such that

$$T \vdash A(x, t(x))$$

The polynomial time algorithm defined by $t(x)$ witnesses in the standard model the validity of the sentences $\forall x \exists y A(x, y)$. Consequences of this witnessing is, for instances:

- Example 1: We give an example of a first order formula for which we believe there is no polynomial time proof. We start by recalling some standard definitions, see [Sol09]. A language $\mathcal{L}$ has a *proof system* if there exists a polynomial time binary predicate[3] $R(x, y) : \quad x \in \mathcal{L} \Leftrightarrow \exists y \; R(x, y)$ where y is called *proof*, i.e., the encoding of a proof that $x \in \mathcal{L}$.

  In this context, *soundness* means that $\exists y \; R(x, y) \Rightarrow x \in \mathcal{L}$, and *completeness* means that $x \in \mathcal{L} \Rightarrow \exists y \; R(x, y)$. Finally, the *complexity* of R is $f_R : \mathbb{N} \to \mathbb{N}$, where

  $$f_R(n) = \max_{\{x \in \mathcal{L}, |x|=n\}} \min_{y, R(x,y)} |y|,$$

  and R is *polynomially bounded* iff f_R is bounded by some polynomial, i.e., there exists a polynomial q such that $f_R \leq q(n)$.

  Let P be a propositional proof systems, e.g., is a proof system for $TAUT$, where $TAUT$ is a language defined as follow:

  $TAUT = \{\langle \phi \rangle : \exists y R(\langle \phi \rangle, y)\}$ where $R(\langle \phi \rangle, y) :=$"y encodes a derivation of ϕ"

[3] An k-ary predicate $S(x_1, \ldots, x_k)$ is polynomial time if exists a Turing Machine M which on input $\langle x_1, \ldots, x_k \rangle$ decides $S(x_1, \ldots, x_k)$ in time $P(|x_1| + \cdots + |x_k|)$ for a fixed polynomial P.

this derivation could be, for example, a truth table.

Any π such that $P(\pi) = \tau$ is called a *P-proof of* τ. The completeness of P can be stated as :

$$\forall x \exists y \left[Fla(x) \Rightarrow (SatNeg(x, y) \vee P(y) = x) \right]$$

where $Fla(x)$ is $\mathcal{L}$-formula formalizing that x is the encoding of a propositional formula, $SatNeg(x, y)$ is $\mathcal{L}$-formula formalizing that x is a propositional formula and y is a truth assignment satisfying the negation of x. Unless P = NP, this is unprovable in T as a witnessing algorithm could be used to decide SAT. As long as T is a polynomial time theory—and this is only a conjecture.

The point is, if $T \vdash p$, then we can witness y in polynomial time (in $|x|$), and therefore compute either an assignment that has not satisfy x, or the encoding of a proof of x if x is a tautology. The existence of such a witnessing function would imply P = NP which we believe not to be the case.

- Example 2: Let h be a *one-way permutation*, i.e, a one-way function[4] that is also a permutation (bijective). As h must be surjective the sentence

$$\forall y \exists x \quad h(x) = y$$

is valid.

[4] $f : \{0.1\}^* \to \{0,1\}^*$ is one-way if can be computed by a polynomial time algorithm, but for every randomized polynomial time algorithm A, $Pr\left[f\left(A\left(f(x)\right)\right) = f(x)\right] < \frac{1}{p(n)}$ for every positive polynomial $p(n)$ and sufficiently large n, assuming that x is chosen from the uniform distribution on $\{0,1\}^n$ and the randomness of A.

But it is not provable in T as the witnessing algorithm would compute the inverse function to h, which is supposed to be impossible because "The existence of a one-way function is still an Open Conjecture". (For more details see [AB09]).

4.1.2 The Theory LA

We present a short survey of **LA**-Theory based on [SC04, Section 2]. **LA** was introduced to study the complexity of the concepts needed to prove the basic theorems of linear algebra.

While we consider matrices over $\{0, 1\}$, the underlying ring is $\mathbb{Z}$, since we require that ΣA compute the number of 1s in the matrix A (which for a 0-1 matrix is simply the sum of all entries—meaning ΣA). Thus, over $\mathbb{Z}$, **LA** translates to $\mathbf{TC}^0$-Frege, [SC04, §6.5].

LA-Theory is a field-independent logical theory for expressing and proving matrix properties. **LA** proves all the ring properties of matrices (i.e., commutativity of matrix addition, associativity of matrix products, etc.). While **LA** is strong enough to prove all the ring properties of matrices, its propositional proof complexity is low, that is, over $\mathbb{Z}$, all the theorems of **LA** translate into $\mathbf{TC}^0$-Frege proofs,[SC04, §6.5].

LA is a quantifier-free theory with three sorts: indices (i.e., natural numbers) denoted $i, j, k, \ldots$, field elements denoted $a, b, c, \ldots$, and matrices denoted $A, B, C, \ldots$, and all theorems hold for any choice of the underlying field. The semantic assumes that objects of type field are from a fixed but arbitrary field, and objects of type matrix have entries from that field. **LA** allows the basic ring properties of matrices to be formulated and proved.

Soltys in ([Sol01]) defined the **LA**-Theory to be a set of sequents, using sequents,

rather than formulas, for two reasons: (i) sequents are convenient for expressing matrix identities, for instance the "hard matrix identities" (see Section 3.2 on [Sol01]), and (ii) **LA** uses sequent calculus proof system to formalize propositional derivations. **LA** is defined as a set of sequents which have derivations from the axioms $A1 - A33$ (see Section 2.3 on [Sol01]), a few of that axiom are expressed below.

The Language $\mathcal{L}_{\mathbf{LA}}$

Terms and formulas are built from the following function and predicate symbols, which together comprise the language $\mathcal{L}_{\mathbf{LA}}$:

$$0_{index}, 1_{index}, +_{index}, *_{index}, -_{index}, div, rem,$$
$$0_{field}, 1_{field}, +_{field}, *_{field}, -_{field}, ^{-1}, r, c, e, \Sigma,$$
$$\leq_{index}, =_{index}, =_{field}, =_{matrix}, cond_{index}, cond_{field}.$$

The intended meanings should be clear, except $-_{index}$ is cutoff subtraction, that is, $(i - j = 0$, if $i < j)$, a^{-1} is the inverse of a field element a with $0^{-1} = 0$, and for the following operations on a matrix $A : r(A), c(A)$ are the number of rows and columns in A, $e(A, i, j)$ is the field element A_{ij} (where $A_{ij} = 0$ if $i = 0$ or $j = 0$ or $i > r(A)$ or $j > c(A)$), $\Sigma(A)$ is the sum of the elements in A. Also $\texttt{cond}(\alpha, t_1, t_2)$ is interpreted **if** α **then** t_1 **else** t_2, where α is a formula all of whose atomic subformulas have the form $m \leq n$ or $m = n$, where m, n are terms of type index, and t_1, t_2 are terms either both of type index or both of type field. This restriction is because in the translations into propositional formulas (see [Sol01, Chapter 7] for the details of these translations) all the free index variables get values, and therefore α will become true or false.

We use n, m for terms of type index, t, u for terms of type field, and T, U for terms

of types matrix. Terms of all three types are constructed from variables and the symbols above in the usual way, except that in addition terms of types matrix are either variables $A, B, C, \ldots$ or λ terms $\lambda ij\langle m, n, t\rangle$. Here i, j are variables of type index bound by the λ operator, intended to range over the row and columns of the matrix. Here also m, n are terms of type index *not* containing i, j (representing the numbers of rows and columns of the matrix) and t is a term of type field (representing the matrix element in position (i, j)). The idea behind the constructed term λ is to avoid having to define a whole range of matrix functions (matrix transpose, matrix addition, etc.). Instead, since matrices can be defined in terms of their entries, we use functions of type field to define matrix functions; the λ operator allows us to do this, for instance, suppose A, B are 2×2 matrices, then $A + B$ can be defined $\lambda ij\langle 2, 2, e(A, i, j) + e(B, i, j)\rangle$.

Atomic formulas have the forms $m \leq_{index} n$, $m =_{index} n$, $t =_{field} u$, $T =_{matrix} U$. Formulas are built from atomic formulas using the propositional connectives $\neg, \vee, \wedge$. Formulas may not have quantifiers.

Terms for LA

The λ terms allow us to construct the sum, product, transpose, etc., of matrices. We use the notation := to introduce abbreviations for terms.

Integer maximum

$$max\{i, j\} := cond(i \leq j, j, i).$$

Matrix sum

$$A + B := \lambda ij\langle max\{r(A), r(B)\}, max\{c(A), c(B)\}, A_{ij} + B_{ij}\rangle.$$

Scalar product

$$aA := \lambda ij\langle r(A), c(A), a * A_{ij}\rangle.$$

Matrix transpose

$$A^t := \lambda ij\langle c(A), r(A), A_{ji}\rangle.$$

Zero and Identity matrices

$$0_{kl} := \lambda ij\langle k, l, 0\rangle$$

and

$$I_k := \lambda ij\langle k, k, cond(i = j, 1, 0)\rangle.$$

Matrix trace

$$tr(A) := \Sigma\lambda ij\langle r(A), 1, A_{ij}\rangle.$$

Dot product

$$A \cdot B := \Sigma\lambda ij\langle max\{r(A), r(B)\}, max\{c(A), c(B)\}, A_{ij} * B_{ij}\rangle.$$

Matrix product

$$A * B := \Sigma\lambda ij\langle r(A), c(B), \lambda kl\langle c(A), 1, e(A, i, k)\rangle \cdot \lambda kl\langle r(B), 1, e(B, k, j)\rangle\rangle.$$

Finally , the decomposition of an $n \times n$ matrix A will be used in our axioms defining $\Sigma(S)$:

$$\mathbf{A} = \begin{bmatrix} a_{11} & R \\ S & M \end{bmatrix}$$

where a_{11} is the $(1, 1)$ entry of A, and R, S are $1 \times (n-1), (n-1) \times 1$ submatrices,

respectively, and M is the principal submatrix of A. Therefore, we make the following precise definitions:

$$\begin{aligned} R(A) &:= \lambda ij\langle 1, c(A)-1, e(A,1,i+1)\rangle \\ S(A) &:= \lambda ij\langle r(A)-1, 1, e(A,i+1,1)\rangle \\ M(A) &:= \lambda ij\langle r(A)-1, c(A)-1, e(A,i+1,j+1)\rangle \end{aligned}$$

Substitution

The propose of this section is to denote the relevance of substitution instance of a term inside the theory **LA**, since **LA**-Theory is closed under substitution on terms of types index, field, and matrix, (for a proof see Section 2.4 Lemma 2.4.1 on [Sol01]) that is,

$$\text{If } \mathbf{LA} \vdash \alpha(x) \text{ then } \mathbf{LA} \vdash \alpha(t)$$

where both x and t are term of type index, field, or matrix. To do this, we follow the terminology and style of (see [Sol01, Section 2]).

Assume that t is a term, and we indicate that a variable x occurs in t by $t(x)$. If t' is also a term, of the same type as the variable x, then $t(t'/x)$ denotes that the free occurrences of the variable x have been substituted throughout t by t', hence, we say that $t(t'/x)$ is a *substitution instance* of t. If α is a formula then$\alpha(t'/x)$ is defined analogously.

However, the existence of bounded variables complicates things, so, to avoid problems, we give a precise definition of substitution, by structural induction on t:

Basis case: t is just a variable x, so $x(t'/x) =_{synt} t'$, where t' must be the same

type as x.

Induction Step: here examine the nine items from terms and formulas over $\mathcal{L}_{\mathbf{LA}}$ (see Section 2.2.1 [Sol01]). All items, except item 6 and 9, are straightforward.(for more details see Section 2.2.4 [Sol01]).

We only present item 6, suppose that t is of the form $\lambda\langle m, n, t\rangle$. If x is i or j, then the substitution has no effect, as we cannot replace bound variables. Assume that x is neither i nor j.

If t' doesn't contain i or j, then $\lambda ij\langle m, n, t\rangle(t'/x)$ is just:

$$\lambda ij\langle m(t'/x), n(t'/x), t(t'/x)\rangle \qquad (*)$$

If t' contain i or j, then, if we substituted like (*), the danger arises that x might occur in m or n, and violate the restriction that i, j do not occur free in m and n.

Moreover, if x also occur in t, then the i and j from t' would get caught in the scope of the λ-operator, and change the semantic of t in an unwanted way. Thus, if t' contains i or j, then, to avoid the problems listed above, we rename i, j in $\lambda ij\langle m, n, t\rangle$ to new index variables i', j', and carry on as in (*).

A detailed exposition of substitution and λ calculus can be found, for instance, in [HS86].

Lemma 4.1.1 *Every substitution instance of a term is a term of the same type. Similarly, every substitution instance of a formula is a formula, and every substitution instance of a sequent is a sequent.*

PROOF: See [Sol01, pg. 21] □

Proofs in LA

We use Gentzen's sequent calculus LK (with quantifier rules omitted) for the underlying logic, see [Bus98, Chapter 1]. A sequent has the form $\alpha_1, \ldots, \alpha_k \longrightarrow \beta_1, \ldots, \beta_l$

where the symbol $\longrightarrow$ is a new symbol called the sequent arrow, and each α_i and β_j is a formula. The intended meaning of the sequent is:

$$\forall x_1 \dots x_n \left(\bigwedge_{i=1}^{k} \alpha_i \supset \bigvee_{j=1}^{l} \beta_j \right)$$

where $x_1, \dots, x_n$ is the list of all the free variable of all three sorts that appear in the sequent. Note that **LA** is a quantifier-free theory, but all the sequents are implicitly universally quantified.

We let $\alpha \supset \beta$ stands for $\neg\alpha \wedge \beta$, and $\alpha \equiv \beta$ stand for $\alpha \supset \beta \wedge \beta \supset \alpha$.

The system LK has the axiom scheme $\alpha \longrightarrow \alpha$, the structural rule Exchange, Contraction, and Weakening (left and right), Cut rule, and rules for introducing each of the three connectives $\neg, \vee, \wedge$ on the left and right.

In addition to these axioms and rules, **LA** has axiom schemes and a rule for equality, an induction rule, and axiom schemes giving the properties of numbers, field, and matrices.

A *proof* in **LA** of a sequent S is a finite sequence of sequents ending in S, such that each sequent in the proof is either an axiom, or follows from earlier sequents by a rule of inference. If α is a formula, then we regard a proof of the sequent $\longrightarrow \alpha$ as a proof of α.

Axioms for LA (for a complete list see [SC04], and Appendix A.5):

The axioms of **LA** are really axiom schemes, because all substitutions instances of axioms are also axioms. The axioms are divided into four groups, equality axioms, the axioms of PA without induction, for indices, the axioms for field elements, and finally the axioms for matrices. For instance:

- the *axioms of arithmetic for indices*, for instance

$$\longrightarrow i + 0 = i.$$

- the *field axioms*, for instance

$$\longrightarrow a + (-a) = 0.$$

- *equality axioms*, for instance

$$x_1 = y_1, \ldots, x_n = y_n \longrightarrow f x_1 \cdots x_n = f y_1 \cdots y_n,$$

  where f can be any of the non-constant function symbols of **LA**.

- *matrix axioms*, for instance

$$(i = 0 \vee r(A) < i \vee j = 0 \vee c(A) < j) \longrightarrow e(A, i, j) = 0$$

  that is, states that $e(A, i, j)$ is zero when i, j are outside the size of matrix A.

Rules for LA

In addition to the logical rules of Gentzen's LK, **LA** has two rules: matrix equality and induction. In specifying the rules below, Γ and Δ are cedents, which can be empty.

Matrix equality rule

$$\frac{\Gamma \longrightarrow \Delta, e(T, i, j) = e(U, i, j) \qquad \Gamma \longrightarrow \Delta, r(T) = r(U) \qquad \Gamma \longrightarrow \Delta, c(T) = c(U)}{\Gamma \rightarrow \Delta, T = U}.$$

Here the variables i, j may not occur free in the bottom sequent; otherwise T and U are arbitrary matrix terms. The semantics implies that i and j are implicitly universally

quantified in the top sequent. The rule allows us to conclude $T = U$, provided that T and U have the same number of rows and columns, and corresponding entries are equal.

Induction rule

$$\frac{\Gamma, \alpha(i) \longrightarrow \alpha(i+1), \Delta}{\Gamma, \alpha(0) \longrightarrow \alpha(n), \Delta}$$

Here the variable i of type index may not occur free in either Γ or Δ. Also $\alpha(i)$ is any formula, n is any term of type index, and $\alpha(n)$ indicates n is substituted for free occurrences of i in $\alpha(i)$. Similarly for $\alpha(0)$.

Definition 4.1.1 *We define the proof system PK-**LA** to be a system of sequent calculus proof, where all the initial sequents are either of the form $\alpha \longrightarrow \alpha$, for any formula α over $\mathcal{L}_{\mathbf{LA}}$, or are given by one of the axiom schemes of* **LA**, *and all the other sequents (if any) follow from previous sequents in the proof by one of the PK rules*[5] *for propositional consequence, or by Induction rule, or by Equality rule.*

Thus, a PK-**LA** proof of a sequent S is an ordered sequence of sequents $\{S_1, S_2, \ldots, S_n\}$, where each S_i is either of the form $\alpha \longrightarrow \alpha$, or is given by one of the axiom schemes of **LA**, or follow from previous sequents, say S_j's by a PK rule for propositional consequence, or by Induction rule, or by Equality rule. The end sequent, S_n is S, that is, the sequent that we want to prove. The *length* of the proof is n.

Definition 4.1.2 *The theory* **LA** *is the set of sequents over $\mathcal{L}_{\mathbf{LA}}$ which have PK-**LA** proofs.*

Recall that the formula α is equivalent in meaning to the sequent $\longrightarrow \alpha$. Hence, we can omit the arrow, but formally **LA** is a theory of sequents, and so the arrow is there.

[5] That is, the weak structural rules, cut rule, and introducing connectives rule. For more about the propositional sequent calculus proof system PK, see [Bus98, Chapter 1]

Also, our derivations are informal, recall that a sequent S is in **LA** if an only if it has a PK-**LA** derivation. The point here is to keep in mind that the derivations can be formalized in PK-**LA**.

Substitution rule

$$\frac{S(x_1, \ldots, x_k)}{S(t_1/x_1, \ldots, t_k/x_k)}$$

Here S is any sequent, and $S(x_1, \ldots, x_k)$ indicate that $x_1, \ldots, x_k$ are variables in S. Finally, the expression $S(t_1/x_1, \ldots, t_k/x_k)$ indicates that the terms $t_1, \ldots, t_k$ replace all free occurrences of the variables $x_1, \ldots, x_k$ in S, simultaneously. Here, x_i has any of the three types, and the term t_i has the same type as x_i.

This completes the description of **LA**. For a more complete treatment see [Sol01], and Appendix A.5.

4.2 A Combinatorial Matrix Theory proof of König's Min-Max Theorem

We start this section giving a standard textbook proof—a Π_2^B proof—of König's Min-Max Theorem (KMM). Afterwards, in Section 4.3 we give a feasible proof of KMM. Recall that Theorem 3.1.2 Section 3.1 is which we will prove within a Combinatorial Matrix Theory approach.

This section is added for completeness; the next section contains our main contribution.

Theorem 4.2.1 (König's Min-Max version II) *Let A be a matrix of size $m \times n$ with entries over $GF(2)$. The minimal number of lines that cover all of the 1s in A*

equals to the maximal number of 1s in A, no two of the 1s on a line.

PROOF: Let ρ'_A be the minimal number of lines that cover all the 1s. Let ρ_A be the maximal number of 1s with no two on the same line. Throughout the entire proof line denote a row or a column interchangeably.

Easy to see $\rho'_A \geq \rho_A$, because if we have k 1s no two on the same line, we need at least k lines to cover them all. So, wherever ρ'_A is, it is bounded below by size of longest set of 1s with no two on the same line.

The other direction is more interesting, so we prove that $\rho'_A \leq \rho_A$.

Definition 4.2.1 *(Proper covering) A proper covering is a covering where at least one row is missing and at least one column is missing.*

We consider two cases: there is a minimal proper covering, and there isn't.

We now argue by induction on $\min\{m, n\}$, where m are the number of rows of matrix A, and n are number of columns of A.

Basis Step: in this case we have $\min\{m, n\} = 1$ and can be checked easily that Theorem 4.2.1 holds.

Inductive Step: the inductive step has two cases:

- Case 1: Suppose that there is no a proper covering of A.

  First, A must have a 1 somewhere, otherwise $\rho'_A = 0 < \min\{m, n\}$ given as a proper covering. Let's rewrite Lemma 3.2.2 from Chapter 3 Section 3.2—Preliminaries to our algorithm—according to the terms used in this proof, that is,

  Lemma 4.2.1 *Let P and Q be permutation matrices. Then $\rho'_A = \rho'_{PAQ}$ and*

$\rho_A = \rho_{PAQ}$. *Also, A has a proper covering if an only if PAQ has a proper covering.*

PROOF: See proof Lemma 3.2.2 Chapter 3, Section 3.2. □

So, continuing with Case 1, we pick up some 1 in A, and permute A so that this 1 is in the upper-left corner:

$$\hat{\mathbf{A}} = \mathbf{PAQ} = \left[\begin{array}{c|c} 1 & \\ \hline & A' \\ & \end{array}\right] \tag{4.1}$$

Thus, A' is the principal sub-matrix of $\hat{A}$.

By Lemma 4.2.1, if A does not have a proper covering, neither does $\hat{A}$. We want to show $\rho'_{\hat{A}} \leq \rho_{\hat{A}}$. So, by I.H.:

$$\rho'_{A'} \leq \rho_{A'}$$

from which we obtain

$$\rho'_{A'} + 1 \leq \rho_{A'} + 1 \tag{4.2}$$

Now, note that we have the following claim:

Claim 4.2.1 *Given a set S of 1's in A′ no two of which are in the same line then* $\rho_{A'} + 1 \leq \rho_{\hat{A}}$

PROOF: Let S be the following set $S = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ where all $i'_a s$ are different and all $j'_a s$ are different, and represent the elements of a set, that will be called matching set, and $\hat{A}_{(i_a, j_a)} = 1$ then $S \cup \{(1, 1)\}$ still has the

property that no two 1s are on the same line, and so $|S \cup \{(1,1)\}| \leq \rho_{\hat{A}}$ and $|S \cup \{(1,1)\}| = |S| + 1 = k + 1$ where we can pick S so that $k = \rho_{A'}$. □

On the other hand, since $\hat{A}$ does not have a proper covering (Case 1 and Lemma 4.2.1), we know that $\rho'_{A'} = \min\{m-1, n-1\}$, more formally:

Claim 4.2.2 *If $\hat{A}$ does not have a proper covering, then $\rho'_{A'} = \min\{m-1, n-1\}$, where A' is the principal submatrix of $\hat{A}$.*

PROOF: First note that for any matrix B, $\rho'_B \leq \min\{row(B), col(B)\}$ since all rows or all columns always form a covering. So, we only need to show that it cannot be the case that $\rho'_{A'} < \min\{m-1, n-1\}$.

Suppose it is. Consider the lines forming the covering of A' such that its size is $\rho'_{A'} < \min\{m-1, n-1\}$; if we add to those lines the first row of $\hat{A}$ and the first column of $\hat{A}$, then we get a covering of $\hat{A}$ of size $\rho'_{A'} + 2 \leq min\{m, n\}$ not consisting of all rows and columns of $\hat{A}$. Hence, $\hat{A}$ would have a proper covering. □

So, by Claim 4.2.2, we have that:

$$\begin{aligned} \rho'_{A'} + 1 &= \min\{m-1, n-1\} + 1 \\ &= \min\{m, n\} \\ &= \rho'_{\hat{A}} \qquad (\dagger) \end{aligned}$$

Putting it all together we have:

$$\begin{aligned}
\rho'_A &= \rho'_{\hat{A}} && \text{By 4.1} \\
&= \rho'_{A'} + 1 && \text{By } \dagger \\
&\leq \rho_{A'} + 1 && \text{By 4.2} \\
&\leq \rho_{\hat{A}} && \text{By Claim 4.2.1} \\
&= \rho_A && \text{By 4.1}
\end{aligned}$$

and therefore $\rho'_A \leq \rho_A$ ending Case 1.

- Case 2: Suppose there is a proper covering.

  Pick a minimal proper covering, it consists of some rows (e many) and some columns (f many) and

  $$\rho'_A = e + f \tag{4.3}$$

  note that we have " $=$ " by minimality and since the chosen covering is proper, $e < m$ and $f < n$.

  Consider now $\hat{A} = PAQ$ where P permutes the e rows of the covering to the top, and Q permutes the f columns of the covering to the left, (for more details see Appendix A.3.1.1, and Appendix A.3.1.2).

$$\hat{A} = PAQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0 \end{array}\right],$$

where:

Sub Matrix	*Dimension*
E	$e \times f$
A_2	$(m-e) \times f$
A_1	$e \times (n-f)$
O	$(m-e) \times (n-f)$

Note that

$$\min\{m-e, f\} < \min\{m, n\} \quad \text{and} \quad \min\{e, n-f\} < \min\{m, n\}$$

by the fact that $(e < m) \wedge (f < n)$, and so, we can apply the I.H. to A_1 and A_2, therfore:

$$\rho'_{A_1} \leq \rho_{A_1} \quad \text{and} \quad \rho'_{A_2} \leq \rho_{A_2} \tag{4.4}$$

Suppose matrix A_1 can be covered with fewer than e lines, say $e' < e$.

Then the first f columns, together with e' lines, cover $\hat{A}$ giving us a covering of $\hat{A}$ of size $e' + f < e + f = \rho'_{\hat{A}}$ and hence a contradiction.

Since e lines cover A_1 and fewer lines cannot cover A_1, we have $\rho'_{A_1} = e$. Similarly

we show that f lines cover A_2, so, we have $\rho'_{A_2} = f$. Therefore,

$$\begin{aligned}
\rho'_A &= \rho'_{\hat{A}} && \text{By Permutations} \\
&= e + f && \text{By 4.3} \\
&= \rho'_{A_1} + \rho'_{A_2} && \text{By 4.4} \\
&\leq \rho_{A_1} + \rho_{A_2} && \text{By } \Pi^B_2\text{-I.H. } (**) \\
&\leq \rho_{\hat{A}} && \text{By } (*) \\
&= \rho_A && \text{By Permutations}
\end{aligned}$$

Where $(*)$ follows since if let S_1 be a subset of 1s of A_1, no two on the same line, and let S_2 be a subset of 1s of A_2, no two on the same lines, then $S = S_1 \cup S_2$ is a subset of 1s of A no two on the same line—as A_1 and A_2 "occupy" different lines of A. Finally, $(**)$ denotes the step of the proof in which we need to apply Π^B_2-Induction, called Π^B_2-I.H., i.e., it is here where we use the fact that our induction is over $\forall A \leq n \;\; \texttt{KMM}(A, n) \in \Pi^B_2$, where the predicate `KMM` expressed the Theorem 4.2.1, for more details see the next section.

So, we finish the proof of Case 2.

Finally, the Theorem König's Min-Max version II (4.2.1) follows. □

4.3 A feasible proof of König's Min-Max Theorem

In this section we present the main result of this thesis, that is, we show that the well-known König's Min-Max Theorem (KMM) can be proven in the first order theory **LA** with induction restricted to Σ^B_1 formulas. The standard proof ([BR91]) of

KMM requires Π_2^B induction, and hence does not yield feasible proofs—while our new approach does (it yields a Σ_1^B proof). In order to prove KMM in $\exists$**LA**, we need to restrict induction to Σ_1^B formulas. The main technical contribution is presented in Claim 4.3.4 Section 4.3.3. Basically, we introduce a polynomial time procedure, whose proof of correctness can be shown with $\exists$**LA**, that works as follow: given a matrix of size $e \times f$ such that $e \leq f$, where the minimum cover is of size e, our procedure computes a maximum selection of size e, row by row.

LA is a weak theory that essentially captures the ring properties of matrices; however, equipped with Σ_1^B induction **LA** is capable of proving KMM, and a host of other combinatorial theorems such as Menger's, Hall's and Dilworth's theorems. Therefore, our result formalizes Min-Max type of reasoning within a feasible framework.

4.3.1 Introduction

In this section we are concerned with the complexity of formalizing reasoning in combinatorial matrix theory. We are interested in the strength of the bounded arithmetic theories necessary in order to prove the fundamental results of this field (the *uniform* side of Proof Complexity Theory). We show, by introducing new proof techniques, that the theory of Bounded Arithmetic **LA** with induction restricted to bounded existential matrix quantification is sufficient to formalize a large portion of combinatorial matrix theory.

Perhaps the most famous theorem in combinatorial matrix theory is the König's Min-Max Theorem (KMM) which arises naturally in all areas of combinatorial algorithms — for example "network flows" with "Min-Cut Max-Flow" type of reasoning. See [Kön16a, Kön16b] for the original papers introducing KMM, and see [CD12] for

recent work related to formalizing proof of correctness of the Hungarian algorithm, which is an algorithm based on KMM. As far as we know, we give the first feasible proof of KMM.

As KMM is a cornerstone result in Combinatorial Matrix Theory, it has several counter-parts in related areas of mathematics: Menger's Theorem (Section 4.3.5.1), counting disjoint paths; Hall's Theorem (Section 4.3.5.2), giving necessary and sufficient conditions for the existence of a "system of distinct representatives" of a collection of sets; Dilworth's Theorem (Section 4.3.5.3), counting the number of disjoint chains in a poset, etc.

We show that KMM can be proven feasibly (Theorem 4.3.1), and we do so with a new proof of KMM that relies on introducing a new notion (Definition 4.3.1 and Claim 4.3.4, which introduce a polynomial time procedure, whose proof of correctness can be shown with $\exists\mathbf{LA}$). Furthermore, we show that the theorems related to KMM, and listed in the above paragraph, can also be proven feasibly; in fact, all these theorems are equivalent to KMM, and the equivalence can be shown in **LA** (Theorem 4.3.2). We believe that this captures the proof complexity of Min-Max reasoning.

Our results show that Min-Max reasoning can be formalized with uniform Extended Frege. It would be very interesting to know whether the techniques recently introduced by [HT11] could bring the complexity further down to quasi-polynomial Frege.

4.3.2 Background

Let A be a matrix of size $n \times m$ with entries in $\{0, 1\}$, what we sometimes call a 0-1 matrix. A *line* of A is an entire row or column of A; given an entry A_{ij} of A (when giving **LA** formulas we shall denote such an entry with $A(i, j)$), we say that a

line *covers* that entry if this line is either row i or column j. Then KMM states the following: the minimal number of lines in A that cover all of the 1s in A is equal to the maximal number of 1s in A with no two of the 1s on the same line. Note that KMM is stated for $n \times m$ matrices, but for simplicity we shall state it for $n \times n$ (i.e., square) matrices; of course, all results given in this Section (4.3) hold for both.

See [BR91, pg. 6] for a classical discussion and proof of KMM (and note that the proof relies, implicitly, on a Π_2^B type of induction, for more details about this proof see Section 4.2, and Appendix A.1).

We give the first, as far as we know, feasible proof of KMM in the logical theory **LA** defined in [Sol01]. By restricting the induction to be over Σ_1^B formulas, that is formulas whose prenex form consists of a block of bounded existential matrix quantifiers—and no other matrix quantifiers—we manage to prove KMM in a fragment of **LA** called $\exists$**LA**. While the matrices in the statements of the theorems have $\{0, 1\}$ entries, we assume that the underlying ring is $\mathbb{Z}$, the set of integers, so, the theorems of **LA** translate into $\mathbf{TC}^0$-Frege while the theorems of $\exists$**LA** translate into extended Frege, [SC04, §6.5]. We require the integers as one of our fundamental operations will be counting the number of 1s in a 0-1 matrix, i.e., computing ΣA, the sum of all the entries of A.

The background for **LA** is given in a Section 4.1.2 and Appendix A.5—for more details of **LA**-Theory see [Sol01].

The main contribution of this section is to show that KMM can be proven in the theory $\exists$**LA** which implies that it can be proven feasibly. We mention here an important observation of Jeřábek from [Je05, pg. 44]: $\exists$**LA** does not necessarily translate into a polytime proof system (e.g., extended Frege) when the matrices are

over $\mathbb{Z}$. However, if we restrict the quantified matrices to be over $\{0,1\}$, which is what we do in our proofs, it readily translates into extended Frege.

We use $|A| \leq n$ to abbreviate $r(A) \leq n \wedge c(A) \leq n$, that is, the number of rows of A is bounded by n, and the number of columns of A is bounded by n. We let $(\exists A \leq n)\alpha$ — resp. $(\forall A \leq n)\alpha$ — abbreviate $(\exists A)[|A| \leq n \wedge \alpha]$ — resp. $(\forall A)[|A| \leq n \rightarrow \alpha]$. These are *bounded matrix quantifiers*.

While **LA** allows for bounded index quantification and arbitrary matrix quantification, its induction is restricted to be over formulas without matrix quantifiers, i.e., over $\Sigma_0^B = \Pi_0^B$ formulas. On the other hand, in $\exists$**LA** we allow induction over so called Σ_1^B formulas. These are formulas, which when presented in prenex form, contain a single block of bounded existential matrix quantifiers. The set of formulas Π_1^B is defined similarly, except the block of quantifiers is universal.

In general, Σ_i^B is the set of formulas which, when presented in prenex form, start with a block of bounded existential matrix quantifiers, followed by a block of bounded universal matrix quantifiers, with i such alternating blocks. The set Π_i^B is the same, except it starts with a block of universal matrix quantifiers.

It follows more or less directly that our **LA** results can also be formalized in the theory $\mathbf{VTC}^0$ (and vice versa), defined in [CN10, pg. 283]. The reason is that the function ΣA is exactly Buss' function Numones(A) ([Bus86] and [Bus90, pg. 6]), i.e., the function that counts the number of 1s in A, and $\mathbf{TC}^0$ is the $\mathbf{AC}^0$ closure of Numones, [CN10, Proposition IX.3.1]. On the other hand, our $\exists$**LA** results can also be formalized in $\mathbf{V}^1$, defined in [CN10, pg. 133].

The main contribution of this section can now be stated more precisely: following König's original proof of KMM, which is also the standard presentation of the proof

in the literature (see the seminal work in the field [BR91, pg. 6]) one can construct a proof with Π_2^B induction, which does not yield translations into extended Frege proofs (for more details see Appendix A.1). On the other hand, we are able to give a proof that uses only Σ_1^B induction, which do yield extended Frege proofs, and thereby a feasible proof of KMM. Our insight is that while we are doing induction over the size of matrices, we can pre-arrange our matrices in a way that lowers the complexity of the induction. This is accomplished with the procedure outlined in the Definition 4.3.1—the diagonal property for matrices—and the subsequent proof of Claim 4.3.1.

The language of **LA** is well suited to express concepts in combinatorial matrix theory. We show how to express the concepts necessary to state KMM in the language $\mathcal{L}_{\mathbf{LA}}$. First, we say that the matrix α is a *cover* of a matrix A with the predicate:

$$\text{Cover}(A, \alpha) := \forall i, j \leq r(A)(A(i, j) = 1 \rightarrow \alpha(1, i) = 1 \vee \alpha(2, j) = 1) \tag{4.5}$$

We allow some leeway with notation: $\forall i, j \leq r(A)$ is of course $(\forall i \leq r(A))(\forall j \leq r(A))$. The matrix α keeps track of the lines that cover A; it does so with two rows: the top row keeps track of the horizontal lines, and the bottom row keeps track of the vertical line. The condition ensures that any 1 in A is covered by some line stipulated in α.

The next predicate expresses that the matrix β is a selection of 1s of A so that no two of these ones are on the same line. Thus, β can be seen as a "subset" of a permutation matrix; that is, each β is obtained from some permutation matrix by deleting some (possibly none) of the 1s. We say that β is a *selection* of A, and it is

given with the following formula:

$$\begin{aligned}\text{Select}(A,\beta) := \forall i,j \leq r(A)((\beta(i,j)=1 \rightarrow A(i,j)=1)\\ \wedge \forall k \leq r(A)(\beta(i,j)=1 \rightarrow \beta(i,k)=0 \wedge \beta(k,j)=0))\end{aligned} \tag{4.6}$$

We are interested in a minimum cover (as few 1s in α as possible) and a maximum selection (as many 1s in β as possible). The following two predicates express that α is a minimum cover and β a maximum selection.

$$\text{MinCover}(A,\alpha) := \tag{4.7}$$
$$\text{Cover}(A,\alpha) \wedge \forall \alpha' \leq c(\alpha)(\text{Cover}(A,\alpha') \rightarrow \Sigma\alpha' \geq \Sigma\alpha)$$
$$\text{MaxSelect}(A,\beta) := \tag{4.8}$$
$$\text{Select}(A,\beta) \wedge \forall \beta' \leq r(\beta)(\text{Select}(A,\beta') \rightarrow \Sigma\beta' \leq \Sigma\beta)$$

Clearly MinCover and MaxSelect are Π_1^B formulas. We can now state KMM in the language of $\mathcal{L}_{\mathbf{LA}}$ as follows:

$$\text{MinCover}(A,\alpha) \wedge \text{MaxSelect}(A,\beta) \rightarrow \Sigma\alpha = \Sigma\beta \tag{4.9}$$

Note that (4.9) is a Σ_1^B formula. The reason is that in prenex form, the universal matrix quantifiers in MinCover and MaxSelect become existential as we pull them out of the implication; they are also bounded.

Given a matrix A, its *n-th principal minor* consists of A with the first $r(A)-n$ rows deleted, and the first $c(A)-n$ columns deleted. For instance, for a square matrix A, when $n=|A|$, the n-th submatrix is just A, and when $n=1$, then n-th submatrix

is just $[A_{|A|,|A|}]$, i.e., the matrix consisting of just the lower-right entry. Let $A[n]$ denote the n-th principal minor, and note that $A[n]$ can be expressed as follows in the language of **LA**: $\lambda ij\langle n, n, e(A, r(A) - n + i, c(A) - n + j)\rangle$.

Let $\mathrm{KMM}(A, n)$ be the following Σ_1^B formula: it is a conjunction of the statement that A is an $n \times n$ matrix, which we abbreviate informally as $|A| = n$, and which in $\mathcal{L}_{\mathbf{LA}}$ is stated as $r(A) = n \wedge c(A) = n$, more precisely, the n-th submatrix of A, i.e., A replaced by $A[n]$, and (4.9) in prenex form:

$$\mathrm{KMM}(A, n) := |A| = n \wedge \mathrm{prenex}(4.9). \tag{4.10}$$

Given a matrix A, let l_A and o_A denote the minimum number of lines necessary to cover all the 1s of A, and the maximum number of 1s no two on the same line, respectively. (Of course, König's theorem says that for all A, $l_A = o_A$.) In terms of the definitions just given, we have that $l_A = \Sigma\alpha$ where $\mathrm{MinCover}(A, \alpha)$, and $o_A = \Sigma\beta$ where $\mathrm{MaxSelect}(A, \beta)$.

Finally, the fact that P is a permutation matrix can be stated easily with a predicate free of matrix quantification, i.e., a Σ_0^B predicate. The original **LA**-Theory [Sol01] has no index quantification. When we use in our formulas the concept of permutation matrices, we need have bounded index quantification, nevertheless, it turns out that the translation over $\mathbb{Z}$ is into $\mathbf{TC}^0$-Frege proofs. (For this result see [SC04, §6.5].) Using bounded index quantification in **LA** we state that a given matrix P is a permutation matrix:

$$\begin{aligned}&[r(P) = c(P)] \ \wedge [(\forall i \leq r(P))(\exists! j \leq c(P))e(i, j, P) = 1] \\ &\wedge [(\forall j \leq c(P))(\exists! i \leq r(P))e(i, j, P) = 1]\end{aligned} \tag{4.11}$$

as the entries are in $\{0,1\}$, if $e(i,j,P) \neq 1$, it follows that $e(i,j,P) = 0$. Let 4.11 abbreviated by Perm(P). Where, $\exists! x\ \alpha(x)$ stands for

$$\exists x \forall y\ [(\alpha(y) \supset x = y) \wedge (x = y \supset \alpha(y))].$$

See for more details [Sol04]. Alternatively, it can be stated with a Σ_0^B predicate that a matrix P is a permutation matrix as follows:

$$\text{Perm}(P) := (\forall i \leq |P| \exists j \leq |P| P_{ij} = 1) \wedge (\forall i, j \neq k \leq |P| (P_{ij} = 0 \vee P_{ik} = 0)).$$

4.3.3 The main result

With the basic machinery in place, we prove the main theorem with a sequence of Lemmas.

Theorem 4.3.1 $\exists$**LA** $\vdash$ KMM.

What this Theorem says is that KMM can be shown in **LA** with Σ_1^B induction, and thus in uniform extended Frege, which in turn means feasibly. We prove KMM for any matrix A by induction on the principal minors of A. The rest of this section consists of a proof of this theorem.

Some of the intermediate results can be shown with just **LA** induction (i.e., induction over formulas without matrix quantifiers, that is, over formulas in $\Sigma_0^B = \Pi_0^B$). We use the weaker theory whenever possible.

We start by showing the following technical Lemma which states that l_A and o_A are invariant under permutations of rows and columns.

Lemma 4.3.1 *Given a matrix A, and given any permutation matrix P, we have that*

- $\mathbf{LA} \vdash l_{PA} = l_{AP} = l_A$
- $\mathbf{LA} \vdash o_{PA} = o_{AP} = o_A$

That is, these four equalities can be proven in **LA**, i.e., with induction restricted to formulas without matrix quantifiers.

PROOF: **LA** shows that if we reorder the rows or columns (or both) of a given matrix A, then the new matrix, call it A', where $A' = PA$ or $A' = AP$, has the same size minimum cover and the same size maximum selection. Of course, we can reorder both rows and columns by applying the statement twice: $A' = PA$ and $A'' = A'Q = PAQ$.

The first thing that we need to show is that:

- $\mathbf{LA} \vdash \text{Cover}(A, \alpha) \rightarrow \text{Cover}(A', \alpha')$
- $\mathbf{LA} \vdash \text{Select}(A, \beta) \rightarrow \text{Select}(A', \beta')$

where A' is defined as in the above paragraph, and α' is the same as α, except the first row of α is now reordered by the same permutation P that multiplied A on the left (and the second row of α is reordered if P multiplied A on the right). The matrix β is even easier to compute, as $\beta' = P\beta$ if $A' = PA$, and $\beta' = \beta P$ if $A' = AP$. It follows from P being a permutation matrix that $\Sigma\alpha = \Sigma\alpha'$ and $\Sigma\beta = \Sigma\beta'$: we can show by **LA** induction on the size of matrices[6] that if X' is the result of rearranging X (i.e., $X' = PXQ$, where P, Q are permutation matrices), then $\Sigma X = \Sigma X'$. We do so first on X consisting of a single row, by induction on the length of the row. Then we take the single row as the basis case for induction over the number of rows of a general X.

[6]For more about this kind of induction over size of matrices on **LA**-Theory see Appendix A.1.

It is clear that given A', the cover α' has been adjusted appropriately; same for the selection β'. We can prove it formally in **LA** by contradiction: suppose some 1 in A' is not covered in α'; then the same 1 in A would not be covered by α. For the selections, note that reordering rows and/or columns we maintain the property of being a selection: we can again prove this formally in **LA** by contradiction: if β' has two 1s on the same line, then so would β.

The next thing to show is that

- $\mathbf{LA} \vdash \text{MinCover}(A, \alpha) \rightarrow \text{MinCover}(A', \alpha')$
- $\mathbf{LA} \vdash \text{MaxSelect}(A, \beta) \rightarrow \text{MaxSelect}(A', \beta')$

and the reasoning that accomplishes this is by contradiction. As permuting only reorders the matrices (it does not add or take away 1s), if the right-hand side does not hold, we would get that the left-hand side does not hold by applying the inverse of the permutation matrix.

All these arguments can be easily formalized in **LA**, and we leave the details in Appendix A.2. □

The next definition is a key concept in the Σ_1^B proof of KMM.

Definition 4.3.1 *We say that an $n \times n$ matrix over $\{0,1\}$ has the* diagonal property *if for each diagonal entry (i,i) of A, either $A_{ii} = 1$, or $(\forall j \geq i)[A_{ij} = 0 \wedge A_{ji} = 0]$.*

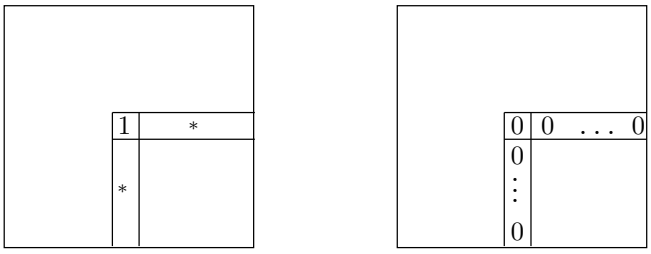


Figure 4.1: Either $A_{ii} = 1$ or $(\forall j \geq i)[A_{ij} = 0 \wedge A_{ji} = 0]$.

Claim 4.3.1 *Given any matrix A, $\exists$**LA** proves that there exist permutation matrices P, Q such that PAQ has the diagonal property.*

Proof: We construct P, Q inductively on $n = |A|$. Let the *i-th layer of A* consist of the following entries of A: A_{ij}, for $j = i, \ldots, n$ and A_{ji} for $j = i+1, \ldots, n$. Thus, the first layer consists of the first row and column of A, and the n-th layer (also the last layer), is just A_{nn}. We transform A by layers, $i = 1, 2, 3, \ldots$. At step i, let A' be the result of having dealt already with the first $i-1$ layers. If $A'_{ii} = 1$ move to the next layer, $i+1$. Otherwise, find a 1 in layer i of A'. If there is no 1, also move on to the next layer, $i+1$. If there is a 1, permute it from position $A_{ij'}$, $j' \in \{i, \ldots, n\}$ to A'_{ii}, or from position $A_{j'i}$, $j' \in \{i+1, \ldots, n\}$. Note that such a permutation does not disturb the work done in the previous layers; that is, if A'_{kk}, $k < i$, was a 1, it continues being a 1, and if it was not a 1, then there are no 1s in layer k of A'. Note that each layer can be computed independently of the others. □

It is Claim 4.3.1 that allows us to bring down the complexity of the proof of KMM from Π^B_2 to Σ^B_1. As we shall see, by transforming A into A', so that $A' = PAQ$

where P, Q are permutation matrices and A' has the diagonal property, we can prove KMM for A' with just Σ_1^B induction, and then by Lemma 4.3.1 we obtain an $\exists$**LA** proof of KMM for A. All of this is made precise in the following Lemma; recall that $\mathrm{KMM}(A, n)$ is defined in (4.10).

Lemma 4.3.2 $\exists$**LA** $\vdash \forall n \mathrm{KMM}(A, n)$.

We are going to prove Lemma 4.3.2 by induction on n, breaking it down into Claims 4.3.2 and 4.3.3. Once we have that $\forall n \mathrm{KMM}(A, n)$ is enough to prove KMM for A since letting $n = |A|$ we obtain $A[n] = A$. Therefore obtain an $\exists$**LA** proof of (4.9), and thereby a proof of Theorem 4.3.1.

From Lemma 4.3.1 and Claim 4.3.1 we know that it is sufficient to prove Lemma 4.3.2 for appropriate PAQ, which ensures the diagonal property spelled out in Claim 4.3.1. Thus, in order to simplify notation, we assume that our A is the result of applying the permutations; i.e., A has the diagonal property.

Claim 4.3.2 **LA** $\vdash o_A \leq l_A$.

Proof: Given a covering of A consisting of l_A lines, we know that every 1 we pick for a maximal selection of 1s has to be on one of the lines of the covering. We also know that we cannot pick more than one 1 from each line. Thus, the number of lines in the covering provide an upper bound on the size of such selection, giving us $o_A \leq l_A$.

We can formalize this argument in **LA** as follows: let $\mathcal{A}$ be an $l_A \times o_A$ matrix whose rows represent the lines of a covering, and whose columns represent the 1s no two on the same line. Let $\mathcal{A}(i, j) = 1 \iff$ the line labeled with i covers the 1 labeled

with j. Then,

$$o_A = c(\mathcal{A}) \leq \Sigma\mathcal{A} \tag{a}$$

$$= \Sigma_i(\Sigma\lambda pq\langle 1, c(\mathcal{A}), \mathcal{A}(i,q)\rangle) \tag{b}$$

$$\leq \Sigma_i 1 = r(\mathcal{A}) = l_A, \tag{c}$$

where the inequality in the line labeled by (a) can be shown by induction on the number of columns of $\mathcal{A}$ which has the condition that each column contains at least one 1(i.e., each 1 from the selection must be covered by some line); (b) follows from the fact that we can add all the entries in a matrix by rows; and (c) can be shown by induction on the number of rows of $\mathcal{A}$ which has the condition that each row contains at most one 1 (i.e., no two 1s from the selection can be on the same line). □

We briefly discuss the implications of Claim 4.3.2 for the provability of variants of the pigeonhole principle (PHP) in **LA**. We showed that if we have a set of n items $\{i_1, i_2, \ldots, i_n\}$ and a second set of m items $\{j_1, j_2, \ldots, j_m\}$, and we represent the matching by A as follows: $A(p,q) = 1 \iff i_p \mapsto j_q$, then injectivity means that each column of A has at most one 1. Thus:

$$n \leq \Sigma A = \Sigma_i(\text{col } i \text{ of } A) \leq \Sigma_i 1 \leq m.$$

This is to be expected as we already mentioned that **LA** over $\mathbb{Z}$ corresponds to $\mathbf{VTC}^0$, which proves PHP.

Bondy's Theorem states that for any $n \times n$ 0-1 matrix whose rows are distinct, we can always delete a column so that the remaining $n \times (n-1)$ matrix still has n distinct rows. [SC04, §IX.3.8] investigate the connection between Bondy's Theorem

(**BONDY**) and PHP, and they show that $\mathbf{V}^0 \vdash$ **BONDY** $\leftrightarrow$ PHP. It would be interesting to know if $\mathbf{V}^0 \vdash$ KMM $\leftrightarrow$ PHP.

As Claim 4.3.2 shows, **LA** is sufficient to prove $o_A \leq l_A$; on the other hand, we seem to require the stronger theory $\exists$**LA** (which is **LA** with induction over Σ_1^B formulas) in order to prove the other direction of the inequality.

Claim 4.3.3 $\exists$**LA** $\vdash o_A \geq l_A$.

PROOF: Let

$$A = \left[\begin{array}{c|c} a & R \\ \hline S & M \end{array}\right], \tag{4.12}$$

where a is the top-left entry, and M the principal sub-matrix of A, and R (resp. S) is $1 \times (n-1)$ (resp. $(n-1) \times 1$). By Claim 4.3.1 we can ensure that A has the diagonal property (see Definition 4.3.1), which simplifies the analysis of the cases. Indeed, from the diagonal property we know that one of the following two cases is true:

Case 1. $a = 1$

Case 2. a, R, S consist entirely of zeros

In the second case, $o_A \geq l_A$ follows directly from the induction hypothesis, $o_M \geq l_M$, as $o_A = o_M \geq l_M = l_A$. Thus, it is the first case, $a = 1$, that is interesting. The first case, in turn, can be broken up into two subcases: $l_M = n - 1$ and $l_M < n - 1$.

Subcase (1-a) $l_M = n - 1$

By induction hypothesis, $o_M \geq l_M = n - 1$. We also have that $a = 1$, and a is in position $(1, 1)$, and hence no matter what subset of 1s is selected from M, none of them lie on the same line as a. Therefore, $o_A \geq o_M + 1$. Since $o_M \geq n - 1$, $o_A \geq n$, and since we can *always* cover A with n lines, we have that $n \geq l_A$, and so $o_A \geq l_A$.

Subcase (1-b) $l_M < n - 1$

We start with the following definition.

Definition 4.3.2 *Let A and M be as in (4.12), and let α_M be a set of lines of M, i.e., α_M consists of rows $i_1, i_2, \ldots, i_k$, and columns $j_1, j_2, \ldots, j_\ell$. The* extension *of α_M to A, denoted by $\hat{\alpha}_M$, is simply the set of rows $i_1 + 1, i_2 + 1, \ldots, i_k + 1$, and the set of columns $j_1 + 1, j_2 + 1, \ldots, j_\ell + 1$.*

We say that a minimal cover α_A is proper if it does not consists entirely of all the rows or of all the columns of A; that is, α_A is proper if it is minimal, i.e., $|\alpha_A| = l_A$, and each row of α_A has at least one zero. If $l_M < n - 1$, then we know that α_A has a proper cover, as we can always cover A with $\hat{\alpha}_M$ plus the first row and column of A.

Let α_A be a proper minimal cover of A, and let P, Q be two permutations that place all the rows of the cover in the initial position, and place all the columns of the cover in the initial position—Figure 4.2 illustrates this.

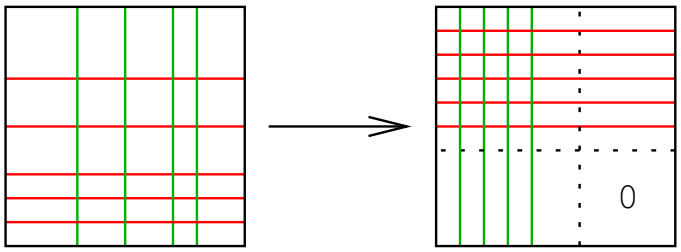


Figure 4.2: Permuting the rows and columns of the cover to be in initial positions.

Now suppose that α_A consists of e rows and f columns (in the diagram, e horizontal lines and f vertical lines). Clearly $l_A = e + f$. The rearranging of A produces four quadrants; the lower-right quadrant, of size $(|A| - f) \times (|A| - e)$, consists entirely of zeros (since no lines cross it), and since α_A is proper, we know that it is not empty.

The upper-right quadrant is of size $e \times (|A| - f)$, and it cannot be covered by fewer than e lines. The lower-left quadrant is of size $(|A| - e) \times f$ and cannot be covered by fewer than f lines.

Claim 4.3.4 $\exists$**LA** *shows that if X is an $e \times h$ matrix, and $l_X = e$, then $o_X \geq e$.*

PROOF: We state the claim formally as follows:

$$[\forall \alpha \leq r(A)\text{Cover}(A, \alpha) \rightarrow \Sigma\alpha \geq r(A)] \rightarrow [\exists \beta \leq r(A)\text{Select}(A, \beta) \wedge \Sigma\beta \geq r(A)]$$

and we prove it by induction on the number of rows of A. To this end, let A_n denote the first n rows of A, so that $A_{r(A)} = A$. We now prove the Σ_1^B formula:

$$\exists \alpha, \beta \leq n \left[(\text{Cover}(A_n, \alpha) \wedge \Sigma\alpha < n) \vee (\text{Select}(A_n, \beta) \wedge \Sigma\beta \geq n)\right],$$

which is equivalent to the formula above it for $n = r(A)$. The claim holds for $n = 1$, as in that case we have a single row, which is either zero and hence has a cover of size 0, or the row has a 1, in which case we can select it. For the induction step, suppose the claim holds for $n = k$. Suppose that any cover for A_{k+1} requires $k + 1$ rows. Then, A_k requires k rows (for otherwise, a cover of A_k of size $< k$ plus row $k + 1$ would give a cover of size $\leq k$ of A_{k+1}, which is a contradiction). By IH, A_k has a selection of size at least k.

More explicitly, we construct a section $\mathcal{S}$ by induction on the rows of A_k. In the first row there must be a 1, since otherwise $e - 1$ lines cover A_k. Pick any 1 in the first row, that is, $\mathcal{S} = \{1, \ell_1\}$, where ℓ_1 is the column of that 1. Suppose we picked 1s from the first k rows, giving us a selection from A_k, $\mathcal{S} = \{(1, \ell_1), (2, \ell_2), \ldots, (k, \ell_k)\}$. Let $C_{\mathcal{S}}$ be the set of k vertical lines going through $\mathcal{S}$. Consider row $k + 1$; we know that this

row cannot be empty (since otherwise $e-1$ lines would be a cover of A_k). If there is a 1 in row $k+1$ not covered by $C_{\mathcal{S}}$, then select that 1. Otherwise, suppose that there are $p > 0$ 1s in row $k+1$; label their columns as $c_1, c_2, \ldots, c_p$. Let $\rho(c_i) = r_i$ be a mapping where r_i is the row with the unique 1 in $\mathcal{S}$ such that $\ell_{r_i} = c_i$.

† Let $\rho_i = \{(k+1, c_i), (r_i, c_i), (r_i, x_1), (y_1, x_1), (y_1, x_2), \ldots, (a, b)\}$, so that each position has a 1 in A_{k+1}, and in particular (a, b) corresponds to a 1 not covered by $C_{\mathcal{S}}$. Then, ρ_i describes a re-arrangement of the selection. A ρ_i with (a, b) not covered by $C_{\mathcal{S}}$ must exist, since otherwise we would have again a cover of size $e-1$. See Figure 4.3 for an illustration.

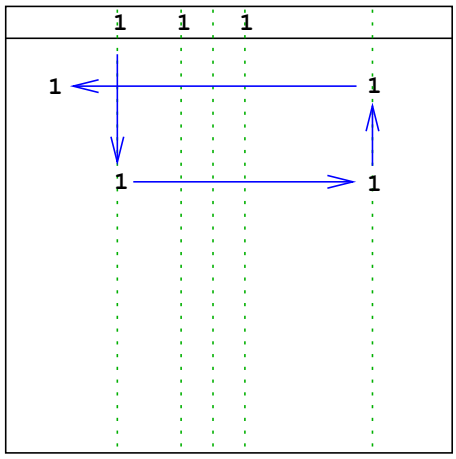


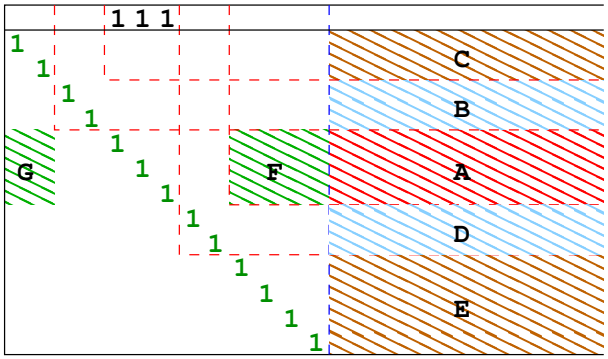
Figure 4.3: ρ_1 consisting of five positions.

Now we expand the last paragraph in that proof of Claim 4.3.4, denoted by †'.

The induction is on the number of rows, where A_k contains the first k rows of A (this is not crucial, but we count the rows from the bottom). We denote the first row of A (in the usual sense, the top row), as row $k+1$. If row $k+1$ is *not* covered by $C_{\mathcal{S}}$ (which consists of the vertical lines going through the points in $\mathcal{S}$), then we pick a 1 from row $k+1$ that is not covered and add it to our selection. This is the easy case;

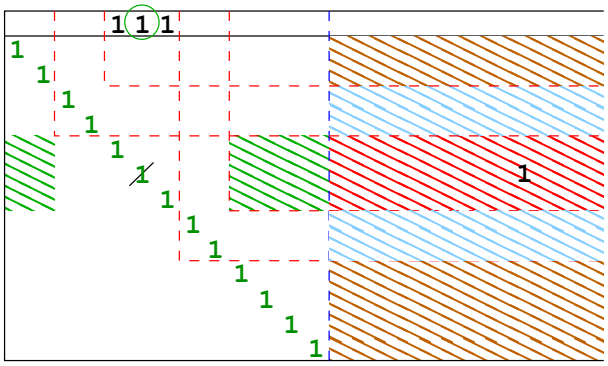
we now assume that C_s does cover row $k+1$.

We re-arrange A_{k+1} by permuting its rows and columns so that it is of the following form:



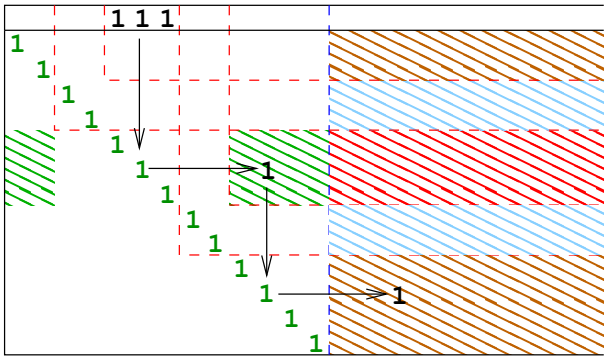
The re-arrangement of A_{k+1}, giving it the above form, is obtained as follows: first, we place the selected 1s (the green 1s) from A_k along the main diagonal of A_k. This permutation might rearrange the columns of row $k+1$, but row $k+1$ continues being in the same position — the first row from the top. We then re-arrange the green 1s so that areas B and D contain only zeros (note that these areas might be empty).

Suppose first that area A contains a 1; say that this 1 is in row i of the matrix. Then pick the green 1 along row i, remove it from the selection, and instead add to the selection the corresponding black 1 (i.e., the 1 in row $k+1$ that is in column $i-1$), and the 1 from area A. The diagram below represents the situation: the black 1 in row $k+1$ with the green circle around it is the selection from row $k+1$. The crossed-out green 1 has be de-selected, and replaced by the black 1 in area A.



Suppose that area A is all zeros. Suppose that areas G and F are also all zeros; then we can cover A_{k+1} with k lines as follows: horizontal lines covering C,B,D,E and vertical lines covering the green 1s between G and F. But this contradicts the assumption that A_{k+1} requires $k+1$ lines to be covered.

Thus, under the assumption that area A is all zeros, we know that there must be 1s in area G or area F. As G and F are symmetric cases, assume that the 1 is in F. Then we re-arrange the selection as in the following diagram.



Following the direction of the path, the black 1 from row $k+1$ is selected; the green 1 is deleted and replaced by the black 1 in area F; the second green 1 on the path is deleted, and replaced by the black 1 in area E. Also note that if E were empty, so would be F.

This ends the proof of Claim 4.3.4. □

Since the size of selections is invariant under permutations, it follows that $o_A \geq e + f = l_A$. This ends the proof of Claim 4.3.3. □

4.3.4 Induced Algorithm

The standard KMM theorem is stated as an implication (see (4.9)), and hence it makes no assertions about the actual existence of a minimal covering or maximal selection of 1s, let alone how to compute them. It only says that if they do exist, they are equal. However, the proof of Lemma 4.3.2 suggests an algorithm for computing both.

Note that computing a minimal cover can be accomplished in polytime with the well-known Hopcroft-Karp (HK)-Algorithm (see [JEH73], and Chapter 2 Section 2.3) as follows: First use the HK-Algorithm to compute a "maximal matching," which in this case is simply a maximal selection of 1s (when we view A—in the natural way—as the adjacency matrix of a bipartite graph). In [SF12], and Chapter 3, we show how to convert, in linear time, a maximal selection into a minimal cover.

Certainly the correctness of the algorithms mentioned in the above paragraph can be shown in $\exists$**LA** (as it captures polytime reasoning —see [Sol01]), and so it follows that we can prove in $\exists$**LA** the existence of a minimal cover and maximum selection. Therefore, $\exists$**LA** can prove something stronger than (4.9). Namely, it can not only show that *if* we have a minimal cover and a maximal selection, then they have the same size, but rather, that there *always exists* a minimal cover and maximal selection, and the two are of equal size. We mention this important remark in order to emphasize that there is a difference between the following two statements:

- *if* minimum cover and maximum selection exists, then they are equal, which is

how it is KMM stated in equation 4.9.

- minimum cover and maximum selection exists and are equal, which is what we proved.

However, instead of doing the heavy lifting necessary to formalize the correctness of HK and [SF12] in $\exists$**LA** (a correctness of [SF12] is presented in Appendix A.3), we present a new simple polytime algorithm for computing minimal covers based on the proof of Lemma 4.3.2. Note that a similar argument would show the existence of a polytime algorithm for maximal selection.

The algorithm works as follows: given a 0-1 matrix A, we first put A in the diagonal property (see Definition 4.3.1). We now work with A which is assumed to be in diagonal property and proceed by computing recursively l_M, the size of a minimal cover of M, where M is the principal submatrix of A. Keeping in mind the form of A given by (4.12), we have the following cases:

Case 1. If $a = 0$ (in which case R, S are also zero, by the fact that A has been put in diagonal property), then $l_A = l_M$, and proceed to compute the minimal cover α_M of M; output $\hat{\alpha}_A$, the extension of α_M (see Definition 4.3.2).

Case 2. If $a \neq 0$, we first examine R to see if the matrix M', consisting of the columns of M minus those columns of M which correspond to 1s in R, has a cover of size $l_M - \Sigma R$ (of course, if $l_M < \Sigma R$, then the answer is "no").

If the answer is "yes", compute the minimal cover of M', $\alpha_{M'}$. Then let α_M be the cover of M consisting of the lines in $\alpha_{M'}$ properly renamed to account for the deletion of columns that transformed M into M', plus the columns of M corresponding to the 1s in R. Then, α_A is the result of extending α_M and adding the first column of A.

Note that both α_M and α_A can be defined with the 0-1 matrix α (the two-row α

that encode the cover).

If the answer is "no", repeat the same with S: let M' be the result of subtracting from M the rows corresponding to the rows with 1s in S. Check whether M' has a cover of size $l_M - \Sigma S$. If the answer is "yes" then build a cover for A as in the R-case, i.e., α_A is the result of extending α_M and adding the first row of A.

If the answer is "no", then compute any minimal cover for M, extend it to A, and add the first row and column of A; this results in a cover for A.

At the end, we apply the permutations P, Q that converted A to the diagonal property, to the final C, and output that as the minimal cover of the original A. As was mentioned above, a similar polynomial time recursive algorithm can compute a maximal selection of 1s. The algorithm works as follows: given a 0-1 matrix A, we put A in diagonal property and proceed by computing recursively o_M, the size of maximal selection of 1s no two on the same line of M, where M is the principal submatrix of A.

Like the previous algorithm, we keep in mind the form of A given by (4.12). We have the following cases:

Case 1. If $a = 0$ then $o_A = o_M$ by diagonal property, then we compute the maximal selection—here we use the HK-Algorithm—L_M of M, i.e., L_M is a set of ones no two on the same row or column; output L_A which is the extension (see Definition 4.3.2) of L_M to A.

Case 2. If $a \neq 0$ then we first examine R to see if the matrix M' formed by the columns of M minus the non-zero columns of R, has a selection of size $o_M - \Sigma R$, again, if $o_M < \Sigma R$ then M' has not such selection, therefore we output the non-zero entries of R plus a.

On the other hand, if M' has a selection of size $o_M - \Sigma R$ we compute the maximal

selection of M', denoted by $L_{M'}$. Then let L_M the selection of M formed by the 1s in $L_{M'}$, plus the columns of M corresponding the 1s in R. Finally, L_A is the result of extending L_M and adding the fist column of A.

If M' has not a selection of size $o_M - \Sigma R$, we proceed in the same way that before but with the submatrix S; let M' be the result of subtracting from M the rows corresponding to the 1s of S. Check whether M' has a selection of size $o_M - \Sigma S$, again, if $o_M < \Sigma S$ then M' has not such selection, therefore we output the non-zero entries of S plus a.

On the other hand, if M' has such selection, then we build a selection of A as in the R-case, but adding the first row of A.

Now, if M' has no a selection of size $o_M - \Sigma S$, then compute *any* maximal selection for M, extend it to A, plus the first row and the first column of A; this results in a selection of A.

At the end, we apply the permutations P, Q to put A in diagonal property, to the final selection L, and output that as a maximal selection of 1s no two on the same line in A.

4.3.5 Related theorems

In this section we are going to prove that the various reformulations of KMM, arising in graph theory and partial orders, can be proven equivalent to KMM in low complexity (**LA**), and therefore they also have feasible proofs. We state this as the following theorem:

Theorem 4.3.2 *The theory* **LA** *proves the equivalence of* KMM*, Menger's, Hall's and Dilworth's Theorems.*

The proof of consists of Lemmas 4.3.3 and 4.3.4, showing the equivalence of KMM and Menger's Theorem in Subsection A; Lemmas 4.3.5 and 4.3.6, showing the equivalence of KMM and Hall's Theorem in Subsection B; Lemmas 4.3.7 and 4.3.9, showing the equivalence of KMM and Dilworth's Theorem in Subsection C. Each subsection starts with a description of how to formalize the given Theorem, followed by the two Lemmas giving the two directions of the equivalence.

4.3.5.1 Menger's Theorem

Given a graph $G = (V, E)$, an x, y-*path* in G is a sequence of distinct vertices $v_1, v_2, \ldots, v_n$ such that $x = v_1$ and $y = v_n$ and for all $1 \leq i < n, (v_i, v_{i+1}) \in E$. The vertices $\{v_2, \ldots, v_{n-1}\}$ are called *internal vertices*; we say that two x, y-paths are *internally disjoint* if they do not have internal vertices in common.

We also say that $S \subseteq V$ is an x, y-*cut* if there is no path from x to y in the graph $G' = (V - S, E')$, where E' is the subset of those edges in E which have no end-point in S.

Let $\kappa(x, y)$ represent the size of the smallest x, y-cut, and let $\lambda(x, y)$ represent the size of the largest set of pairwise internally disjoint x, y-paths.

Menger's theorem states that for any graph $G = (V, E)$, if $x, y \in V$ and $(x, y) \notin E$, then the minimum size of an x, y-cut equals the maximum number of pairwise internally disjoint x, y-paths. That is, $\kappa(x, y) = \lambda(x, y)$. For more details on Menger's Theorem turn to [Meng27, Go00, Pym96]. Menger's Theorem is of course the familiar Min-Cut Max-Flow Theorem where all edges have capacity 1.

We now show how to state Menger's theorem in $\mathcal{L}_{\mathbf{LA}}$. We start by defining the Σ_0^B predicate $\mathrm{Path}(A, x, y, \alpha)$, which states that α encodes the internal vertices of a

path from x to y in A. We define Path by parts; first we state that α has at most one 1 in each row and column:

$$(\forall l \leq n-2)[\Sigma\lambda_{ij}\langle 1, n-2, \alpha(l,j)\rangle = 1 \wedge \Sigma\lambda_{ij}\langle n-2, 1, \alpha(i,l)\rangle = 1] \tag{4.13}$$

Then we say that if the l-th node is p and $l+1$-th node is q, then there is an edge between p and q:

$$(\forall l, p, q \leq n-3)(\alpha(l,p) = 1 \wedge \alpha(l+1,q) = 1) \rightarrow A(p,q) = 1 \tag{4.14}$$

Note that in general different paths are of different lengths; this can be dealt with in a number of ways: for example, by padding α with repetitions of the last row (so that each α has exactly $n-2$ rows). We assume that this is what we do, and can be check that $\boldsymbol{\mathcal{L}_{\mathbf{LA}}}$ can express this easily, like follows:

$$\begin{aligned}(\forall l, p, q \leq n-3)(\alpha(l,p) = 1 \wedge \lambda_{ij}\langle 1, n-2, \alpha(l+1,j)\rangle = \lambda_{ij}\langle 1,1,0\rangle \wedge \\ \alpha(l+1,q) = 0) \rightarrow \lambda_{ij}\langle n-2-l, n-2, \alpha(i,j)\rangle = \lambda_{ij}\langle 1,1,\alpha(l,p)\rangle\end{aligned} \tag{4.15}$$

expressing that **if** l-th node (the last one) is p, and the $(l+1)$-th row of α is not the last row, i.e., $(n-2)$-th row and it is full of zeros, that is, in particular, w.l.g., the position $\alpha(l+1,q)$ is zero, **then** like the length of the path is shorter than $n-2$ we repeat to fill out $n-2$ intermediate nodes, i.e., $n-2-l$.

On the other hand, if i is the first intermediate node then $(x,i) \in E$, and if i is the last intermediate node then $(i,y) \in E$:

$$\alpha(1,i) = 1 \rightarrow A(x,i) = 1 \wedge \alpha(n-2,i) = 1 \rightarrow A(i,y) = 1 \tag{4.16}$$

Putting it all together, the Σ_0^B formula expressing Path is given by the conjunction of $A(x, y) = 0$ together with the above properties, i.e.,

$$\text{Path}(A, x, y, \alpha) := (4.13) \wedge (4.14) \wedge (4.16) \wedge A(x, y) = 0. \tag{4.17}$$

Finally, we state that two paths α, α' are internally disjoint:

$$\begin{aligned} \text{Disjoint}(A, x, y, \alpha, \alpha') := \quad & \text{Path}(A, x, y, \alpha) \wedge \text{Path}(A, x, y, \alpha') \\ & \wedge (\forall i \leq n-2 \forall j \leq n-2)(\alpha(i, j) \cdot \alpha'(i, j) = 0) \end{aligned} \tag{4.18}$$

We must be able to talk about a collection of paths; the 0-1 matrix β will encode a collection of paths $\alpha_1, \alpha_2, \ldots, \alpha_\lambda$:

$$\beta = \boxed{\beta[1] = \alpha_1 \mid \beta[2] = \alpha_2 \mid \ldots \mid \beta[\lambda] = \alpha_\lambda} \tag{4.19}$$

so that β is a matrix of size $(n-2) \times \lambda(n-2)$. Each $\beta[i]$ can be defined thus:

$$\beta[i] := \lambda_{pq}\langle n-2, n-2, \beta(p, (i-1)(n-2)+q)\rangle.$$

We are interested in pairwise disjoint collections of paths:

$$\begin{aligned} & \text{CollectDisj}(A, x, y, \beta, \lambda) := \\ & \forall i \leq \lambda \ \text{Path}(A, x, y, \beta[i]) \ \wedge (\forall i \neq j \leq \lambda) \ \text{Disjoint}(A, x, y, \beta[i], \beta[j]) \end{aligned} \tag{4.20}$$

The following formula expresses $\lambda(x, y)$ for a given A; note that it is a Π_2^B formula:

$$\begin{aligned}&\text{MaxDisj}(A, x, y, \lambda) :=\\&(\exists \beta \leq (n-2)\lambda)\ \text{CollectDisj}(A, x, y, \beta, \lambda) \wedge\\&(\forall \alpha \leq n-2)(\text{Path}(A, x, y, \alpha) \rightarrow \exists i \leq \lambda\ \alpha = \beta[i])\end{aligned} \tag{4.21}$$

Note that the previous formal definition of our 0-1 matrix β can be, alternatively and more simple, defined as follow:

Let β be a 0-1 matrix that encodes disjoint paths, such that, the rows of β correspond to the paths, and the columns to the vertices of G, where $\beta(i, j) = 1$ if path i contains vertex j. The disjointness can be stated by insisting that each column has at most one 1.

Likewise, we need to formalize $\kappa(x, y)$; we start with a 0-1 matrix γ expressing a cut in A:

$$\text{Cut}(A, \gamma) := (\forall i \leq n-2)(\forall j \leq n-2)(\gamma(i, j) = 1 \rightarrow A(i, j) = 1) \tag{4.22}$$

which says that every edge of γ is an edge of A, and it defines the cut implicitly as the set of edges in A but not in γ. Now, the following Σ_2^B formula expresses that there is an x, y-cut of size κ in A:

$$\begin{aligned}\text{CutSize}(A, x, y, \kappa) :=&\exists \gamma \leq (n-2)\text{Cut}(A, \gamma) \wedge \Sigma\gamma = \kappa \wedge (\forall \alpha \leq n-2)\\&\neg\text{Path}(\lambda_{pq}\langle n-2, n-2, A(p, q) - \gamma(p, q)\rangle, x, y, \alpha),\end{aligned} \tag{4.23}$$

and the minimum number of vertices in an x, y-cut can be expressed with a formula that is a conjunction of a Σ_2^B formula with a Π_2^B formula, yielding therefore a formula

in $\Sigma_3^B \cap \Pi_3^B$:

$$\text{MinCut}(A, x, y, \kappa) := \text{CutSize}(A, x, y, \kappa) \wedge \neg\text{CutSize}(A, x, y, \kappa - 1) \tag{4.24}$$

Putting it all together, we can state Menger's theorem in $\mathcal{L}_{\mathbf{LA}}$ with a $\Sigma_3^B \cap \Pi_3^B$ formula as follows:

$$\text{Menger}(A) := \text{MaxDisj}(A, x, y, \lambda) \wedge \text{MinCut}(A, x, y, \kappa) \rightarrow \lambda = \kappa \tag{4.25}$$

(Note that if a formula is in $\Sigma_3^B \cap \Pi_3^B$, then its negation is still in $\Sigma_3^B \cap \Pi_3^B$.)

Lemma 4.3.3 $\mathbf{LA} \cup \text{Menger} \vdash \text{KMM}$.

Proof: Note that the implication resembles the statement of KMM, but the difference is that in KMM the antecedent is a conjunction of two Π_1^B formulas (and hence it is a Π_1^B formula), whereas in Menger's theorem, the antecedent is a $\Sigma_3^B \cap \Pi_3^B$ formula.

Suppose that we have $\text{MinCover}(A, \alpha) \wedge \text{MaxSelect}(A, \beta)$, the antecedent of KMM (see 4.9). Using Menger's theorem (see 4.25) we want to conclude that $\Sigma\alpha = \Sigma\beta$. We do so by restating "covers and selections" of A as "cuts and paths" of a related matrix $A_{x,y}$, see below.

Consider a bipartite graph $G = (V_0 \cup V_1, E)$, where $E \subseteq V_0 \times V_1$. Let A be the adjacency matrix for G where $A(i, j) = 1$ iff $i \in V_0$ and $j \in V_1$ and $(i, j) \in E$. We now extend G to $G_{x,y}$ by adding two new vertices, x and y, and edges $\{(x, v) : v \in V_1\}$, denoted "red edges", and edges $\{(y, v) : y \in V_0\}$, denoted "green edges." See Figure 4.4.

The adjacency matrix $A_{x,y}$ of $G_{x,y}$ is of size $(|A|+1) \times (|A|+1)$ and:

$$A_{x,y}(i,j) = \begin{cases} A(i,j) & \text{for } 1 \leq i,j \leq |A| \\ 1 & \text{one } \{i,j\} \text{ equals } |A|+1 \\ 0 & \text{both } \{i,j\} \text{ equal } |A|+1 \end{cases}$$

Note that $A_{x,y}$ can be stated succinctly as a term of $\mathcal{L}_{\mathbf{LA}}$:

$$A_{x,y} := \lambda ij\langle r(A)+1, c(A)+1, \mathtt{cond}(1 \leq i,j \leq |A|, A(i,j), \mathtt{cond}(i=j=|A|+1, 0, 1))\rangle$$

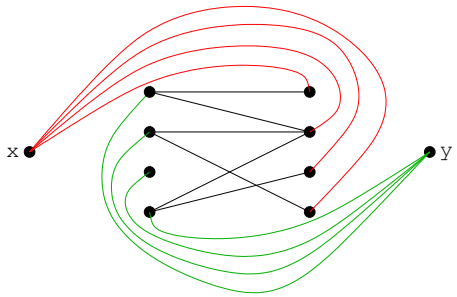


Figure 4.4: Extension of graph G.

A maximal selection in A corresponds to a maximal matching in the related graph, and a minimal cover in A corresponds to a minimal cover in the related graph (recall that a cover in a graph is a subset of vertices so that every edge has at least one end-point in this subset). Furthermore, a maximal matching in the graph related to A corresponds to a maximal subset of internally disjoint paths in the graph related to $A_{x,y}$; similarly, a minimal cover in the graph related to A corresponds to a minimal cut in the graph related to $A_{x,y}$.

As the graphs related to Menger's Theorem are not bipartite, we convert $A_{x,y}$ to a

non-bipartite graph A' as follows:

$$A' = \begin{bmatrix} 0 & A_{x,y} \\ A_{x,y}^T & 0 \end{bmatrix},$$

Let G' be the non-bipartite graph represented by A'. We now finish the proof of the Lemma 4.3.3 with a sequence of claims.

Claim 4.3.5 LA *proves that if there is a cut in G' of size k, then there is a cut in G' if size k that only cuts the red/green edges, i.e., only those edges that are adjacent to either x or y.*

PROOF: Suppose that a black edge is part of a cut. Every x, y-path crosses from V_0 to V_1, and taking off one black edge can only block one x, y-path; the same path is blocked by taking off the corresponding red or green edge. □

Claim 4.3.6 LA *proves the following:*

- $\text{MinCover}(A, \alpha) \iff \text{MinCut}(A', x, y, \Sigma\alpha)$
- $\text{MaxSelect}(A, \beta) \iff \text{MaxDisj}(A', x, y, \Sigma\beta)$

More generally, **LA** *proves the following two:*

1. *G has a matching of size k $\iff$ G' has k disjoint x, y-paths.*
2. *G has a vertex cover of size k $\iff$ G' has an x, y-cut of size k.*

Claim 4.3.6 follows directly from Claim 4.3.5. On the other hand, the direct consequence of Claim 4.3.6 is that the size of a maximum matching in G equals the size of a

maximum set of disjoint x, y-paths in G'; and the size of the minimum vertex cover in G equals the size of the minimum x, y-cut in G'. All this is provable in **LA** as follows: PROOF:

$$\mathbf{LA} \vdash \text{MinCover}(A, \alpha) \iff \text{MinCut}(A', x, y, \Sigma\alpha)$$

First of all, we know that α encodes a number of lines needed to cover all the 1s of A, particularly, this α corresponds to a minimum number of lines to cover all the 1s in A, which in terms of A'—using the construction of Lemma 4.3.3—correspond to a set of vertices whose removal disconnects x from y, that is, an x, y-cut γ, in particular minimum size, say κ, because α is minimum. Hence, $\Sigma\gamma = \kappa = \Sigma\alpha$ concluding that $\text{MinCut}(A', x, y, \kappa)$ is satisfied. On the other hand, we have

$$\mathbf{LA} \vdash \text{MaxSelect}(A, \beta) \iff \text{MaxDisj}(A', x, y, \Sigma\beta)$$

We know that β encodes a selection of 1s no two on the same line on A, in particular this β denotes a maximal selection, which in terms of A', its correspond to a maximal subset of internally disjoint x, y-paths in the related graph of A', say λ, then $\Sigma\beta[\lambda] = \Sigma\beta = \lambda$, where the first β encodes the internally disjoint x, y-paths and the second one encodes the number of 1s no two on the same line in A, this yields the desire equivalence, and finish with the proof of Claim 4.3.6. □

This ends the proof of Lemma 4.3.3 because by Menger's Theorem, the size of the maximum set of disjoint x, y-paths in G' equals the size of the minimum x, y-cut in G'. Therefore, the size of the maximum matching in G equals the size of the minimum vertex cover in G, i.e., $\Sigma\alpha = \Sigma\beta$.

□

Lemma 4.3.4 $\mathbf{LA} \cup \text{KMM} \vdash \text{Menger}$.

PROOF: Suppose that we have $\text{MaxDisj}(A, x, y, \lambda)$ and $\text{MinCut}(A, x, y, \kappa)$; these two formulas assert the existence of β, a collection of λ many pairwise disjoint x, y-paths, and γ, an x, y-cut of size κ. (The constructions of β and γ have been shown earlier in this section.)

Each path in β must have at least one vertex in the cut γ and no vertex of γ can be in more than one path in β, hence $\lambda \leq \kappa$. The proof of this is identical to the proof of Claim 4.3.2.

Thus, it remains to show, using KMM, that $\lambda \geq \kappa$.

The proof of this is inspired by [Aha83]; we assume that G is directed, but a simple construction gives us the undirected case as well. Let $A = \{u \in V : (x, u) \in E\}$ and let $B = \{v \in V : (v, y) \in E\}$. Let $X = V - (A \cup B)$, and also split every vertex $v \in V$ into two vertices v', v''. We now construct a new bipartite graph Γ where the two sides are given by $A' \cup X'$ and $B'' \cup X''$, and where the edges are given by $\{(u', v'') : (u, v) \in E\} \cup \{(x', x'') : x \in X\}$. By KMM there is a matching M and a cover C in Γ of the same size. We let $\mathcal{P}$ be the set of paths $\{x_1, x_2, \ldots, x_k\}$ such that $(x'_i, x''_{i+1}) \in M$, and we let $\mathcal{S}$ be a cut consisting of $\{v \in V : v', v'' \in C \text{ or } v' \in A' \cap C \text{ or } v'' \in B'' \cap C\}$. **LA** can prove that $\mathcal{P}$ is a set of disjoint paths, and $\mathcal{S}$ is a cut, and $|\mathcal{P}| \geq |\mathcal{S}|$. This is enough to prove the lemma as: $\lambda \geq |\mathcal{P}| \geq |\mathcal{S}| \geq \kappa$. □

4.3.5.2 Hall's Theorem

Let $S_1, S_2, \ldots, S_n$ be n subsets of a given set M. Let D be a set of n elements of M, $D = \{a_1, a_2, \ldots, a_n\}$, such that $a_i \in S_i$ for each $i = 1, 2, \ldots, n$. Then[7] D is said to be

[7]Note that we do not add the condition that $a_i \neq a_j$ whenever $i \neq j$ like the classical definition requires, the reason is because we are using set and no multisets. So, implicitly the definition that

a *system of distinct representative* (SDR) for the subsets $S_1, S_2, \ldots, S_n$.

If the subsets $S_1, S_2, \ldots, S_n$ have an SDR, then any k of the sets must contain between them at least k elements. The converse proposition is the combinatorial theorem of P. Hall: suppose that for any $k = 1, 2, \ldots, n$, any $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_k}$ contains at least k elements of M; we call this the *union property*. Then there exists an SDR for these subsets. (See [Hal87, EW49, HV50] for more on Hall's theorem.)

We formalize Hall's theorem in **LA** with an adjacency matrix A such that the rows of A represent the sets S_i, and the columns of A represent the indices of the elements in M, i.e., the columns are labeled with $[n] = \{1, 2, \ldots, n\}$, and $A(i, j) = 1 \iff j \in S_i$. Let $\mathrm{SDR}(A)$ be the following Σ_1^B formula which states that A has a system of distinct representatives:

$$\mathrm{SDR}(A) := (\exists P \leq n)(\forall i \leq n)(AP)_{ii} = 1 \tag{4.26}$$

We reserve the letters P, Q for permutation matrices, and $(\exists P \leq n)\phi$ abbreviates $(\exists P)[\mathrm{Perm}(P) \wedge |P| \leq n \wedge \phi]$ (similarly for $(\forall P \leq n)\phi$, but with an implication instead of a conjunction), where Perm is a Σ_0^B predicate stating that P is a permutation matrix (a unique 1 in each row and column)—see Section 4.3.2 Formula 4.11. See [Sol04] for more details about handling permutation matrices.

The next predicate is a Π_2^B formula stating the union property:

$$\begin{aligned} &\mathrm{UnionProp}(A) := \forall P \leq n \forall k \leq n \exists Q \leq n \\ &[\forall i \leq k(\lambda_{pq}\langle k, 1, (PAQ)_{pi}\rangle \neq \lambda_{pq}\langle k, 1, 0\rangle)] \end{aligned} \tag{4.27}$$

we use is equivalent to the classical one.

Therefore, we can state Hall's theorem as a Σ_2^B formula:

$$\text{Hall}(A) := \text{UnionProp}(A) \rightarrow \text{SDR}(A) \tag{4.28}$$

Lemma 4.3.5 $\mathbf{LA} \cup \text{KMM} \vdash \text{Hall}$.

PROOF: Let A be a 0-1 sets/elements incidence matrix of size $n \times n$. Assume that we have UnionProp(A); our goal is to show in **LA**, using KMM, that SDR(A) holds.

Since by Claim 4.3.1, every matrix can be put in a form that the matrix has the diagonal property, using the fact that we have UnionProp(A), it follows that we can find $P, Q \leq n$ such that $\forall k \leq n (PAQ)_{kk} = 1$. Thus we need n lines to cover all the 1s, but by KMM there exists a selection of n 1s no two on the same line, hence, A is of term rank n.

But this means that the maximal selection of 1s, no two on the same line, constitutes a permutation matrix P (since A is $n \times n$, and we have n 1s, no two on the same line). Note that AP^T has all ones on the diagonal, and this in turn implies SDR(A). □

Lemma 4.3.6 $\mathbf{LA} \cup \text{Hall} \vdash \text{KMM}$.

PROOF: Suppose that we have MinCover(A, α) and MaxSelect(A, β); we want to conclude that $\Sigma\alpha = \Sigma\beta$ using Hall's Theorem.

As usual, let $l_A = \Sigma\alpha$ and $o_A = \Sigma\beta$, and by Claim 4.3.2 we already have that $\mathbf{LA} \vdash o_A \leq l_A$ (see Section 4.3.2). We now show in **LA** that $o_A \geq l_A$ using Hall's Theorem.

Suppose that the minimum number of lines that cover all the 1s of A consists of e rows and f columns, so that $l_A = e + f$. Both l_A and o_A are invariant under permutations of the rows and the columns of A (Lemma 4.3.1), and so we reorder the

rows and columns of A so that these e rows and f columns are the initial rows and columns of A',

$$A' = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix},$$

where A_1 is of size $e \times f$. Now, we shall work with the term rank of A_2 and A_3 in order to show that $o_A \geq l_A$. More precisely, we will show that the maximum number of 1s, no two on the same line, in A_2 is e, while in A_3 it is f.

Let us consider A_2 as an incidence matrix for subsets $S_1, S_2, \ldots, S_e$ of a universe of size $|A| - f$, and A_3^t (which is the transpose of A_3) as an incidence matrix for subsets $S'_1, S'_2, \ldots, S'_f$ of a universe of size $|A| - e$. It is not difficult to prove that UnionProp(A_2) and UnionProp(A_3^t) holds (and can be proven in **LA**; see Claim 4.3.7), which in turn implies SDR(A_2) and SDR(A_3^t), respectively, by Hall's Theorem. But the system of distinct representatives of A_2 (respectively A_3^t) implies that $o_{A_2} \geq e$ (respectively $o_{A_3^t} = o_{A_3} \geq f$), and since $o_A \geq o_{A_2} + o_{A_3}$, this yields that $o_A \geq e + f = l_A$. □

Claim 4.3.7 LA *proves the following:*

- **LA** $\vdash$ UnionProp(A_2)
- **LA** $\vdash$ UnionProp(A_3^t)

Proof: First of all, we can formalize the Union Property for A_2 and the formalization of A_3^t is identical to the former. Hence, using the formula (4.27) on A_2 we have,

$$\text{UnionProp}(A_2) := \forall P \leq n \, \forall k \leq n \, \exists Q \leq n \, [\forall i \leq k (\lambda_{pq}\langle k, 1, (PA_2Q)_{pi}\rangle \neq \lambda_{pq}\langle k, 1, 0\rangle)] \tag{4.29}$$

that is, the first k columns of PA_2Q have a non-zero entry somewhere in the first k rows. Therefore, this show that the corresponding k subsets have at least k elements in their union set, i.e., any $S_1 \cup S_2 \cup \cdots \cup S_e$ contains at least e elements of A, where we use the same e of Lemma 4.3.6, which denotes number of rows. Suppose by contradiction, the we can replace certain of the e rows by fewer rows and preserve the same number of lines that cover all the 1s on A, but this cover is accomplished with fewer than $e + f$ lines contradicting the minimality of l_A, remember that we have MinCover(A, α) and MaxSelect(A, β). Similarly we can prove that UnionProp(A_3^t) but working over f columns instead of e rows, and our claim follows. □

4.3.5.3 Dilworth's Theorem

Let $\mathcal{P}$ be a *finite partially ordered set* or *poset* (we use a "script $\mathcal{P}$" in order to distinguish it from permutation matrices, denoted with P). We say that $a, b \in \mathcal{P}$ are *comparable elements* if either $a < b$ or $b < a$. A subset C of $\mathcal{P}$ is a *chain* if any two distinct elements of C are comparable. A subset S of $\mathcal{P}$ is an *anti-chain* (also called an *independent set*) if no two elements of S are comparable.

We want to partition a poset into chains; a poset with an anti-chain of size k cannot be partitioned into fewer than k chains, because any two elements of the anti-chain must be in a different partition. Dilworth's Theorem states that the maximum size of an anti-chain equals the minimum number of chains needed to partition $\mathcal{P}$. (For more on Dilworth's Theorem see [Dil50, Per63]).

In order to formalize Dilworth's theorem in **LA**, we represent finite posets $\mathcal{P} = (X = \{x_1, x_2, \ldots, x_n\}, <)$ with an incidence matrix $A = A_{\mathcal{P}}$ of size $|X| \times |X|$, which expresses the relation $<$ as follows: $A(i, j) = 1 \iff x_i < x_j$. For more material

regarding formalizing posets see [Sol11].

We let a $1 \times n$ matrix α encode a chain as follows:

$$\text{Chain}(A, \alpha) := (\forall i \neq j \leq n)[\alpha(i) = \alpha(j) = 1 \to A(i, j) = 1 \vee A(j, i) = 1]. \quad (4.30)$$

In a similar fashion to (4.30) we define an anti-chain γ; the only difference is that the succedent of the implication expresses the opposite: $A(i, j) = 0 \wedge A(j, i) = 0$.

Recall that using (4.19) we were able to talk about a collection of paths; in a similar vain, we can use **LA** to talk about a collection of chains of $\mathcal{P}$: β is an $1 \times \kappa \cdot n$ matrix which encodes the contents of κ many chains. We can then talk about a minimal collection of chains, or a maximal size of an anti-chain in the usual fashion. Both concepts are formalized like follows, first of all, we must be able to talk about a collection of disjoint chains; the 0-1 matrix β will encode a collection of disjoint chains $\alpha_1, \alpha_2, \ldots, \alpha_\lambda$:

$$\beta = \boxed{\beta[1] = \alpha_1 \mid \beta[2] = \alpha_2 \mid \ldots \mid \beta[\kappa] = \alpha_\kappa} \quad (4.31)$$

so that β is a matrix of size $1 \times \kappa \cdot n$. Each $\beta[i]$ can be defined thus:

$$\beta[i] := \lambda_{pq}\langle 1, n, \beta(p, (i-1)n + q)\rangle.$$

We are interested in pairwise disjoint collections of chains:

$$\begin{aligned} &\text{CollectDisjChains}(A, \beta) := \\ &\forall i \leq n \;\; \exists j \leq n \; (\beta[j](1, i) = 1) \qquad (*) \\ &\wedge (\forall i \leq n \forall j, k \leq n) \;\; (\beta[j](1, i) = 1 \to ((j \neq k) \wedge \beta[k](1, i) = 0)) \quad (**) \end{aligned} \quad (4.32)$$

where $(*)$ denotes that every element in $\mathcal{P}$ is in one of the κ chains, and $(**)$ express the disjointness condition, also note that we use the notation $\beta[k](i,j)$ to express the entry (i,j) of the k-th chain.

The predicate $\text{MinChain}(A,\beta,\kappa)$ asserts that β is a collection of κ many chains that partition the $\mathcal{P}$ for a given A; note that it is a Π_1^B formula:

$$\begin{aligned}&\text{MinChain}(A,\beta,\kappa) := \forall\beta' \leq n^2\ [\text{CollectDisjChains}(A,\beta')\wedge \\ &(\text{CollectDisjChains}(A,\beta) \rightarrow \texttt{div}(c(\beta'),n) \geq \texttt{div}(c(\beta),n)] \wedge \kappa = \texttt{div}(c(\beta),n)\end{aligned} \tag{4.33}$$

Beside note that we are using n^2 (where $n \times n = |A|$) instead of $n \cdot \kappa$ to bound our matrices by the worst case—in sense that we are looking for a minimum chain no maximum—which is, each singleton is a chain, and we use $\texttt{div}(c(\beta),n)$—that is, the quotient 2-ary function that belong to **LA**—to denote the number of chains in β.

We already define a 0-1 matrix γ encoding anti-chains, then we need to formalize the predicate $\text{MaxAntiChain}(A,\gamma,\lambda)$ which it asserts that γ is an anti-chain consisting of λ elements:

$$\begin{aligned}&\text{MaxAntiChain}(A,\gamma,\lambda) := \\ &\text{AntiChain}(A,\gamma) \wedge |\gamma| = \lambda\ \wedge \forall\gamma' \leq n^2(\text{AntiChain}(A,\gamma') \rightarrow \Sigma\gamma' \leq \lambda)\end{aligned} \tag{4.34}$$

Finally, we can state Dilworth's Theorem as following Σ_1^B formula:

$$\begin{aligned}\text{Dilworth}(A) := (\exists\beta \leq |A|^2)(\exists\gamma \leq |A|) \\ \text{MinChain}(A,\beta,\kappa) \wedge \text{MaxAntiChain}(A,\gamma,\lambda) \rightarrow \lambda = \kappa\end{aligned} \tag{4.35}$$

Lemma 4.3.7 $\mathbf{LA} \cup \text{KMM} \vdash \text{Dilworth}$

PROOF: Suppose that we have $\text{MinChain}(A, \beta, \kappa)$ and $\text{MaxAntiChain}(A, \gamma, \lambda)$; we want to use **LA** reasoning and KMM in order to show that $\lambda = \kappa$.

As usual we define a matrix A' whose rows are labeled by the chains in β, and whose columns are labeled by the elements of the poset. As there cannot be more chains than elements in the poset, it follows that the number of rows of A' is bounded by $|A|$ (while the number of columns is exactly $|A|$). The proof of this (Lemma 4.3.8) is similar to the proof of Claims 4.3.2, and **??**. That is,

Lemma 4.3.8 $\mathbf{LA} \vdash \lambda \leq \kappa$

PROOF: We formalize this argument in **LA** as follows: let A' be a matrix described above. Let $A'(i,j) = 1 \iff$ chain i contains element j. Then,

$$\begin{aligned} \lambda = c(A') &\leq \Sigma A' && (*) \\ &= \Sigma_i(\Sigma\lambda pq\langle 1, c(A'), A'(i,q)\rangle) && (**) \\ &\leq \Sigma_i 1 = r(A') = \kappa, \end{aligned}$$

where the inequality in the line labeled by $(*)$ can be shown by induction on the number of columns of a matrix which has the condition that each column contains at least one 1, that is, each element appear in at least one chain, as β is a partition of $\mathcal{P}$, e.g., the trivial one which is a singleton; and the equality labeled with $(**)$ follows from the fact that we can add all the entries in a matrix by columns. Besides A' is such that each row contains at most one 1 because each element can be at most in one chain (of course there are chains that are not disjoints and that is the reason why $\Sigma_i 1 \geq \Sigma_i(\Sigma\lambda pq\langle 1, c(A'), A'(i,q)\rangle)$), but like we are working over mutually disjoint

chains, if we pick up one 1 then we can not choose that element (j-position of A') again to be part of another chain (row). □

On the other hand, rows may contain more than one 1, as in general chains may have more than one element.

Note that a maximal selection of 1s, no two of them on the same line, corresponds naturally to a maximal anti-chain; such a selection picks one 1 from each line, and so its size is the number of rows of A'. By KMM, it follows that

$$\lambda = o_{A'} = l_{A'} = r(A') = \kappa,$$

where $r(A')$ is the number of rows of A'. □

Lemma 4.3.9 $\mathbf{LA} \cup \text{Dilworth} \vdash \text{KMM}$

PROOF: It is in fact easier to show that that $\mathbf{LA} \cup \text{Dilworth} \vdash \text{Hall}$, and since by Lemma 4.3.6 we have that $\mathbf{LA} \cup \text{Hall} \vdash \text{KMM}$, we will be done.

In order to prove Hall using Dilworth and **LA** reasoning, we assume that we have A, a 0-1 sets/elements incidence matrix of size $n \times n$. Assume that we have UnionProp(A); our goal is to show in **LA**, using Dilworth, that SDR(A) holds.

Let $S_1, S_2, \ldots, S_n$ be subsets of $\{x_1, x_2, \ldots, x_n\}$ where $n = |A|$. We define a partial order $\mathcal{P}$ based on A; the universe of $\mathcal{P}$ is $X = \{S_1, S_2, \ldots, S_n\} \cup \{x_1, \ldots, x_n\}$. The relation $<_{\mathcal{P}}$ is defined as follows: $x_i <_{\mathcal{P}} S_j \iff A(i,j) = 1$.

Claim 4.3.8 *The maximum size of an anti-chain in $\mathcal{P}$ is n.*

PROOF: The $\{x_1, \ldots, x_n\}$ form an anti-chain of length n, and we cannot add any of the S_j, as some $x_i \in S_j$, and hence $x_i <_{\mathcal{P}} S_j$. □

By Dilworth we can partition $\mathcal{P}$ into n chains, where each of the chains has two elements $\{x_i, S_j\}$, giving us the set of distinct representatives, and hence SDR(A). $\square$

Chapter 5

Concluding Remarks

5.1 Summary of Contributions

We give the first feasible proof of a fundamental theorem in Combinatorial Matrix Theory (CMT), called König's Mini-Max Theorem (KMM). Our results show that Min-Max reasoning can be formalized with uniform Extended Frege.

Also, we show, by introducing new proof techniques, that **LA**-Theory with Σ_1^B induction is sufficient to formalize a large portion of CMT. Σ_1^B-**LA** corresponds to polynomial time reasoning, also known as $\exists$**LA**. While we consider matrices over $\{0,1\}$, the underlying ring is $\mathbb{Z}$, since we require that ΣA compute the number of 1s in the matrix A (which for a 0-1 matrix is simply the sum of all entries—meaning ΣA). Thus, over $\mathbb{Z}$, **LA** translates to $\mathbf{TC}^0$-Frege, while, as mentioned before, $\exists$**LA** translates into Extended Frege, [SC04, §6.5].

In order to prove KMM in $\exists$**LA**, we need to restrict induction to Σ_1^B formulas. The main technical contribution is presented in Claim 4.3.4 Section 4.3.3. Basically, we introduce a polynomial time procedure, whose proof of correctness can be shown

with $\exists\mathbf{LA}$, that works as follow: given a matrix of size $e \times f$ such that $e \leq f$, where the minimum cover is of size e, our procedure computes a maximum selection of size e, row by row.

Furthermore, we show that Menger's Theorem, Hall's Theorem, and Dilworth's Theorem—theorems related to KMM—can also be proven feasibly; in fact, all these theorems are equivalent to KMM, and the equivalence can be shown in **LA**. We believe that this captures the proof complexity of Min-Max reasoning rather completely.

We also present a new Permutation-Based algorithm for computing a Minimum Vertex Cover from a Maximum Matching in a bipartite graph. Our algorithm uses properties of KMM Theorem and it is interesting for providing a new permutation—and CMT—perspective on a well-known problem.

5.2 Future Works

In this section we present the future works of this thesis in order of major importance.

5.2.1 Can LA prove KMM?

Can we show in **LA**-Theory KMM? Or, if **LA**-Theory is to weak to show KMM, can we prove that? In other words, can we construct a model $\mathcal{M}$ of **LA** such that

$$\mathcal{M} \models \mathbf{LA} \text{ , but } \mathcal{M} \not\models \text{KMM}$$

.

5.2.2 Can we prove KMM in $\mathbf{NC}^2$-Frege?

$\mathbf{NC}^2$ seems to correspond in a natural way to reasoning about matrices: Can we prove KMM in $\mathbf{NC}^2$-Frege? It seems to require to give a good parallel algorithm for translating from the minimum number of lines to cover all the 1s to the maximum number of 1s no two on the same line expressed in KMM, so, our algorithm—Chapter 3—that we have, as far is a polynomial time algorithm, but if we could give an $\mathbf{NC}^2$ algorithm for that, allow us to give an $\mathbf{NC}^2$-Frege proof for KMM.

That is why it will be interesting, because it will require new ideas and a breakthrough, in the sense that the $\mathbf{NC}^2$ algorithm for translating minimum number of lines to cover all the 1s into maximum number of 1s no two on the same line is not obvious. Our intuition is that it should exists such $\mathbf{NC}^2$ algorithm, because so many thing about matrices can be computed in $\mathbf{NC}^2$.

One of the most interesting aspect of $\mathbf{NC}^2$ seems to be that we can formalize in the complexity class $\mathbf{NC}^2$ Berkowitz's algorithm ([Ber84]) to compute characteristic polynomial of the matrices, but the problem seems to be "prove the correctness of Berkowitz's algorithm in $\mathbf{NC}^2$" which still is open.

5.2.3 Are KMM and PHP equivalents in LA?

We know that $\exists\mathbf{LA} \vdash$ KMM (Theorem 4.3.1) and that KMM is equivalent to a host of other combinatorial theorems—and this equivalence can be shown in the weak theory **LA** (Theorem 4.3.2)— it would be interesting to know what is the relationship between the PHP and KMM, that is, whether it is also the case that:

- $\mathbf{LA} \cup \text{KMM} \vdash \text{PHP}$?

- $\mathbf{LA} \cup \text{PHP} \vdash \text{KMM}$?

that is, whether **LA** can prove the equivalence of KMM and the pigeonhole principle. We conjecture that the first assertion is true, and that it should not be too difficult to show it. The second assertion is difficult to say. Note that in the proof of Claim 4.3.2 in Chapter 4 we implicitly show a certain weaker kind of the PHP in **LA**: we showed that if we have a set of n items $\{i_1, i_2, \ldots, i_n\}$ and a second set of m items $\{j_1, j_2, \ldots, j_m\}$, and we can match each i_p with some j_q, and this matching is both definable in **LA** and its injectivity is provable in **LA**, then $n \leq m$. We did this by defining an incidence matrix A such that $A(p,q) = 1 \iff i_p \mapsto j_q$. If this mapping is injective, then each column of A has at most one 1; thus:

$$n \leq \Sigma A = \Sigma_i(\text{col } i \text{ of } A) \leq \Sigma_i 1 \leq m.$$

It follows more or less directly that our **LA** results can also be formalized in the theory $\mathbf{VTC}^0$ (and vice versa), defined in [CN10, pg. 283]. The reason is that the function ΣA is exactly Buss' function Numones(A) ([Bus86] and [Bus90, pg. 6]), i.e., the function that counts the number of 1s in A, and $\mathbf{TC}^0$ is the $\mathbf{AC}^0$ closure of Numones, [CN10, Proposition IX.3.1]. On the other hand, our $\exists\mathbf{LA}$ results can also be formalized in $\mathbf{V}^1$, defined in [CN10, pg. 133]. This is to be expected as we already mentioned that **LA** over $\mathbb{Z}$ corresponds to $\mathbf{VTC}^0$, which proves PHP.

Bondy's Theorem states that for any $n \times n$ 0-1 matrix whose rows are distinct, we can always delete a column so that the remaining $n \times (n-1)$ matrix still has n distinct rows. [CN10, §IX.3.8] investigate the connection between Bondy's Theorem (**BONDY**) and PHP, and they show that $\mathbf{V}^0 \vdash \mathbf{BONDY} \leftrightarrow \text{PHP}$. It would be interesting to know if $\mathbf{V}^0 \vdash \text{KMM} \leftrightarrow \text{PHP}$. We know that **LA** over $\mathbb{Z}$ proves PHP,

so the question is: $\mathbf{V}^0 \vdash \text{KMM} \leftrightarrow \text{PHP}$? Or equivalently, does **LA** without Σ prove KMM $\leftrightarrow$ PHP?

5.2.4 Can LA $\cup$ KMM prove Hard Matrix Identities?

We would like to know whether **LA** $\cup$ KMM can prove hard matrix identities, such as $AB = I \rightarrow BA = I$. (See more about Hard Matrix Identities [Sol01], and [SU04]). Of course, we already know from [HT11] that (non-uniform) $\mathbf{NC}^2$-Frege is sufficient to prove $AB = I \rightarrow BA = I$, and from [SC04] we know that $\exists$**LA** can prove them also.

On the other hand, is it possible that **LA** together with $AB = I \rightarrow BA = I$ can prove KMM? This would imply that $AB = I \rightarrow BA = I$ is "complete" for combinatorial matrix algebra, in the sense that all of combinatorial matrix algebra follows from this principle with proofs of low complexity.

5.2.5 A question about l_{AB} and o_{AB} over 0-1 matrices

Given two 0-1 matrices A, B, what can we say about l_{AB} and o_{AB}, where l_{AB} and o_{AB} denote the minimum number of lines necessary to cover all the 1s of AB, and the maximum number of 1s no two on the same line, respectively? From Lemma 4.3.1 we know that if B is a permutation matrix, then $l_{AB} = l_A$ and $o_{AB} = o_A$ (and similarly, if A is a permutation matrix); but what can be said in general? Of course, the understanding here is that multiplication is over $GF(2)$.

5.2.6 Can we extend our results to General Graphs?

Having in mind our approach of the Proof Complexity of Combinatorial Matrix Theory, will be interesting to apply concepts of **LA**-Theory—and possibly extend

it—in order to formalize the general non-bipartite graphs considering the seminal work of Edmonds [Edm65], in which J. Edmonds present his well-known Blossom Shrinking algorithm to solve Maximum Matching problem in General Graphs in polynomial time.

5.2.7 Can we extend our result to Infinite Bigraphs?

In the same flavor of General Graphs—previous question—will be interesting give a treatment of infinite bipartite graphs in which KMM holds, from a purely Combinatorial Matrix Theory and Proof Complexity mix approach. Was Ron Aharoni [Aha83] how proved Menger's Theorem for infinite graphs, and was Erdös how first conjecture KMM for infinite graphs, later proved by Aharoni [Aha84], so, in this vein will be interesting to investigate what logical theory can formalize those proofs? And investigate if can we show that **LA** is too weak to prove infinite KMM?

5.2.8 More questions about General Graphs

Let $\mathcal{L}$-*MinMax* be the following language:

$$\mathcal{L}\text{-}MinMax = \{\langle G\rangle : |\text{MaxSelect}(A_G, \alpha)| = |\text{MinCover}(A_G, \beta)|\}$$

where G is a graph and A_G is its adjacency matrix, and α is a matrix that it keeps track of the lines that cover A; it does so with two rows: the top row keeps track of the horizontal lines, and the bottom row keeps track of the vertical line. The condition ensures that any 1 in A is covered by some line stipulated in α, finally, β is a matrix which is a selection of 1s of A so that no two of these ones are on the same line. Thus, β can be seen as a "subset" of a permutation matrix. (For more details see Section 4.3.2. So, will be interesting to know:

1- What is the weakest complexity class that contains $\mathcal{L}$-*MinMax*?

2- What is the largest class of graphs for which $\mathcal{L}$-*MinMax* holds?

5.2.9 Can LA prove Decomposition Theorems?

The definitions and concepts of Decomposition Theorem that we present in this section come from [BR91, Chapter 4 Section 4.4].

Let A be an $m \times n$ matrix, and let $\mathcal{P}$ denote a class of matrices. By a *decomposition theorem* we mean a theorem which asserts that there is an expression for A of the form

$$A = P_1 + P_2 + \cdots + P_k + X \qquad (*)$$

where the matrices $P_1, P_2, \ldots, P_k \in \mathcal{P}$. We may require X to be restricted in some way, perhaps equal to a zero matrix. The purpose of the theorem may be to maximize k in $(*)$ or to minimize k in the event that X is required to be zero matrix, or the purpose may be to maximize or minimize some other quantity that can be associated with the decomposition $(*)$.

The KMM can be viewed as a decomposition theorem. Recall that an $m \times n$ 0-1 matrix P is a *subpermutation matrix of rank r* provided P exactly r 1s and no two 1s of P are on the same line. Let A be an $m \times n$ 0-1 matrix. Then KMM asserts that A can be expressed in the form

$$A = P + X$$

where P is a subpermutation matrix of rank r and X is a 0-1 matrix if and only if A does not have a line cover consisting of fewer than r lines.

Hence, considering our Proof Complexity of Combinatorial Matrix Theory approach,

will be interesting to apply concepts of **LA**-Theory in order to formalize Decomposition Theorems. (For more about Decomposition Theorems see [BR91, Gup67]).

Appendix A

Appendix

A.1 A Π_2^B-Inductive proof of König's Min-Max Theorem in LA

In this appendix we show the classical proof in **LA**-Theory with Π_2^B induction—therefore not feasible—of König Min-Max Theorem (KMM), a fundamental result in Combinatorial Matrix Theory (CMT). In order to make this appendix self-contained we review some definitions and notation. Basically, in this appendix we are formalizing in $\mathcal{L}_{\mathbf{LA}}$ the classical Π_2^B inductive proof.

We are interested in the proof complexity of König Min-Max Theorem over the ring of $\mathbb{Z}$. Unless otherwise specified, we assume that all our matrices are over $GF(2)$, that is, the Galois Field of two elements $\{0, 1\}$. We use the following notation—for more details about the **LA** notation see Section 4.1.2 and [SC04, Section 2].

We use A_{ij} to denote entry (i, j) of the matrix A, which in $\mathcal{L}_{\mathbf{LA}}$ correspond to the 3-ary function symbol $e(A, i, j)$ where the first argument is a type element matrix,

and the other two are of type element index; $r(A)$, or sometimes $row(A)$, and $c(A)$, or sometimes $col(A)$, to denote number of rows of 0-1 matrix A and number of columns of 0-1 matrix A respectively.

Sometimes, we shall apply function $f : M_{m\times n}(GF(2)) \mapsto \mathbb{Z}$, that is, function from 0-1 matrices of size $m \times n$ to integers. We require the integers as one of our fundamental operations will be counting the number of 1s in a 0-1 matrix, i.e., compute $f := \Sigma A$, the sum of all the entries of A. In fact, we will just do Boolean Matrices since we are interested in combinatorial properties more than arithmetical ones.

Also $\texttt{cond}(\alpha, t_1, t_2)$ is interpreted **if** α **then** t_1 **else** t_2, where α is a formula all of whose atomic subformulas have the form $m \leq n$ or $m = n$, where m, n are terms of type index, and t_1, t_2 are terms either both of type index or both of type field.

We are going to use the decomposition of an $m \times n$ matrix A will be used in our proof:

$$\mathbf{A} = \begin{bmatrix} a_{11} & R \\ S & M \end{bmatrix}$$

where a_{11} is the $(1, 1)$ entry of A, and R, S are $1 \times (n-1), (m-1) \times 1$ submatrices, respectively, and M is the principal submatrix of A. Therefore, we make the following precise definitions:

$$\begin{aligned} R(A) &:= \lambda ij\langle 1, c(A) - 1, e(A, 1, i + 1)\rangle \\ S(A) &:= \lambda ij\langle r(A) - 1, 1, e(A, i + 1, 1)\rangle \\ M(A) &:= \lambda ij\langle r(A) - 1, c(A) - 1, e(A, i + 1, j + 1)\rangle \end{aligned}$$

Also we are going to use the folowing $\mathcal{L}_{\mathbf{LA}}$ rule:

Induction rule

$$\frac{\Gamma, K(i) \to K(i+1), \Delta}{\Gamma, K(0) \to K(n), \Delta}$$

Here the variable i of type index may not occur free in either Γ or Δ. Also $K(i)$ is any formula, n is any term of type index, and $K(n)$ indicates n is substituted for free occurrences of i in $K(i)$. Similarly for $K(0)$. Nevertheless, we are going to do induction over Π_2^B formulas, i.e., $K(i) \in \Pi_2^B$.

Before start with details about the KMM, we are going to explain the inductive approach used to prove it. So, we will prove KMM—which concern to 0-1 matrices from the Combinatorial Matrix Theory viewpoint, by induction on matrices size.

Considering that **LA**-theory is indeed a theory of matrix algebra we outline a strategy for proving claims about matrices by induction on their size. We follow the presentation and the notation of the next outline from [Sol01, Chapter 3]. First of all, note that it is possible to define empty matrices[1] (matrices with zero rows or zero columns) but such a matrices are consider to be special. Our theorems holds for this special case, by axioms A28 and E (see [Sol01]), so we implicitly assume that it holds. The Basis Case in the inductive proofs is when there is one row or one column. Therefore, instead of doing induction on i (see **LA** Induction Rule), we do induction on j, where $i = j + 1$.

As matrix has two parameters, we deal with this as follow, suppose that we want to prove something for all matrix A. We define a new constructed matrix $M(i, A)$ as

[1]From axioms for matrices, we have that the axiom **E**(mpty) is necessary to take care of empty matrices. There is nothing that prevents us from construction a matrix, for instance, $\lambda ij\langle 0, 3, t\rangle$, and we want Σ of such matrix to be 0_{field}, regardless of t.

follows: let $d(A) := \mathtt{min}\{r(A), c(A)\}$. Now let:

$$M(i, A) := \lambda pq\langle r(A) - d(A) + i, c(A) - d(A) + i, e(A, d(A) - i + p, d(A) - i + q)\rangle$$

i.e., $M(i, A)$ is the i-th principal submatrix of A.

To prove that the property $\mathcal{P}$ holds for A, we prove that $\mathcal{P}$ holds for $M(1, A)$, i.e., the Basis Case, and we prove that if $\mathcal{P}$ holds for $M(i, A)$, it also holds for $M(i+1, A)$, i.e., Induction Step. From this we conclude, by induction rule, that $\mathcal{P}$ holds for $M(d(A), A)$, but this is exactly A. Note that in the Basis Case we might have to prove that $\mathcal{P}$ holds for a $(1 \times k)$-row vector or a $(k \times 1)$-column vector, and this in turn can also be done by induction on k.

Basically, we use induction to prove claims for bigger and bigger submatrices. We can define and parameterize these submatrices using constructed terms ($\lambda ij\langle m, n, t\rangle$).

Also, we use $|A| \leq n$ to abbreviate $r(A) \leq n \wedge c(A) \leq n$, that is, the number of rows of A is bounded by n, and the number of columns of A is bounded by n. We let $(\exists A \leq n)\alpha$ — resp. $(\forall A \leq n)\alpha$ — abbreviate $(\exists A)[|A| \leq n \wedge \alpha]$ — resp. $(\forall A)[|A| \leq n \rightarrow \alpha]$. These are *Bounded Matrix Quantifiers*.

Note that **LA** allows for reasoning with arbitrary quantification; however, in **LA** we only allow induction over formulas without matrix quantification. On the other hand, in $\exists$**LA** we allow induction over so called Σ_1^B formulas. These are formulas, which when presented in prenex form, contain a single block of bounded existential matrix quantifiers. The set of formulas Π_1^B is defined similarly, except the block of quantifiers is universal.

In general, Σ_i^B is the set of formulas which, when presented in prenex form, start with a block of bounded existential matrix quantifiers, followed by a block of bounded

universal matrix quantifiers, with i such alternating blocks. The set Π_i^B is the same, except it starts with a block of universal matrix quantifiers. Remember that the theory $\forall$**LA** is the conservative extension of **LA** that also includes induction axioms for $\forall$**LA** formulas, and $\exists$**LA** conservative extension is similar; sometimes the quantifier is going to be omitted when the context is clear. Finally, we use ":=" to define new objects.

We start giving a first high level approach of the Π_2^B-proof. So, let[2] `KMM` be the formalization of König Min-Max Theorem in $\forall$**LA**, i.e., given by the following formula:

$$\texttt{KMM}(A,n) := |A| = n \wedge \underbrace{\text{MinCover}(A,\alpha) \wedge \text{MaxSelect}(A,\beta)}_{(8)} \rightarrow \underbrace{\sum \alpha = \sum \beta}_{(9)} \quad \text{(A.1)}$$

where the first terms is the abbreviation of $r(A) = n \wedge c(A) = n$; there are two types of formulas on A.1, $\prod_1^B$ and $\mathcal{L}_{\mathbf{LA}}$.

The $\prod_1^B$ formulas are:

$$\text{MinCover}(A,\alpha) := \underbrace{\text{Cover}(A,\alpha)}_{(4)} \wedge \forall \alpha' \leq r(\alpha) \underbrace{\left(\text{Cover}(A,\alpha') \supset \sum \alpha' \geq \sum \alpha\right)}_{(5)}$$

$$\text{MaxSelect}(A,\beta) := \underbrace{\text{Select}(A,\beta)}_{(6)} \wedge \forall \beta' \leq r(\beta) \underbrace{\left(\mathit{select}\left(A,\beta'\right) \supset \sum \beta' \leq \sum \beta\right)}_{(7)}$$

where the $\mathcal{L}_{\mathbf{LA}}$ formulas are:

$$\text{Cover}(A,\alpha) := \forall i,j \leq r(A) \underbrace{(A_{ij} = 1 \supset \alpha_{1,i} = 1 \vee \alpha_{2,j} = 1)}_{(3)}$$

$$\text{Select}(A,\beta) := \forall i,j \leq r(A) \left(\underbrace{(\beta_{ij} = 1 \supset A_{ij} = 1)}_{(1)} \wedge \forall k \leq r(A) \underbrace{(\beta_{ij} = 1 \supset \beta_{ik} = 0 \wedge \beta_{kj} = 0)}_{(2)} \right)$$

Where α and β denote two 0-1 matrices of sizes $2 \times n$ and $m \times n$ respectively, such that α denote the number of lines that cover the 1s on matrix A and β denote

[2]Observe that, we use KMM to abbreviate König's Min-Max Theorem, and `KMM` to denote the **LA**-formalization.

the number of 1s on matrix A no two on the same line. Through this appendix we use $\alpha(i,j)$, and $\alpha_{i,j}$ indistinguishable, same for β, and A.

Let us start with the intended meaning of formula (A.1) and its subformulas in order to do clarify the roll of theses terms on the Π_2^B-proof, so:

In Select, (1) means that every β entry nonzero correspond to the same (i,j) nonzero entry of A, and (2) means that the rest of the ith and jth elements are defined to be zero. Together give as an assertion of the correctness of β.

In Cover, (3) denote an assertion of correctness of α, that is, if the (i,j) entry of A has a 1 then must be exists either an horizontal line over row i of A, denoted by 1 on the first row of α, or a vertical line over the j column of A, denoted by 1 on the second row of α.

In MinCover, (4) and (5) means the selection of the minimum number of lines that cover all the 1s in A. On the other hand, in MaxSelect, (6) and (7) mean the selection of the maximum number of 1s no two on the same line in matrix A.

Finally, given a matrix A of size $m \times n$ with entries over $GF(2)$, in formula A.1, if (8) is true, i.e., if matrix α correspond to the minimum number of lines that cover all 1s in A, and matrix β correspond to the maximum number of 1s no two on the same line in A, then the sum over all entries of α and β are equal, that is, (9) in formula A.1.

The main question of this appendix is: what is the weakest fragment of **LA**-Theory that prove KMM? More specific, what is the weakest induction the we can achieve? Of course, an improve and complete answer to this question was gave in Chapter 4 of this thesis, so, the main intention of this appendix is to show the naive Π_2^B-Inductive proof of KMM.

So, we start rewriting in the $\mathcal{L}_{\mathbf{LA}}$ the classical version of KMM (Theorem 4.2.1 Chapter 4 Section 4.2), which succinctly is expressed by:

Theorem A.1.1 $\prod_2^B \mathbf{LA} \vdash \mathtt{KMM}$

PROOF: (Sketch). The idea of the Π_2^B-proof is the follow: we show by induction on the size of the matrices, that

$$\forall n \ \mathtt{KMM}(A, n) \tag{A.2}$$

from which we can deduce

$$\mathtt{KMM}(A, |A|) \tag{A.3}$$

so,

$$|A| = |A| \ \wedge \ \mathrm{MinCover}(A, \alpha) \wedge \ \mathrm{MaxSelect}(A, \beta) \longrightarrow \sum \alpha = \sum \beta$$

and[3] since $\longrightarrow |A| = |A|$ is an Axiom, by a Cut-Rule[4] we get König Min-Max Theorem, i.e.,

$$\mathrm{MinCover}(A, \alpha) \wedge \mathrm{MaxSelect}(A, \beta) \longrightarrow \sum \alpha = \sum \beta \tag{A.4}$$

And this complete the Π_2^B-proof of KMM. □

Now we proceed to prove in detail each part of the sketched proof mention before, so, we proceed to prove that $\forall A \leq n \ \ \mathtt{KMM}(A, n)$ by induction on n.

But first, we start showing that:

Claim A.1.1 $\forall A \leq n \ \ \mathtt{KMM}(A, n) \in \Pi_2^B$

[3]Recall that $\sum \alpha$ and $\sum \beta$ symbols denote the sum of all entries of matrix α and matrix β respectively, returning a term of type field, in our case $\mathbb{Z}$.

[4]Remember that the underline logic of **LA**-Theory is Gentzen's Calculus.

PROOF: Note that the equation A.4 is implicitly Matrix Universally uantified, consequently we have

$$\forall\alpha \leq n \;\; \forall\beta \leq n \; \text{MinCover}(A,\alpha) \wedge \text{MaxSelect}(A,\beta) \longrightarrow \sum \alpha = \sum \beta \qquad \text{(A.5)}$$

But note[5] that $(\forall X \phi \supset \zeta) \equiv \exists X \, (\neg\phi \vee \zeta)$, therefore we have that the formula A.5 is logically equivalent to,

$$\exists\alpha \leq n \;\; \exists\beta \leq n \;\; \Big(\neg \left(\text{MinCover}(A,\alpha) \wedge \text{MaxSelect}(A,\beta)\right) \vee \sum \alpha = \sum \beta\Big) \qquad \text{(A.6)}$$

That is, formula A.6 is a Σ_1^B-formula. Actually, this is the formula A.1, i.e., `KMM`. So, $\texttt{KMM}(A,n)$ is Σ_1^B-formula. Hence, $\forall A \leq n \;\; \texttt{KMM}(A,n)$ in prenex form[6] has a block of universal quantifiers ($\forall$) followed by a block of existential ones ($\exists$). Therefore, our claim follows. □

When we work with a *general* matrix variable, we use α, but when we work with *particular* matrix we use α_0. Also, we use the $\lambda - \mathcal{L}_{\mathbf{LA}}$ constructors to define

$$[0] := \lambda_{ij}\langle 1,1,0\rangle$$

where the first two values are type index element and the third one is type field element, similarly we define

$$\alpha_0 = \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix} := \lambda_{ij}\langle 1,2,0\rangle$$

[5]We let $\delta \supset \gamma$ abbreviate $\neg\delta \vee \gamma$, and $\delta \equiv \gamma$ abbreviate $\delta \supset \gamma \wedge \gamma \supset \delta$.

[6]A formula A is in *prenex form* is A has the form $Q_1 x_1 \ldots Q_n x_n B$, where Q is either $\forall$ or $\exists$, and B is a quantifier-free formula

PROOF: So, after proving that $\forall A \leq n \;\; \texttt{KMM}(A,n) \in \Pi_2^B$, we proceed with the next step which is to prove by induction on n the equation A.2, that is,

$$\overbrace{\forall A \leq n \;\; \text{MinCover}(A,\alpha) \wedge \text{MaxSelect}(A,\beta) \longrightarrow \sum \alpha = \sum \beta}^{\texttt{KMM}(A,n)}$$

Basis Case: Let $n = 1$ there are two possibilities $A = [0]$ or $A = [1]$.

- **Case 1:** $A = [0]$, so,

$$\alpha_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

is a Cover, that is Cover $\left([0], \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right)$ is true, and since $\sum \alpha_0 = 0$, it is in fact a minimum cover, so,

Claim A.1.2

$$\mathbf{LA} \vdash \text{MinCover} \left([0], \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right).$$

PROOF: We are going to use indistinguishably $[0]$ and $0_{m \times n}$ with $n = 1$ and $m = 1$, and also we use in same way $0_{\mathbb{Z}}$ and 0. On the other hand, in the case that we use an array of more than one row or one column, we denote that explicitly, for instance, an array of zeros of size n columns and 1 row, is denoted by $0_{1 \times n}$.

The proof of Claim A.1.2 is derived by a few claims A.1.4, A.1.5, and A.1.6. But first, we show that $\mathbf{LA} \vdash \text{Cover}\,(A, \alpha)$, where α predicate was defined in A.1.

Claim A.1.3 $\mathbf{LA} \vdash \forall \alpha \leq r\,(0_{m\times n}) \quad \text{Cover}\,(0_{m\times n}, \alpha)\,.$

Proof: First, note that by construction of the Cover formula we know that for ever (i,j)-entry of matrix A, such that, if that entry is 1, we have a 1 in one of the two rows of matrix α, such that, it can be in the ith column or in the jth column depending of the orientation line, horizontal or vertical, respectively. But, in our case, the form of A is exactly the full zero $m \times n$ matrix, therefore the claim is vacuously true, because no matter what α is, that always $A_{ij} = 1$ is going to be false in Cover formula. □

By consequence of Claim A.1.3 is that:

$$\mathbf{LA} \vdash \text{Cover}\left([0], \begin{bmatrix} 0 \\ \\ 0 \end{bmatrix}\right).$$

which Gentzen's Style proof associated is

$$\dfrac{\dfrac{\dfrac{\dfrac{\longrightarrow \neg e\,([0],i,j) = 1}{\longrightarrow \neg e\,([0],i,j) = 1, \quad \alpha_0(1,i) = 1 \vee \alpha_0(2,j) = 1}\,\text{Weak-Right}}{\longrightarrow e\,([0],i,j) = 1 \supset \alpha_0(1,i) = 1 \vee \alpha_0(2,j) = 1}\,\supset\text{-Right}}{i \leq r([0], j \leq r([0]) \longrightarrow e\,([0],i,j) = 1 \supset \alpha_0(1,i) = 1 \vee \alpha_0(2,j) = 1}\,\text{Weak-Left(two times)}}{\forall i \leq r([0]), \forall j \leq r([0]) \longrightarrow e\,([0],i,j) = 1 \supset \alpha_0(1,i) = 1 \vee \alpha_0(2,j) = 1}\,\forall\text{-Left(two times)}$$

Before to continue with the proof of Claim A.1.2 we introduce a few abbreviation used in the following claims.

Abbreviations (for more detail see Section 4.1.2, and Appendix A.5):

$\sum \alpha_0 := \sum \lambda_{ij} \langle r(\alpha_0), c(\alpha_0), e(\alpha_0, i, j)\rangle,$

$\sum \alpha_0' := \sum \lambda_{ij} \langle r(\alpha_0'), c(\alpha_0'), e(\alpha_0', i, j)\rangle,$

$\sum \alpha_0 \leq \sum \alpha_0' := (\sum \alpha_0 < \sum \alpha_0') \vee (\sum \alpha_0 = \sum \alpha_0').$

Claim A.1.4 $\mathbf{LA} \vdash \sum 0_{m\times n} = 0_{\mathbb{Z}}$

PROOF: By induction on m, then we have,

Basis case $m = 1$,

$$\begin{aligned} \sum 0_{1\times n} &= \sum \lambda_{ij}\langle 1, n, e\,(0, i, j)\rangle && \text{By } \lambda\text{-Construction} \\ &= 0_{\mathbb{Z}} && \text{By Axioms A30-33 recursive def. of } \Sigma \end{aligned}$$

Inductive Step:

$$\begin{aligned} \sum 0_{m\times n} &= \sum 0_{m-1\times n} + \sum 0_{1\times n} && \text{By Matrix Arithmetic} \\ &= 0_{\mathbb{Z}} + 0_{\mathbb{Z}} && \text{By I.H.} \\ &= 0_{\mathbb{Z}} && \text{By } \mathbb{Z} \text{ Arithmetic} \end{aligned}$$

□

Claim A.1.5 $\mathbf{LA} \vdash \sum \alpha \geq 0$

PROOF: Considering that α is of size $2 \times n$ with entries over $GF(2)$, we have that every entry of $\alpha \geq 0$, i.e., $\forall i \leq r(\alpha)\ j \leq c(\alpha)\ \ e(\alpha, i, j) \geq 0$, so using λ-constructor and axioms $A30 - 33$ that define recursively $\sum$, we get $\sum \alpha = \sum \lambda\langle m, n, e(\alpha, i, j)\rangle \geq 0_{\mathbb{Z}}$ □

Claim A.1.6 $\mathbf{LA} \vdash \sum \alpha \geq \sum 0_{m\times n}$

PROOF: Follows by Claims A.1.4 and A.1.5. □

So, in particular we choose $\alpha_0 = 0_{m\times n} = [0]$ and $\alpha'_0 = \alpha$. Therefore, Claim A.1.6 is,

$$\mathbf{LA} \vdash \sum \begin{bmatrix} 0 \\ 0 \end{bmatrix} \geq \sum [0] \tag{A.7}$$

or equivalent,

$$\mathbf{LA} \vdash \sum \alpha \leq \sum \alpha'_0 \tag{A.8}$$

so,

$$\dfrac{\dfrac{\dfrac{\longrightarrow \sum \alpha_0 \leq \sum \alpha'_0}{\alpha'_0 \leq r(\alpha_0) \longrightarrow \neg\left(\text{Cover}\left([0], \alpha'_0\right), \sum \alpha_0 \leq \sum \alpha'_0\right)}\text{Weak(2 times)}}{\alpha'_0 \leq r(\alpha_0) \longrightarrow \neg\left(\text{Cover}\left([0], \alpha'_0\right) \supset \sum \alpha_0 \leq \sum \alpha'_0\right)}\supset\text{-Right}}{\longrightarrow \forall \alpha'_0 \leq r(\alpha_0)\left(\text{Cover}\left([0], \alpha'_0\right) \supset \sum \alpha_0 \leq \sum \alpha'_0\right)}\forall\text{-Right}$$

from which we get,

$$\mathbf{LA} \vdash \forall \alpha'_0 \leq r\left(\alpha_0\right)\left(\text{Cover}\left([0], \alpha'_0\right) \supset \sum \alpha'_0 \geq \sum \alpha_0\right) \tag{A.9}$$

□

This complete the proof for Case 1 when $A = [0]$ and α_0.—**end of proof of Claim A.1.2.**

Furthermore, proceeding in similar way, $\beta_0 = [0]$ satisfies Select$([0], [0])$, and only $\beta_0 = [0]$ satisfies Select, and so

Claim A.1.7

$$\mathbf{LA} \vdash \text{MaxSelect}\left([0], [0]\right)$$

PROOF: Here we proceed in a similar way of the proof of Claim A.1.2.

For every β that we choose, Select$([0], \beta)$ always is going to be valid because by construction of Select itself, more precisely,

$$\text{Select}\,(A, \beta) := \forall i, j \leq r(A) \left(\underbrace{(\beta_{ij} = 1 \supset A_{ij} = 1)}_{(1)} \wedge \forall k \leq r(A) \underbrace{(\beta_{ij} = 1 \supset \beta_{ik} = 0 \wedge \beta_{kj} = 0)}_{(2)} \right)$$

where (1) and (2) always are true, because:

- a: First of all, in term (2) of Select formula, the subformula $(\beta_{ik} = 0 \wedge \beta_{kj} = 0)$ can be viewed like $\beta_{ij} = 0$ because $\beta_{1\times 1} = [0]$.
- b: Secondly, in term (1) of Select we have two cases for β on Select $([0], \beta)$:
  - b_1: $\beta_{ij} = [0]$, this case follow from the truth assignment of Select $([0], [0])$ formula.
  - b_2: $\beta_{ij} = [1]$, in this particular case, the truth value of *Select* $([0], [1])$ is false, which appear to be contrary to the assert that for all $\beta \leq r([0])$ *Select* $([0], \beta)$, but this particular case just can not happen because can not exist a nonzero $(i, j)-$entry on β without a nonzero $(i, j)-$entry on A, because β is constructed, i.e., defined, from exploring the entries of A. Similarly, can not happen that $\beta_{ij} = 0 \supset A_{ij} = 1$ which it is identically to this case but considering $A_{ij} = [1]$ (see below, Case 2). So, the only possible situation of term (1) of Select is with $\beta_{ij} = 0 \supset A_{ij} = 0$, which is Case b_1. Similarly, $\beta_{ij} = 1 \supset A_{ij} = 1$ which is

consider in Case 2 (see below, Case 2).

So,

$$\mathbf{LA} \vdash \forall \beta \leq r([0]) \;\; \text{Select}\,([0], \beta)\,.$$

Hence, we have that still the same argument that Select is valid on MaxSelect$([0], \beta)$, and in particular with $\beta = [0]$, therefore our claim follows. □

Finally, $\sum \alpha_0 = 0 = \sum \beta_0$, and our Theorem A.1.1 follows–**end Basis Case** $A = [0]$**.**

- **Case 2:** $A = [1]$, so, the possible minimum covers are (we are discarding the case in which α_0 is all ones, because it isn't necessary to use two lines to cover a 1 on A.), hence, we have the following claim,

  Claim A.1.8

$$\mathbf{LA} \vdash \text{MinCover}\,([1], \alpha_0) \supset \left(\alpha_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \vee \alpha_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right)$$

  Proof: Let A be the single value [1], i.e., $\forall i, j \leq r(A) \;\; A_{ij} = 1$. So, by construction of α we have that $\alpha_0(1, i) = 1$ or $\alpha_0(2, j) = 1$, and by hypothesis actually we are in the case of α_0 be minimum, hence the claim follows. □

  Which in both cases we have a Cover, that is, Cover$([1], \alpha_0)$ is true, and since in both cases $\sum \alpha_0 = 1$, it is in fact a *minimum cover*, i.e., MinCover, therefore

Claim A.1.9

$$\mathbf{LA} \vdash \text{MinCover}\,([1], \alpha_0)$$

PROOF: The proof is identically to Claim A.1.2. □

Note that Claim A.1.9 is true wherever α_0 is.

On the other hand, $\beta_0 = [1]$ such that Select$([1], [1])$ is valid, and only $\beta_0 = [1]$ satisfies Select, and so

Claim A.1.10

$$\mathbf{LA} \vdash \text{MaxSelect}\,([1], [1])$$

PROOF: Identically to Claim A.1.7. □

Finally, $\sum \alpha_0 = 1 = \sum \beta_0$, and our Theorem A.1.1 follows—**end Basis Case** $A = [1]$**.**

This complete the proof of Theorem A.1.1 for the Basis Case.

Inductive Step: Suppose $\texttt{KMM}(A, n)$ holds; we shall show that $\texttt{KMM}(A, n+1)$ holds as well. So, $|A| = n + 1$,intended meaning is $(r(A) = n + 1 \wedge c(A) = n + 1)$, and suppose we have MinCover(A, α) and MaxSelect(A, β).

We start proving the first inequality, i.e.,

$$\forall A \leq n + 1 \;\; \text{MinCover}(A, \alpha) \wedge \text{MaxSelect}(A, \beta) \rightarrow \sum \alpha \geq \sum \beta. \tag{A.10}$$

To do this we proceed by proving the Claim A.1.11 and Claim A.1.12.

Claim A.1.11 *Let X, Y be 0-1 matrices, then*

$$\mathbf{LA} \vdash (\forall i \leq \max\{|X|,|Y|\} \;\; \forall j \leq \max\{|X|,|Y|\} \, (X_{ij} = 1 \supset Y_{ij} = 1)) \supset \sum X \leq \sum Y$$

PROOF: By induction on the size of the matrices.

Basis case: $|X| = |Y| = 1$; here we have the following cases:

$X_{11} = 1 \supset Y_{11} = 1$, so $\sum X = 1 = \sum Y$, or

$X_{11} = 0 \supset Y_{11} = 0 \vee Y_{11} = 1$, so $\sum X \leq \sum Y$.

And finally, $X_{11} = 1 \supset Y_{11} = 0$, which is not possible because, semantically speaking, it means that exist a line that is used to cover a position of A which has a zero, which is a contradiction because lines are used to cover 1s.

Inductive step: here we know by inductive hypothesis that

$$\forall i, j \left((X[1|1])_{ij} = 1 \supset (Y[1|1])_{ij} = 1 \right) \supset \sum X[1|1] \leq \sum Y[1|1]$$

Where the notation means that the matrices are expressed without a row and without a column, that is the standard notation for the Principal Minor of X and Y. More specifically, $A[-|k]$ denotes that only the k-th column has been deleted, similarly, $A[l|-]$ denotes that only the l-th row has been deleted, and $A[-|-] = A$.

Also, the structure of X and Y are such that:

$$\mathbf{X} = \begin{pmatrix} x_{11} & R_X \\ S_X & M_X \end{pmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{pmatrix} y_{11} & R_Y \\ S_Y & M_Y \end{pmatrix}$$

Applying the I.H. we get that:

$$x_{11} \leq y_{11}, \sum R_X \leq \sum R_Y, \sum S_X \leq \sum S_Y, \sum M_X \leq \sum M_Y$$

So,

$$\begin{aligned} \sum X &= x_{11} + \sum R_X + \sum S_X + \sum X[1|1] \\ &\leq y_{11} + \sum R_Y + \sum S_Y + \sum Y[1|1] \\ &= \sum Y \end{aligned}$$

□

Note that Claim A.1.11 is easy to prove because the mapping between entries of X and Y is just the identity map, for instance, we denote it here by f, so, entry (i, j) of X is mapped to entry $f(i, j) = (i, j)$ of Y, but in the case of α which is a $2 \times c(A)$ 0-1 matrix and β which is a $c(A) \times c(A)$ matrix, where $c(A)$ was arbitrary chosen to be the $\texttt{min}\{r(A), c(A)\}$; the mapping is:
an entry (i, j) of β gets mapped to

$$f(i,j) = \begin{cases} (1, i) & \text{if in the } i\text{th row goes an horizontal line} \\ (2, j) & \text{if in the } j\text{th column goes a vertical line} \end{cases}$$

of α. Here the mapping is still one-to-one because there is one 1 per row and per column in β. This argument is formalized in the next claim.

Claim A.1.12 *Let β and α be defined like equation A.1. Then we can "match" every 1 that belong to β with a unique 1 in α. That is, $\forall i, j \leq c(\beta)$ if $\beta_{ij} = 1$, then*

$(\alpha(1,i) = 1 \vee \alpha(2,j) = 1)$. *More formally in* $\mathcal{L}_{\mathbf{LA}}$ *language this is:*

$$\begin{aligned} \mathbf{LA} \vdash & |A| \leq n, \text{MinCover}(A,\alpha), \text{MaxSelect}(A,\beta) \longrightarrow \\ & \longrightarrow \forall i,j \leq c(\beta)\ \ (\beta_{ij} = 1 \supset (\alpha(1,i) = 1 \vee \alpha(2,j) = 1)) \end{aligned} \tag{A.11}$$

PROOF: We proceed by cases,

- let $n = 1$ there are two possibilities for β, i.e., $\beta_0 = [0]$ or $\beta_0 = [1]$.
  - **Case 1:** $\beta_0 = [0]$, here the claim is vacuously true.
  - **Case 2:** $\beta_0 = [1]$, again, the analogous cases, that is, then since $\text{MinCover}(A, \alpha_0)$ true,
    $$\alpha_0 = \begin{bmatrix} 1 \\ \\ 0 \end{bmatrix} \quad or \quad \alpha_0 = \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix}$$
    i.e., $\alpha_0(1,1) = 1 \vee \alpha_0(2,1) = 1$, so, Claim A.1.12 follows.
- let $n > 1$, that is, β be an $r(A) \times c(A)$ 0-1 matrix we have two cases:

  a: if $\beta_{ij} = 0$ nothing to do.

  b: if $\beta_{ij} = 1$ we want to show that there must be a 1 in positions $\alpha\,(1,i)$ or $\alpha\,(2,j)$. But, we have by $\text{MaxSelect}(A,\beta)$ that β is such that $\beta_{ij} = 1$ then $A_{ij} = 1$, and by $\text{MinCover}(A,\alpha)$, $A_{ij} = 1$, so, we have α of the form $\alpha(1,i) = 1$ or $\alpha(2,j) = 1$, therefore Claim A.1.12 follows.

□

So, from Claim A.1.11, considering β like X, and α like Y, and Claim A.1.12 we **conclude the first size of the inequality of** $\texttt{KMM}(A, n+1)$.

Now we show the other inequality, i.e.,

$$\forall A \leq n+1 \;\; \text{MinCover}(A,\alpha) \wedge \text{MaxSelect}(A,\beta) \longrightarrow \sum \alpha \leq \sum \beta. \tag{A.12}$$

We consider the two possible cases here:

- **Case 1.** There is a minimal proper covering– a covering without all rows and without all columns.
- **Case 2.** There isn't a minimal proper covering.

We are going to proceed in similar way of the proof of Theorem 4.2.1 of Chapter 4 Section 4.2.

- **Case 1:** in this case we have a covering where at least one row is missing and at least one column is missing. So, α is such that has at least a two columns missing, because that represent, at least, the horizontal line and the vertical line which were missed. On the other hand, by hypothesis exists an α such that $\text{MinCover}(A,\alpha)$ is valid, and like we assume that A has a proper covering composed of e horizontal lines (rows) and f vertical lines (columns), that is, a cover of size $e+f$, so, we have that $\sum \alpha = e + f$ where

$$e = \sum \alpha[2|-] \qquad \text{i.e., } \alpha[2|-] := \text{first row of } \alpha$$
$$f = \sum \alpha[1|-] \qquad \text{i.e., } \alpha[1|-] := \text{second row of } \alpha$$

Recall that here we are considering matrix of sizes greater or equal than 2×2, because cases in which $e = 1$ or $f = 1$ were considered in the Basis Case of Theorem A.1.1, therefore, just note that, $e \geq 1$ and $f \geq 1$, i.e., $e + f \geq 2$.

Also, recall that α is a 0-1 matrix of size $2 \times |A|$, but with at least two missed columns, that is, $|\alpha| \leq |2 \times (|A| - 2)|$, in other words, we have that $(e < n + 1)$, and $(f < n + 1)$.

Now, consider the matrix PAQ, where P and Q are permutation matrices[7] such that our 0-1 matrix A assume[8] the form

$$PAQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0_{(n+1)-e \times (n+1)-f} \end{array}\right],$$

where:

Sub Matrix	*Dimension*
E	$e \times f$
A_1	$e \times (n + 1 - f)$
A_2	$(n + 1 - e) \times f$
O	$(n + 1 - e) \times (n + 1 - f)$

That is, we permute lines of A so that the e horizontal lines and the f vertical lines are in the initial square denoted by E. Note that the e's are the horizontal lines associate to A_1's row, while f's are vertical lines associate to A_2's columns.

Therefore, we have that size of $|A_1| < |A|$ this is because $\texttt{min}\{e, n+1-f\} < n+1$, and this is clear because $f \geq 1$ and $e < n+1$, similarly, $|A_2| < |A|$, therefore, we

[7]See for more details about permutation matrices in Chapter 3, and Appendix A.3.
[8]We are using square matrices to simplify the calculus.

apply the inductive hypothesis (I.H.) to A_1 and A_2. Let us make this inductive step more explicitly, this is important because it is here in this stage of the proof, that we need to apply Π_2^B-Induction, i.e., it is here where we use the fact that our induction is over $\forall A \leq n \;\; \texttt{KMM}(A, n) \in \Pi_2^B$. Therefore,

- The matrix A_1 has e horizontal lines (rows), so, by I.H. there are at least e's many 1s no two on the same line on A_1 denoted by $\sum \beta_{A_1}$, hence, $\sum \alpha[2|-] \leq \beta_{A_1}$. That is,

$$\overbrace{\forall A_1 \leq n \;\; \text{MinCover}(A_1, \alpha) \wedge \text{MaxSelect}(A_1, \beta) \longrightarrow \sum \alpha \leq \sum \beta}^{\texttt{KMM}(A_1, n)}$$

  Because if we suppose that A_1 can be cover with fewer than $\sum \alpha[2|-]$ lines, say $e' < \sum \alpha[2|-]$, then $f + e' < f + e = \sum \alpha$ giving us a contradiction of the fact that $\text{MinCover}(A, \alpha)$.

- On the other hand, the matrix A_2 has f vertical lines (columns), so, by I.H. there are at least f's many 1s no two on the same lines on A_2 denoted by $\sum \beta_{A_2}$, hence, $\sum \alpha[1|-] \leq \sum \beta_{A_2}$. That is,

$$\overbrace{\forall A_2 \leq n \;\; \text{MinCover}(A_2, \alpha) \wedge \text{MaxSelect}(A_2, \beta) \longrightarrow \sum \alpha \leq \sum \beta}^{\texttt{KMM}(A_2, n)}$$

  Because if we suppose that A_2 can be cover with fewer than $\sum \alpha[1|-]$ lines, say $f' < \sum \alpha[1|-]$, then $f' + e < f + e = \sum \alpha$ giving us a contradiction of the fact that $\text{MinCover}(A, \alpha)$.

Note that this Π_2^B proof is not feasible, and this is, fundamentally, because we

have to reapply different permutations matrices in every inductive step.

Putting all together, and considering that exactly e many horizontal lines cover A_1, i.e., $\sum \alpha[2|-] = e$, and f many vertical lines cover A_2, i.e., $\sum \alpha[1|-] = f$, so, we have

$$\begin{aligned}
\sum \alpha &= e + f && \text{By Def. } \alpha \\
&= \sum \alpha[2|-] + \sum \alpha[1|-] && \text{By Def. of } e \text{ and } f \\
&\leq \sum \beta_{A_1} + \sum \beta_{A_2} && \text{By } \Pi_2^B\text{-Ind} \\
&\leq \sum \beta && \text{By (*)}
\end{aligned}$$

where the last inequality $(*)$ comes from the fact that, let us define S to be a set of 1s, so, if S_1 is a subset of $1s$ of A_1 no two on the same line, and S_2 is a subset of $1s$ no two on the same line of A_2, then $S = S_1 \cup S_2$ is a subset of $1s$ of A no two on the same line—as A_1 and A_2 occupy different lines of A, in other terms, $S_1 \cap S_2 = 0$. Hence, the inequality symbol follows since E has $1s$ no two on the same lines while A_1 and A_2 have some lines full of zeros.

End of the second inequality A.12—Case 1.

- **Case 2:** it is when α "is" given by the form of Figure A.1, that is one of the rows of α is full of zeros and the other one is full of ones, so, $\sum \alpha = c(A)$. To be more precisely, in this case, we choose arbitrarily to have all horizontal lines or all vertical lines, because to cover all the 1s no two on the same line we know for sure that we need $c(A)$ or $r(A)$ lines, and does not matter if there are a mix of vertical and/or horizontal, we decide to use all vertical or all horizontal lines.

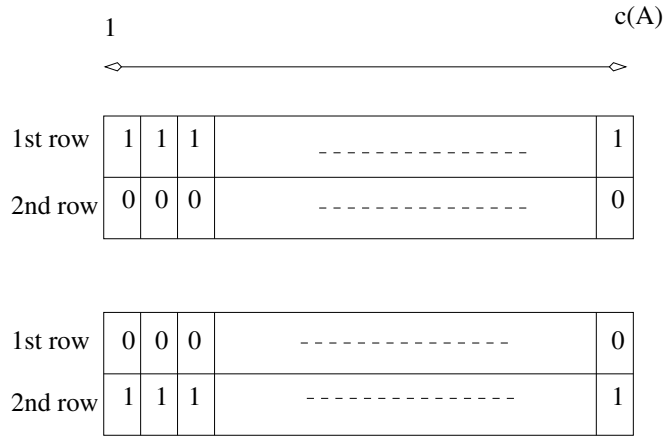


Figure A.1: Two possible α's when there is NOT a minimal proper covering

So, there exists an α, say α_0, such that MinCover(A, α_0), and either row 1 is full of ones and row 2 full of zeros, or row 2 is full of ones, and row 1 full of zeros. If row 1 is full of ones, we have all horizontal lines in which case we choose $\texttt{min}\{r(A), c(A)\} = r(A) = m$. Similar in case that row 2 is full of ones, i.e., $\texttt{min}\{r(A), c(A)\} = c(A) = n$. Again, we can consider square matrices to make more easy the calculus, so we use indistinguishable m or n, except when it is necessary denote the difference.

So we proceed by induction on n, the size of the matrices.

– **Basis Case:** Here we have two situation of α,

$$\alpha_0 = \begin{bmatrix} 1 \\ \\ 0 \end{bmatrix} \quad \text{or} \quad \alpha_0 = \begin{bmatrix} 0 \\ \\ 1 \end{bmatrix}$$

i.e., $\alpha_0(1,1) = 1 \vee \alpha_0(2,1) = 1$. Hence[9], there must be a 1 on β_0, i.e.,

[9]Recall that when we work with a *general* matrix variable, we use β, but when we work with *particular* matrix we use β_0.

$\beta_0 = [1]$, because, otherwise we are selecting an horizontal or vertical line to cover a one that does not exist, arriving to a contradiction. Also by Claim A.1.10 we have that MaxSelect(A, β_0) is true in **LA**, so, finally $\sum \beta_0 \geq \sum \alpha_0$, that is, equation A.12 holds.

– **Inductive Step:** the proof consist of Claim A.1.13 and Claim A.1.14. Without loss of generality[10] we choose one of the two possible configurations of α, that is, first row all ones and second row all zeros or the other way around. We choose α_0 first row full of ones, and also this α_0 is such that MinCover(A, α_0) is true on **LA** by the following claim:

Claim A.1.13 $\mathbf{LA} \vdash \forall A \leq n$

$$\left(\text{MinCover}\,(A, \alpha_0) \supset \left(\alpha_0 = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{bmatrix} \vee \alpha_0 = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \end{bmatrix} \right) \right)$$

Proof: By induction on n and using the Claim A.1.8, the result follows. □

So, we want to show that the $n \times n$ matrix β_0 has n 1's considering that α_0 is a $2 \times n$ matrix.

We know that β_0 has at least a 1 somewhere, otherwise $\sum \beta_0 = 0$ which means that $A = 0$, i.e., A is a full of zeros matrix, and therefore α_0 must be zero, that is, $\sum \alpha_0 = 0$, which is a contradiction of the two

[10] We can proceed in this way, because we prove, many times before, the symmetric results of both configuration of α, that is, both configuration yield the same result.

possible configurations of α_0. So, proceeding in similar way that proof of Theorem 4.2.1 Chapter 4 Section 4.2.

We permute[11] β_0 in such a way that it is in the form

$$\hat{\beta}_{\mathbf{0}} = \begin{bmatrix} 1 & R_{\beta_0} \\ S_{\beta_0} & M_{\beta_0} \end{bmatrix} \tag{A.13}$$

and consider the following claim:

Claim A.1.14 *Let β be a 0-1 matrix such that after permutation it get the form A.13, and let α be given by one of the two configurations of Figure A.1 then if $\hat{\beta}_0$ does not have a proper covering, then its principal submatrix M_{β_0} neither.*

PROOF: Considering one of the two configurations of α_0, so we have that there are n 1s denoting the n-lines on A which is an $n \times n$ matrix, in other words,we are in the case where A does not have a proper covering, so, by construction those 1s correspond to nonzero entries of β_0, which again is a $n \times n$ matrix, hence, it does not have a proper covering. Now we know that after permutation we get that $\hat{\beta}_0(1,1) = 1$, then by construction of β_0, $R_{\beta_0} = 0$, and $S_{\beta_0} = 0$. Then by permutation on the form A.13, the principal submatrix of $\hat{\beta}_0$, that is,M_{β_0} has size $(n-1 \times n-1)$ which correspond to the $(n-1)$ 1s, otherwise contradict the fact that $\hat{\beta}_0$ does not have a proper covering. Therefore M_{β_0} has not a proper covering. □

Now, we have that α_0 is not a proper covering, so, in particular $\alpha_0[-|j]$, where $[j] = n$, and where the intended meaning is represent α_0 without the

[11] See for more details about permutation matrices in Chapter 3, and Appendix A.3.

jth column, so, the new $\alpha_0[-|j]$ is of size $2 \times n - 1$, and it is not a proper covering.

Also, we know that the $\beta_0(1,1)$ entry must has associate a line horizontal or vertical, indistinguishably, which is denoting by a particular column of α_0, that is, if the one in position $\hat{\beta}_0(1,1)$ before permuting β_0 correspond to the entry, say (p,q), then α_0 is going to has a 1, either in $\alpha_0(1,p)$ or $\alpha_0(2,q)$, see Figure A.2.

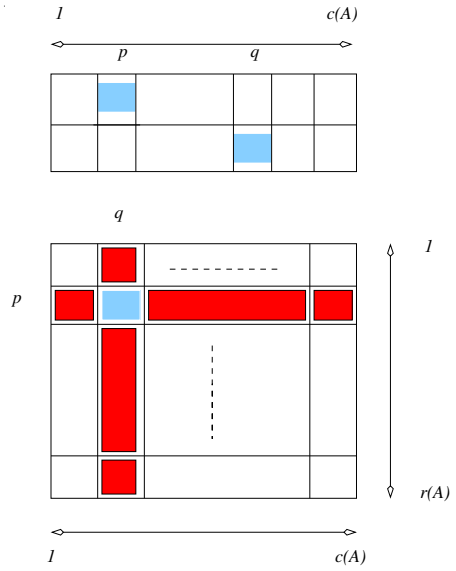


Figure A.2: Fist matrix (up) correspond to α_0 with two possible lines according to a particular β_0 matrix entry depicted in the second place (down).

So, in particular, without loss of generality, we can choose $\alpha_0[-|q]$ meaning α_0 without the qth column, which still it is not a proper covering.

Therefore, considering $\alpha_0[-|q]$ and $\beta_0[1|1]$, we have

$$\begin{aligned}
\sum \hat{\beta}_0 &= \sum \beta_0 && \text{By permutation} \\
&= \sum \beta_0[1|1] + 1 && \text{By matrix arithmetic} \\
&\geq \sum \beta_0[1|1] + \alpha_0(2,q) && \text{By Basis Case } ^{12} \\
&\geq \sum \alpha_0[-|q] + \alpha_0(2,q) && \text{By I.H. and Claim A.1.14} \\
&= \sum \alpha_0 && \text{By matrix arithmetic}
\end{aligned}$$

End of the second inequality of A.12—Case 2

This ends of Π_2^B-Proof of $\forall A \leq n$ $\texttt{KMM}(A,n)$ □

Conclusion, expressed informally, is that

$$\Pi_2^B\mathbf{LA} \vdash \text{König Min-Max Theorem}$$

Note that in the previous proof, we made use of permutations in order to rearrange the entries of a particular matrix, in order to make easier to deal within the proof with concepts like cover or number of $1s$ no two on the same line, and mainly to see graphically how the induction works inside the proof.

We finish this appendix with a concept that should be explicitly establish, which is that every time that we apply some permutation to our matrix, we "preserve" the cover that we are working on it. This was described in Chapter 3 Section 3.2 Lemma 3.2.2, where this concept of preserve was introduced the concept of *order preserving*.

[12] Without loss of generality, we can choose, $\alpha_0(1,p)$ instead of $\alpha_0(2,q)$, because denote one of the two possibilities to cover a 1 on A_{pq}.

This idea is important with respect to the correctness of our Permutation-Based algorithm—see Appendix A.3,and we formalize this idea inside the **LA**-Theory, so, we start with the following definition:

Definition A.1.1 *Let A and β be a 0-1 matrices of sizes $m \times n$, we say that β is a selection of A, denoted by $\beta \sqsubseteq A$ if $\forall i \leq r(\beta)\ \forall j \leq c(\beta)\ \ (\beta_{ij} = 1 \supset A_{ij} = 1)$*

Lemma A.1.1 *Let P and Q two permutation matrices. If $\mathbf{LA} \vdash \beta \sqsubseteq A$ then $P\beta Q \sqsubseteq PAQ$.*

Proof: Firs of all, we use a direct correspondence between permutations—like bijections—$\pi : [m] \longrightarrow [m]$ and $\tau : [n] \longrightarrow [n]$, and permutation matrices, that is, P_π and Q_τ which permutes, for instance, rows and columns of β, respectively. In order to make easier the calculus we can consider $\pi : [n] \longrightarrow [n]$, i.e., working over β and A like square matrices. The matrix P_π is obtained from the identity matrix by exchanging the rows according to π, and the matrix Q_τ is obtained from the identity matrix by exchanging the rows according to τ. Then $(P_\pi \beta Q_\tau)_{ij} = (\beta)_{\pi^{-`}(i)\tau^{-1}(j)}$. Our permutations P, Q work in such a way that they place the 1s on the main diagonal of β in the original order; that is, if the 1s of β where in positions:

$$(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$$

with $i_1 < i_2 < \cdots < i_k$, and remember that $k \leq n$, where[13] $n = |A|$, because β is a selection of A. Then our permutations π, τ are given as follows:

[13] $|A| = n$ intended meaning is $(r(A) = n \wedge c(A) = n)$

$$\begin{array}{rcl@{\qquad}rcl} i_1 & \mapsto & 1 & j_1 & \mapsto & 1 \\ i_2 & \mapsto & 2 & j_2 & \mapsto & 2 \\ & \vdots & & & \vdots & \\ i_k & \mapsto & k & j_k & \mapsto & k \end{array}$$

By definition are order preserving, and according to rows.

So, we want to show that $P_\pi \beta Q_\tau \sqsubseteq P_\pi A Q_\tau$. Let the entry (p, q) of $P_\pi \beta Q_\tau$, that is, $(P_\pi \beta Q_\tau)_{pq} = \beta_{\pi^{-1}(p)\tau^{-1}(q)}$, then must exist $a \in [n]$ such that $i_a = \pi^{-1}(p)$ and $j_a = \tau^{-1}(q)$. On the other hand the value of β_{i_a,j_a} can be 0 in such a case, we do nothing, or can be the case that is 1, then we have the following derivation

$$\begin{array}{lll} (P_\pi \beta Q_\tau)_{pq} = \beta_{\pi^{-1}(p)\tau^{-1}(q)} & & \text{By Permutation} \\ = \beta i_a, j_a & & \text{By bijection} \\ = 1 & & \text{By case} \\ \sqsubseteq A_{i_a,j_a} & & \text{By hypothesis} \\ = A_{\pi^{-1}(p)\tau^{-1}(q)} & & \text{By bijection} \\ = (P_\pi A Q_\tau)(\pi(\pi^{-1}(p))(\tau(\tau^{-1}(q)))) & & \text{By Permutations} \\ = (P_\pi A Q_\tau)_{pq} & & \text{By bijection.} \end{array}$$

□

Lemma A.1.2 *Let P and Q two permutation matrices. If $LA \vdash cover(A, \alpha)$, where $\alpha := \begin{bmatrix} \alpha_{11} \\ \alpha_{21} \end{bmatrix}$, then $\alpha' := \begin{bmatrix} P\alpha'_{11} \\ \alpha'_{21}Q \end{bmatrix}$ is a cover of PAQ, i.e., $LA \vdash cover(A, \alpha')$.*

Proof: The proof is identical to Lemma 4.3.1 Chapter 4 Section 4.3.3. □

A.2 An LA proof of Lemma 4.3.1 Section 4.1

Given a matrix A, let l_A and o_A denote the minimum number of lines necessary to cover all the 1s of A, and the maximum number of 1s no two on the same line, respectively. (Of course, König's Min-Max Theorem says that for all A, $l_A = o_A$.) In terms of the definitions just given, we have that $l_A = \Sigma\alpha$ where MinCover(A, α), and $o_A = \Sigma\beta$ where MaxSelect(A, β). (See definition of MinCover and MaxSelect, Chapter 4 Section 4.3.2.)

Lemma A.2.1 *(Lemma 4.3.1) Given a matrix A, and given any permutation matrix P, we have*

- $\mathbf{LA} \vdash l_{PA} = l_{AP} = l_A$
- $\mathbf{LA} \vdash o_{PA} = o_{AP} = o_A$

That is, these four equalities can be proven in **LA***, i.e., with induction restricted to formulas without matrix quantifiers.*

Proof: The proof follows from—see below—Claim A.2.1 and Claim A.2.2, and their respectively corollaries.

We adopt bijections, say π and τ, to describe permutation matrices, say P, such that $\pi : [n] \to [n]$ and $\tau : [n] \to [n]$, and as permutation matrix P_π or P_τ, according if we use it multiplying to the left or to the right of A, respectively. That is, P_π permute the rows of matrix A, and P_τ permutes the columns of matrix A. The matrix P_π (P_τ) is obtained from the identity matrix by exchanging the rows (columns) according to π (τ). Then $(P_\pi A P_\tau)_{ij} = (A)_{\pi^{-1}(i)\tau^{-1}(j)}$. Also, we omit the subscript π or τ when the context is clear.

Therefore, P_π and P_τ work in such a way that they place the 1s on the main diagonal in the *original* order , for instance, according to the rows of A.

Notation: Let X be a matrix,

$$X[i : -] := i\text{-th row of } X$$

$$X[- : j] := j\text{-th column of } X$$

Recall that the proof of Lemma A.2.1 follows from the next sequence of claims and corollaries.

Claim A.2.1

$$\mathbf{LA} \vdash \text{Cover}(A, \alpha) \longrightarrow \text{Cover}(PA, \alpha') \wedge \text{Cover}(AP, \alpha')$$

Proof: Suppose that $\text{Cover}(A, \alpha)$ holds; we show that **LA** reasoning is sufficient to show that $\text{Cover}(PA, \alpha')$ holds, where conforming to how we apply to matrix A the permutation matrix P we define:

- If PA is the case, let $\alpha' := \alpha[1 : -]P$, that is, α' is the same as α except the first row of α, defined by $\alpha[1 : -]$ is replaced by $\alpha[1 : -]P$.
- If AP is the case, let $\alpha' := \alpha[2 : -]P$, that is, α' is the same as α except the second row of α, defined by $\alpha[2 : -]$ is replaced by $\alpha[2 : -]P$.

The idea is that if row i of A is moved to row i' by P, then column $\alpha[1 : i]$ was part of the cover α then now column i', i.e., $\alpha[1 : i']$ is part of cover α' conforming to P, that is, $\alpha' := \alpha[1 : -]P$. Symmetrically with the j column of A, and the second row of α.

We have two possible lines that go through the non-zero entries of A, say (i, j); which in terms of α' entries, that means that either $\alpha_{(1,i)} = 1$ or $\alpha_{(2,j)} = 1$. When the context is clear, we shall use indistinguishably the follow notation for the entry of a matrix X, X_{ij}, or $X_{(i,j)}$, or $X(ij)$.

Therefore, we have the following cases:

- **Case 1:** the line on position (i, j) of A is horizontal. Hence, we have $\alpha_{(1,i)} = 1$, and of course $A_{ij} = 1$. Consider any entry (p, q) of $P_\pi A$, i.e., $(P_\pi A)_{pq} = (A)_{\pi^{-1}(p)q}$. If $(A)_{\pi^{-1}(p)q} = 0$ nothing to prove, otherwise its means that there exist an $a \in [n]$ such that $i_a = \pi^{-1}(p)$. Hence,

$$
\begin{aligned}
(P_\pi A)_{pq} &= A_{\pi^{-1}(p)q} && \text{(a)}\\
&= A_{i_a q} && \text{(b)}\\
&= 1 && \text{(c)}\\
&\supset \alpha_{(1,i_a)} = 1 \vee \alpha_{(2,q)} = 1 && \text{(d)}\\
&= \alpha_{(1,\pi^{-1}(p))} = 1 \vee \alpha_{(2,q)} = 1 && \text{(e)}\\
&= \alpha_{(1,\pi(\pi^{-1}(p)))} P_\pi = 1 \vee \alpha_{(2,q)} P_\pi = 1) && \text{(f)}\\
&= \alpha_{(1,p)} P_\pi = 1 \vee \alpha_{(2,q)} P_\pi = 1 && \text{(g)}\\
&= \alpha'_{(1,p)} = 1 \vee \alpha'_{(2,q)} = 1. && \text{(h)}
\end{aligned}
$$

  where (a) is by rows permutation, (b) is by π-bijection, (c) is because we are in case 1, (d) by hypothesis, (e) by bijection, (f) by rows permutation, (g) by bijection, and (h) by notation, note that in this step $\alpha_{(2,q)} P_\pi = \alpha'_{(2,q)}$ because we are permuting over the first row of α so, $\alpha[2 : -] = \alpha'[2 : -]$.

Therefore, $\text{Cover}(PA, \alpha')$, where $\alpha' := \alpha[1 : -]P_\pi$, holds. That is,

$$\mathbf{LA} \vdash \text{Cover}(A, \alpha) \rightarrow \text{Cover}(PA, \alpha').$$

- **Case 2:** the line on position (i, j) of A is vertical. This case is symmetrically to case 1, but, instead of $P_\pi A$ we use AP_τ, and work over the columns of A. Deriving on

  $$(AP_\tau)_{pq} \supset \alpha'_{(1,p)} = 1 \vee \alpha'_{(2,q)} = 1.$$

  Therefore, $\text{Cover}(AP, \alpha')$, where $\alpha' := \alpha[2 : -]P_\tau$, holds. That is,

  $$\mathbf{LA} \vdash \text{Cover}(A, \alpha) \rightarrow \text{Cover}(AP, \alpha').$$

Finally by cases 1 and 2 our Claim A.2.1 follows. □

By a consequence of Claim A.2.1, we have the following corollary, which basically says that having a minimum cover is *closed under permutation*. That is, if A has a minimum cover, say α_0, then either $P_\pi A$ or AP_τ have $\alpha'_0 := \alpha_0[1 : -]P_\pi$ or $\alpha'_0 := \alpha_0[2 : -]P_\tau$ like their minimum covers, respectively, and where the subindex 0 on a matrix means that we work with *particular* matrix, i.e., α_0.

Corollary A.2.1

$$\mathbf{LA} \vdash \text{MinCover}(A, \alpha) \rightarrow \text{MinCover}(PA, \alpha') \wedge \text{MinCover}(AP, \alpha')$$

PROOF: Suppose that $\text{MinCover}(A, \alpha)$ holds. The idea of the proof is the following, we are going to show that **LA** reasoning is sufficient to show that $\text{MinCover}(PA, \alpha')$

holds. Following the same approach used in Claim A.2.1, we split it into two cases, one permuting rows of A, and working over $\alpha[1:-]$, and the other permuting columns of A, and working over $\alpha[2:-]$.

Finally, in both cases we have that by Claim A.2.1, and by hypothesis we get:

$$\mathbf{LA} \vdash \text{MinCover}(A, \alpha) \rightarrow \text{MinCover}(PA, \alpha')$$

and

$$\mathbf{LA} \vdash \text{MinCover}(A, \alpha) \rightarrow \text{MinCover}(AP, \alpha')$$

where if PA is the case, $\alpha' := [1:-]P_\pi$, and if AP is the case $\alpha' := \alpha[2:-]P_\tau$. □

We shall proceed using the same definition and notation of permutation matrices and bijections that we used on proof of Claim A.2.1 in the following claims.

Claim A.2.2

$$\mathbf{LA} \vdash \text{Select}(A, \beta) \rightarrow \text{Select}(PA, \beta') \wedge \text{Select}(AP, \beta')$$

where if PA is the case, $\beta' := P_\pi\beta$, otherwise $\beta' := \beta P_\tau$.

Proof: We show that **LA** reasoning is sufficient to prove that a selection of the permutation matrix of A holds, i.e., $\text{Select}(PA, \beta')$ holds, where $\beta' := P_\pi\beta$ is the result of rearranging the rows of β according to the permutation P. The idea is that if a row i is moved to row i', then if row i was part of the selection, then row i' is part of the selection according to $P_\pi A$. And symmetrically, βP_τ is the result of rearranging th columns of β according to the permutation P.

We split the analysis in two cases, according to if we permute rows or if we permute columns of A.

- **Case 1:** we apply P_π in order to permute the rows of A. From the definition of predicate Select, we have to prove two statements— A.14, A.15:

$$(\beta')_{pq} = 1 \rightarrow (P_\pi A)_{pq} = 1 \tag{A.14}$$

Consider any (p, q) of $P_\pi\beta$, i.e., $(P_\pi\beta)_{pq} = \beta_{\pi^{-1}(p)q}$. If $\beta_{\pi^{-1}(p)q} = 0$ there is nothing to prove, otherwise exists $a \in [n]$ such that, $i_a = \pi^{-1}(p)$. Therefore,

$$\begin{aligned}
(\beta')_{pq} &= (P_\pi\beta)_{pq} && \text{(a)}\\
&= \beta_{\pi^{-1}(p)q} && \text{(b)}\\
&= \beta_{i_a q} && \text{(c)}\\
&= 1 && \text{(d)}\\
&\supset A_{(i_a,q)} && \text{(e)}\\
&= 1 && \text{(f)}\\
&= A_{(\pi^{-1}(p),q)} && \text{(g)}\\
&= (P_\pi A)_{(\pi(\pi^{-1}(p)),q)} && \text{(h)}\\
&= (P_\pi A)_{pq}. && \text{(i)}
\end{aligned}$$

where (a) is by notation, (b) is by row permutation, (c) is by π-bijection, (d) is by case, (e) is by hypothesis, (f) by case,(g) by bijection, (h) by row permutation, and (i) by bijection. Concluding with (A.14).

And, on the other hand,

$$\forall k \leq r(P_\pi A) \left((\beta')_{pq} = 1 \rightarrow \left(\beta'_{pk} = 0 \wedge \beta'_{kq} = 0 \right) \right) \tag{A.15}$$

$$\begin{aligned}
\beta'_{pq} &= (P_\pi \beta)_{pq} && \text{(a)} \\
&= \beta_{\pi^{-1}(p)q} && \text{(b)} \\
&= \beta_{i_a q} && \text{(c)} \\
&= 1 && \text{(d)} \\
&\supset \forall k \leq r(P_\pi A) \left(\beta_{(i_a,k)} = 0 \wedge \beta_{(k,q)} = 0 \right) && \text{(e)} \\
&= \forall k \leq r(P_\pi A) \left(\beta_{(\pi^{-1}(p),k)} = 0 \wedge \beta_{(k,q)} = 0 \right) && \text{(f)} \\
&= \forall k \leq r(P_\pi A) \left(P_\pi \beta_{(\pi(\pi^{-1}(p)),k)} = 0 \wedge P_\pi \beta_{(k,q)} = 0 \right) && \text{(g)} \\
&= \forall k \leq r(P_\pi A) \left(P_\pi \beta_{(p,k)} = 0 \wedge P_\pi \beta_{(k,q)} = 0) \right) && \text{(h)} \\
&= \forall k \leq r(P_\pi A) \left(\beta'_{(p,k)} = 0 \wedge \beta'_{(k,q)} = 0) \right). && \text{(i)}
\end{aligned}$$

where (a) is by notation, (b) is by row permutation, (c) is by π-bijection, (d) is by case, (e) is by hypothesis, (f) by bijection, (g) by row permutation, (h) by bijection, and (i) is by notation. Concluding with (A.15).

Therefore, from (A.14) and (A.15), case 1 follows, that is,

$$\mathbf{LA} \vdash \text{Select}(A, \beta) \rightarrow \text{Select}(P_\pi A, \beta')$$

holds, where $\beta' := P_\pi \beta$.

- **Case 2:** we apply P_τ in order to permute the columns of A. Proceeding in

symmetrically manner that case 1 (using columns), we get

$$\mathbf{LA} \vdash \text{Select}(A, \beta) \rightarrow \text{Select}(AP_\tau, \beta')$$

holds, where $\beta' := \beta P_\tau$.

Finally, by cases 1 and 2 Claim A.2.2 follows. □

By a consequence of Claim A.2.2, we have the following result

Corollary A.2.2

$$\mathbf{LA} \vdash \text{MaxSelect}(A, \beta) \rightarrow \text{MaxSelect}(PA, \beta') \wedge \text{MaxSelect}(AP, \beta')$$

where if PA is the case, $\beta' := P_\pi \beta$, otherwise $\beta' := \beta P_\tau$.

Proof: Similar to Corollary A.2.1. □

Now putting all the results together, and let A, α, β be matrices such that satisfy $\text{MinCover}(A, \alpha) \wedge \text{MaxSelect}(A, \beta)$, by Corollaries A.2.1 and A.2.2, the following two statements are satisfiable on **LA**:

- $l_A = \Sigma\alpha \quad \supset \quad l_{PA} = \Sigma P\alpha \quad \wedge \quad l_{AP} = \Sigma\alpha P$, and
- $o_A = \Sigma\beta \quad \supset \quad o_{PA} = \Sigma P\beta \quad \wedge \quad o_{AP} = \Sigma\beta P$,

this conclude with the proof of Lemma A.2.1. □

A.3 Correctness of Permutation-Based Algorithm (Chapter 3)

We start this appendix with an outline of the proof of correctness of the algorithm described in Chapter 3—that is, Permutation-Based algorithm, when we refer to "our algorithm" that means our Permutation-Based algorithm. All algorithms in this appendix are very simple and self-evident, nevertheless, we present a detailed presentation of each algorithm because we are interested in the correctness of the Permutation-Based algorithm in **LA**-Theory. First of all, we pad with zeros A_G, or just A, and M_{A_G}, or just M, that is, the adjacency matrix of a graph G, and the matrix representation of a maximum matching, so, after this padding process, we shall consider to be working on squared matrices. The proof is divided into fifth interconnected parts, more precisely,

1- Computing P from M_{A_G}—Algorithm 2—using the matrix representation of the maximum matching, we compute P , the permutation matrix that will be used to put the matrices in the following form

$$PA_GQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0 \end{array}\right] \tag{A.16}$$

.

2- Computing Q from PM_{A_G}—Algorithm 4—we use the output of the previous algorithm to be able to put the matrix in form A.16, in order to do that, we compute the permutation matrix Q to be multiply by left.

3- Computing $\vec{O}$ from PA_GQ—Algorithm 6—applying permutation matrices P, Q

to A_G we obtain A.16, so, in this algorithm define which entry among the k-th main diagonal ones of E, is black or green according to the exploration of A_1, and eventually, if the line of A_1 was full of zeros, explore A_2.

4- Computing μ from PA_GQ and $\vec{O}$—Algorithm 7—from matrix PA_GQ and the initialized orientation vector $\vec{O}$ this algorithm group into two blocks of black 1s and green 1s the kth ones of the main diagonal, and update the orientation vector $\vec{O}$ according to the grouping.

5- Finally, the recursive call: at this point we have the lines that goes over each green one in our orientation vector $\vec{O}$, so, remain compute the orientations of the lines covering the black 1s. We do so **recursively**, by repeating the procedure from **Step 2** of our algorithm—see Chapter 3 Section 3.3—with the matrix obtained from the $k \times k$ upper-left quadrant of PA_GQ, but with the following modifications: the square enclosing the green 1s is zeroed out, and we also place zeros wherever those lines crossed the rest of the quadrant. In Figure A.5, we place zeros in the upper-left quadrant wherever the horizontal red lines crosses.

Note that we are assuming that all the concepts inside this proof are formalizable in **LA**-Theory. Besides, we are supposing to be in **Step 2** of our algorithm, such that, algorithms in points $1, 2$, and 3 of the previous outline correspond to the case **Step 2a**—$(A_1 = 0 \vee A_2 = 0)$—where there is no recursion, otherwise, we make use of the fifth algorithms described in the outline, that is in **Step 2b**—$(A_1 \neq 0 \wedge A_2 \neq 0)$—which correspond to grouping ones, updating orientation vector, and making the recursive call[14]. For more about algorithms see [DFS88, Dij76, Sol12, CLRS09].

[14]We are numerating the algorithms starting from 2 because we consider the whole algorithm (the one described in Chapter 3) to be the algorithm number 1.

A.3.1 Computing Permutations

In this section we are going to study more in details how permutation matrices works in our algorithm. Recall that M_{A_G}, or just M, be the matrix representation of a maximum matching of a bipartite graph G as computed by Hopcroft-Karp Algorithm (HK-Algorithm) on input $\langle A_G \rangle$, where A_G, or just A, is the adjacency matrix of G. The order of M_A comes from the order of A_G, here is m rows and n columns. From the correctness of HK-Algorithm—Chapter 2 Section 2.3—we know that the matching $\mathcal{M}$ is maximum and it is, for instance, of size k, that is $|\mathcal{M}| = k$. (We use script $\mathcal{M}$ in order to distinguish its set representation from its matrix representation denoted with M).

On the other hand, our algorithm works using two 0-1 matrices over $\mathbb{Z}$, but we are going to do a simple padding process before that our algorithm use them, in order to work with square matrices. So, the two updated matrices are:

- The matrix representation of the maximum matching M plus a padding with zeroes of rows and columns, if it is necessary.
- The adjacency matrix representation of G plus a padding with zeroes of rows and columns, if it is necessary.

In this process of padding matrices, arbitrarily, we let n be the maximum value between the two set of vertices i.e., $\max\{|V_1|, |V_2|\}$. So, after padding the matrices, let defined the size of both matrices such that $|M| = k \times k$ where k correspond to[15] the major index of the maximum matching output of HK-Algorithm, and $|A_G| = n \times n$,

[15]Remember that $\mathcal{M} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ i.e., $\mathcal{M}$ is a list of all the positions of M with a 1 in them ($k = |\mathcal{M}|$).

where n is defined below. Therefore, after padding M and A with zeroes, we are going to be working with square matrices.

Finally, our permutations P_π and Q_τ (also we use P and Q, respectively) works in such a way that it places the 1s from the maximum matching, in the main diagonal in the original order, i.e., P, Q are order preserving (see Chapter 3 Lemma 3.2.1).

We use the following notation in our algorithms: the end of line is denote by ";", the $Z \leftarrow \emptyset$ denote initialization statement, and sometime, we use an alternative notation $Z = \emptyset$ denoting assignment statement, also, we use $Z == \emptyset$ to denote equality condition statement. We use $|\cdot|$ to denote the cardinality of a set, and $\neq$ to denote the not equality. We denote by using sub-indices the follow assign operation, for instance, if P is a 0-1 matrix of size $m \times m$, $P \leftarrow 0_m$ means assign 0 to each entry of the matrix P, similarly, if P is a vector of m entries, the same notation—according to the context—means that we assign 0 to each element of the vector P.

A.3.1.1 Computing P from M_{A_G}

Before continue with the simple algorithm to compute P, let us mention a roughly idea of how it works: on[16] input $\langle M_{A_G} \rangle$ the algorithm goes trough each row of M_{A_G} storing in two different vectors i and j the nonzero positions and the zero rows, respectively. At the end, when both vectors are filled in line 14, the function `FillPermRows` is called in order to put the 1's on P according to i and j. (See π-permutation below).

When we compute the permutation matrix P we are going to be use of the following facts, let m be the number of rows of A, i.e., $r(A)$, sometime we use $row(A)$ to clarify, and let $i[1], i[2], \ldots, i[k]$ be the rows of M that have at least one 1 in them with $i[1] < i[2] < \cdots < i[k]$ (according to rows). Let $j[k+1], j[k+2], \ldots, j[m-k]$ be

[16] We are using M_{A_G} instead of M just for clarity, and avoid confusion.

the remaining rows; i.e., a row $j[p]$ has only zeros where $p \in [m] - \{1, 2, \ldots, k\}$.

Also, we use a primitive function named $row_i(A)$ to compute the following operation over the i-th row of a matrix A: if the i-th row is non-zero, return the non-zero j-th position, otherwise return zero, which means that the i-th row is zero.

Finally we use interchangeably the follows notation for express an entry of an matrix A, either $A(i, j)$ or $A_{i,j}$.

Pre-condition: matrix M_{A_G} has at most one 1 per line.

Algorithm 2: Computing P

input : Max-Matching Matrix M_{A_G}

output : Permutation Matrix P

```
1  m ← row(M_{A_G});                    /* Assign the number of rows of M_{A_G} */
2  P ← 0_m;                  /* Assign m × m zeros into each entry of P */
3  r ← 0, q ← 0;
4  i[1...m] ← 0, j[1...m] ← 0;
5  p ← 1;
6  while p ≤ m do
7      if row_p(M_{A_G}) ≠ 0 then
8          q ← q + 1;
9          i[q] ← p;
10     else
11         r ← r + 1;
12         j[r] ← p;                        /* store the full zero rows */
13     p ← p + 1
14 FillPermRows(P, q);              /* Put ones according to i and j */
```

Post-condition: Matrix P is such that when it's multiplied by M_{A_G} from the right side, each row of PM_{A_G} is such that either

- it consists entirely of zeros, or
- it has exactly one 1,

where all zero rows are in the bottom.

That is,

$$\mathbf{P} \cdot \mathbf{M_{A_G}} = \left[\begin{array}{ccc} \cdots & 1 & \cdots \\ 1 & \cdots & \cdots \\ \cdots & \ldots & 1 \\ \vdots & \vdots & \vdots \\ \hline & 0 & \end{array}\right] \tag{A.3.1.17}$$

Let us look how to place the 1's in correct position in P.

Permutation matrix P can be easily computed from i and j, we want the following permutation of rows:

$$\pi := \tag{A.3.1.18}$$

$$\begin{aligned} i[1] &\mapsto 1 \\ i[2] &\mapsto 2 \\ &\vdots \\ i[k] &\mapsto k \\ j[1] &\mapsto k+1 \\ j[2] &\mapsto k+2 \\ &\vdots \\ j[m-k] &\mapsto m \end{aligned}$$

So, P has 1's in positions:

$(1, i[1]), (2, i[2]), \ldots, (k, i[k]), (k+1, j[1]), (k+2, j[2]), \ldots, (m, j[m-k])$ which are put in place by `FillPermRows`, see Algorithm 3.

Algorithm 3: Placing ones in P

input : Permutation Matrix P, k the number of rows not full of zeros.

output : Filled-Updated Permutation Matrix P.

```
1 m ← row(P);
2 p ← 1;
3 while p ≤ m do
4     if p ≤ k then
5         P_{p,i[p]} ← 1;
6     else
7         P_{p,j[p−k]} ← 1;
8     p ← p + 1;
```

Observe that k is the local variable of Algorithm 3 associated to the local variable q from the Algorithm 2, and of course, corresponding to the same k value of π-permutation (see A.3.1.18).

PROOF: **Correctness of the Algorithm 2, Computing P.** The proof is divided into two parts, the first correspond to the correctness—partial correctness and termination—of the loop of the Algorithm 2. The second part refers to the correctness of the function `FillPermRows`—Algorithm 3.

Therefore we start with the first part of the proof by proposing the following loop invariant,

$$(m \geq q + r) \wedge (p \geq 1). \tag{A.3.1.19}$$

where, m is the number of rows of M_{A_G}, q and r the number of nonzero rows and zero rows of M_{A_G} respectively, and p is just an index of which row the Algorithm 2 is working.

We show that (A.3.1.19) holds after each iteration of the loop.

Basis case: (i.e., zero iterations of the loop) we are before line 6 (Algorithm 2): $p = 1$, $r = 0$, $q = 0$, and $m = row(M_{A_G})$, this imply that $p \geq 1$ and $m \geq q + r$ the last inequality holds because our implicit assumption, i.e., $row(M_{A_G}) \geq 1$, is that we are working with matrices of dimension $m \times n$ were $m \geq 1 \wedge n \geq 1$, then (A.3.1.19) holds.

Induction step: suppose $m \geq q + r$ and we go once more through the loop, and let p', q' and r' be the new values of p, q, r, respectively. Since we executed the while loop one more time it follows that $p' \leq m$ and $p' > p \geq 1$, finally by pre-condition we have that either $q' > q$ or $r' > r$ which by induction hypothesis we have that $m \geq q' + r'$, and so still satisfy the loop invariant.

Now we use the loop invariant to show that, if Algorithm 2 terminates and the

pre-condition holds, then the post-condition holds. So, the loop terminates when is true that $\neg(p \leq m)$, i.e., is true $m < p$. On the other hand (A.3.1.19) holds on each iteration of the loop, in particular in the last one. So, at this point of the execution we have two cases,

Case 1: $p > m$ and $m > q+r$, but this last inequality is not true because by pre-condition we have a matrix with a fixed number of row such that:

- either all full of zeros then $r = m \wedge q = 0$ or all full of rows with exactly 1 one per row then $q = m \wedge r = 0$, or
- there are q rows with exactly a 1 per row plus r rows full of zeros, such that $q + r \leq m$.

Case 2: $p > m$ and $m = q + r$ the inequality comes from our supposition of termination and the equality comes by pre-condition, so we get our loop post-condition.

Until here we have partial correctness, so the next step is to show that the loop actually terminate, so, we use a non-negative monotone decreasing sequence related to the loop, denoted by v and it is defined by:

$$v_p := (m - p) + 1 \tag{A.3.1.20}$$

By loop pre-condition we know that $p > 0$, and also we know that m is integer fixed number such that $m \geq 1$. The sequence $v_1, v_2, v_3, \dots$ is a decreasing sequence of positive integers because $p \leq m$, so by Least Number Principle[17] (*LNP*), it is finite,

[17] The Least Number Principle says that every non-empty subset of the natural numbers must have a least element. A consequence of *LNP* is that every non-negative sequence of integers must terminate, i.e., if $V = \{v_1, v_2, v_3, \dots\} \subseteq \mathbb{N}$ where $v_i > v_{i+1}$ for all i, then V is a finite subset of $\mathbb{N}$.

and so the loop terminates.

At this point we proved the partial correctness and termination of the loop inside Algorithm 2.

We are going to use prime notation to define the local variables of `FillPermRows`. Note that inside of `FillPermRows` we have that the number of rows non-zero on M_{A_G} is denoted by $k' = q$, and the remaining rows—the zero rows—is denoted by $r' = m - q'$, and p' is just an index of which row the Algorithm 3 is working.

Also, we have that the invariant $(m \geq k' + r') \wedge (p' \geq 1)$ and using identically the same argument that before—see part one of this proof—we get that the loop invariant holds after each iteration of the loop.

Now we use the loop invariant to show that if `FillPermRows` terminates and its pre-condition holds, then `FillPermRows` post-condition holds. So, from the loop pre-condition we get that $p' \geq 1$ and $m \geq 1$ and when the loop terminates the is true that $p' > m$, and loop invariant holds after each iteration, and in particular the last iteration. Then we get that our loop post-condition is trivially true because of the true of $\neg(p' \leq m)$.

But, we will prove that the `FillPermRows` post-condition holds. So, at this point of the execution we have that the loop invariant holds after each execution of the loop, and from the `FillPermRows` pre-condition we know that $0 \leq k' \leq m$ then we have the following situation:

- if $k' = 0$ we do not have any line with a one on M_{A_G} and therefore $r' = m$ because $q' = 0$ then $p = 1 > k' = 0$, hence P has exactly a 1 on each p'-position defined on line 7 of the Algorithm 3.

- if $k' \neq 0$ we have two sub-cases, either $p' \leq k'$ in which case we have exactly k'-positions (i.e., k'-rows, line 5 of the Algorithm 3) with exactly[18] a 1, or $p' > k'$, then we have $(p' - k')$-positions ($(p' - k')$-rows, line 7 of the Algorithm 3) again with exactly a 1. In any sub cases we have exactly $k' + (p' - k') = q' + r' = m$ rows with exactly a 1 per each.

Then in every case we get the `FillPermRows` post-condition.

To show termination we use again that the sequence given by $v_1, v_2, v_3, \ldots$ is a non-negative decreasing sequence of integers because $p' \leq m$ and by *LNP* it is finite. □

Note that, in fact, the post-condition of Algorithm 3 is place 1's in P, transforming the initial full of zeros P into the correct permutation matrix, correct in sense that the equation $P \cdot M_{A_G}$ is of the form A.3.1.17. So, the post-condition of `FillPermRows` is in fact the post-condition of Algorithm 2.

Implicitly, we assume an order defined over the set of vertices V of a graph G, and the set of rows of the matrix representation of G, that is, we assume that the vertices are numbered $1, 2, \ldots, |V|$ in some arbitrary manner, then the matrix representation of G consists of a 0-1 matrix of size $|V| \times |V|$. Hence, we use this order to express that one row is less, in the list of rows (vertices), than another one. Therefore, every time we permute two rows r_1, r_2 with $r_1 < r_2$, we obtain that for all row r such that $r < r_1$, the row r_2—after be permuted— satisfy $r < r_2$, that is, all 1's that are in rows less than r_1 are going to continue been less than the 1 corresponding to the row r_2 permuted with row r_1. We never permute the row r_1 with some row less than it. Note that the row r_2 correspond to the first row which is not full of zero, in that way, all

[18]Note that if we are in this case, we know by HK-Algorithm that the equality it is satisfied, that is, we know that $p' = k'$.

the zero rows of M_{A_G} remain at the same place but in the permuted matrix PM_{A_G}, are stored at the bottom of it. This is showing that the whole permutation process is order preserving.

A.3.1.2 Computing Q from PM_{A_G}

Again, before continue with the Algorithm to Compute Q, let us mention how it works: on[19] input $\langle P \cdot M_{A_G} \rangle$ the algorithm goes trough each column of $P \cdot M_{A_G}$ storing in two different vectors i and j the nonzero columns and the zero columns, respectively. At the end, when both vectors are filled in line 14, the function `FillPermCols` is called in order to put the 1's on Q according to i and j. (See τ-permutation below)

We will use the same notation that we used in the previous section, plus the follow notation, let n be the number of columns of A, i.e., $c(A)$, sometime we use $col(A)$ to be clear in the context, and let $i[1], i[2], \ldots, i[k]$ be the columns of $P \cdot M_{A_G}$ that have at least one 1 in them, such that, $i[1] < i[2] < \cdots < i[k]$ (according to the columns). Let $j[k+1], j[k+2], \ldots, j[n-k]$ be the remaining columns; i.e., a row $j[p]$ has only zeros where $p \in [n] - \{1, 2, \ldots, k\}$.

Also, we use a primitive function named $row_i(A)$ to compute the following operation over the i-th row of a matrix A: if the i-th row is non-zero, return the non-zero j-th position, otherwise return zero, meaning that the i-th row is full of zero, and analogously, we define $col_j(A)$.

Finally we use interchangeably the follows notation for express an entry of an matrix A, either $A(i, j)$ or $A_{i,j}$

Recall that we will pad with zeros the matrix representation of the maximum matching in such a way that we our matrices are going to be square, hence it is not

[19] We are using this notation, i.e., $P \cdot M_{A_G}$ instead of $P \cdot M$ just for clarity, and avoid confusion.

necessary to make the distinction between number of rows and columns, therefore we shall consider interchangeable m and n.

We are going to compute the permutation matrix Q by using the previous computation of P times M_{A_G} like input of the following algorithm:

Pre-condition: Each row of PM_{A_G} is such that has q rows with exactly one 1 and the all zero rows are in the bottom side.

Algorithm 4: Computing Q from PM_{A_G}

input : P-Permuted Max-Matching Matrix, i.e., PM_{A_G}

output: Permutation Matrix Q

```
1  m ← row(PM_{A_G});          /* Assign the number of rows of PM_{A_G} */
2  Q ← 0_n;                    /* Assign n × n zeros into each entry of Q */
3  t ← 0, s ← 0;
4  i[1...n] ← 0, j[1...n] ← 0;
5  p ← 1;
6  while p ≤ m do
7      if row_p(PM_{A_G}) ≠ 0 then
8          s ← s + 1;
9          i[s] ← row_p(PM_{A_G});   /* assign the nonzero column of row p
           */
10     if col_p(PM_{A_G}) == 0 then
11         t ← t + 1;
12         j[t] ← p;                 /* column p full of zeros */
13     p ← p + 1;
14 FillPermCols(Q, s);             /* Place 1s according to i and j */
```

Post-condition: Q is a matrix that when we multiply it by PM_{A_G} from the left side, each row of $PM_{A_G}Q$ is form, such that, the $k \times k$ upper left square has in its diagonal all the 1's corresponding to the maximum matching.

Such type of matrix multiplied to PM_{A_G} by the right yields:

$$\mathbf{PM_{A_G}} \cdot \mathbf{Q} = \left[\begin{array}{cc|c} \cdots & 1 & \\ 1 & \cdots & 0 \\ \cdots & \ldots & \\ \vdots & \vdots & \\ \hline & 0 & \end{array}\right] \tag{A.3.1.21}$$

Let see how to place the 1's in the correct position in Q.

Let Q be the matrix associated to the permutation:

$$\tau := \tag{A.3.1.22}$$

$$\begin{array}{rcl} i[1] & \mapsto & 1 \\ i[2] & \mapsto & 2 \\ & \vdots & \\ i[k] & \mapsto & k \\ j[1] & \mapsto & k+1 \\ j[2] & \mapsto & k+2 \\ & \vdots & \\ j[n-k] & \mapsto & n \end{array}$$

So, Q has 1's in positions:

$(i[1],1),(i[2],2),\ldots,(i[k],k),(j[1],k+1),(j[2],k+2),\ldots,(j[n-k],n)$ which are put in place by the following function `FillPermCols`, see Algorithm 5:

Algorithm 5: Placing ones in Q

input : Permutation Matrix Q, k the number of columns not full of zeros

output : Filled-Updated Permutation Matrix Q

```
1 n ← col(Q);
2 p ← 1;
3 while p ≤ n do
4     if p ≤ k then
5         Q_{i[p],p} ← 1;
6     else
7         Q_{j[p−k],p} ← 1;
8     p ← p + 1;
```

Observe that k is the local variable of Algorithm 5 associated to the local variable q from the Algorithm 4, and of course, corresponding to the same k value of τ-permutation (see A.3.1.22).

Note that there are two situation that "appear" to be not contemplated in the Algorithm 4, which are the case of either $row_p(PM_{A_G}) == 0$, or $col_p(PM_{A_G}) \neq 0$. So, consider the following two cases:

- Case 1: If $row_p(PM_{A_G} == 0$ then $col_p(PM_{A_G}) == 0$. Suppose by contradiction, that $col_p(PM_{A_G}) \neq 0$, then $\exists i \leq k$ such that the entry $(PM_{A_G})(i,p) \neq 0$ but this means that $row_i(PM_{A_G}) \neq 0$, but choosing i to be p we arrive to a contradiction.

- Case 2: If $col_p(PM_{A_G}) \neq 0$ then $row_p(PM_{A_G}) \neq 0$. Again, suppose by contradiction, that $row_p(PM_{A_G}) == 0$, then $\exists j \leq k$ such that the entry $(PM_{A_G})(p, j) = 0$, but this means that $col_j(PM_{A_G}) == 0$, but choosing j to be p we arrive to a contradiction.

Therefore, by Case 1 we catch the case in line 10 of the Algorithm 4, and by Case 2 we catch the case in line 7 of the Algorithm 4. So, at the end we are contemplating all cases.

PROOF: **Correctness of the Algorithm 4, Computing Q.** The correctness of Algorithm 4 (Computing Q) essentially is similar to the proof of correctness of Algorithm 2 (Computing P), divided into two parts:

The first part we use the same—but working over columns instead of rows—loop invariant relabelling q by s and r by t in such a way that $s + t \leq m$, therefore, the proof of what the invariant holds after each iteration of the loop is the same.

Now, we use the loop invariant $(m \geq s + t) \wedge p \geq 1$ to show that, if Algorithm 4 terminates and the pre-condition holds, then the post-condition holds. So, when the loops terminates, is true that $m < p$, and the loop invariant holds on each iteration of the loop, in particular in the last one. Therefore, at this point of the execution we have the following cases:

Case 1: $p > m$ and $m > s + t$, but this is false because the last inequality always is false by the following reasons:

- either all columns full of zeros then $t = m$ and $s = 0$, or all columns with exactly 1 one per column, then $s = m$ and $t = 0$, or
- there are s rows exactly with a one per row, plus $t = n - s$ columns full of

zeros such that $t + s \leq n = m$, where the last equality comes because we work—after padding (if necessary)—on square matrices.

Case 2: $p > m$ and $m = s + t$ the first term comes from our assumption of termination and the second term comes from the pre-condition which states that we have a P-permuted matrix, i.e., PM_{A_G}, such that s rows are with exactly a 1 per row and $m - s$ rows full of zeros which is equivalent to say that we have t columns full of zeros, see line 12 of Algorithm 4.

So, finally we get our post-condition.

Therefore, until here we proved partial correctness, so the next step is to show that the loop actually terminate, but this is exactly with the same non-negative decreasing sequence used in Algorithm 2, hence, we conclude with proof of correctness of Algorithm 4.

Now, we continue with the second part of the proof which refers to `FillPermCols` subroutine—Algorithm 5. This part is similar to the proof of `FillPermRows` (Algorithm 3) with the following difference, we use the coordinates of the vectors—which represent columns—like rows[20] in order to produce the correct permutation, where correct means permuted according to (A.3.1.22), i.e., τ-permutation.

So, we get a permutation matrix Q such that when we multiply to PM_{A_G} by right

[20]Recall that when we assign values to vectors i and j in Algorithm 4, we are storing the nonzero row coordinate of column p, on the other hand, in the case that the column is zero, we store the column coordinate itself.

side we obtain:

$$\mathbf{PM_{A_G}} \cdot \mathbf{Q} = \left[\begin{array}{cc|c} \cdots & 1 & \\ 1 & \cdots & 0 \\ \cdots & \ldots & \\ \vdots & \vdots & \\ \hline & 0 & \end{array}\right] \tag{A.3.1.23}$$

□

Finally, considering the structure—given by π and τ—of how were yield permutation matrices P and Q, we proceed to multiply PA_GQ obtaining

$$PA_GQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0 \end{array}\right] = \left[\begin{array}{cc|c} 1 & & \\ & 1 & \\ & \ddots & A_1 \\ & & 1 \\ \hline & A_2 & O \end{array}\right]$$

Here we denoted by red those 1s that represent that 1s which correspond to the maximum matching obtained by HK-Algorithm (Chapter 2 Section 2.3).

Note that an important observation is that *it is not necessary to know the, maybe, complex structure of the matrix A_G in the whole process of yield the matrix form A.16, that at the end we get the expected result, that is, the 1s from the maximum matching are in the main diagonal. But more important, from those 1s we extract, in a constructively way, a minimum vertex cover.*

This is an example of our combinatorial approach to a graph theoretical problem like is Minimum Vertex Cover, because linear algebraic properties of one of the matrices

associated with a graph, for instance, M_{A_G}, is used to get useful combinatorial information about the graph G.

A.3.1.3 Computing $\vec{O}$ from PA_GQ

We begin this section recalling that the structure of the input $\langle PA_GQ \rangle$ to the Algorithm to Compute $\vec{O}$, that is,

$$PA_GQ = \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0 \end{array}\right] \tag{A.3.1.24}$$

where we let P, Q be two permutation matrices such that P is $m \times m$ and Q is $n \times n$, and the dimension of the sub-matrix are $E^{k\times k}$, $A_1^{k\times(n-k)}$, $A_2^{(m-k)\times k}$ and $O^{(m-k)\times(n-k)}$, where $k = |M_{A_G}|$ denote the size of the maximum matching.

Before continue with the presentation of the Algorithm to Compute $\vec{O}$ we mention some remarks, fist of all, we are going to consider in the analysis and design of the algorithm to compute $\vec{O}$ just[21] **Step 2b** of our main Algorithm (Chapter 3 Section 3.3), and the reason is because the other steps are computational straightforward, and secondly, we will use—in this section and the next one—interchangeability $\vec{O} := \{o_1, o_2, \ldots, o_k\}$ and $\vec{O} := (o[1], o[2], \ldots, o[k])$ depending of the context, that is, sometimes one notation is more clear than the other one. Also, we use a primitive function named $row_i(A)$ to perform the following operation over the i-th row of a matrix A: if the i-th row is non-zero, return the non-zero j-th position, otherwise return zero, meaning that the i-th row is full of zero, and analogously, we define $col_j(A)$.

So, the algorithm is computational very simple, it works as follow: going through the diagonal of E, for each diagonal element, i.e., e_{ii} for $i \leq k$, we ask if, in the ith

[21] Recall that Step 2b means that we are in case $A_1 \neq 0 \wedge A_2 \neq 0$.

row of E, the ith row of A_1 is $\neq 0$? If yes, the orientation vector of the element e_{ii} will be horizontal, otherwise, we ask if, in the ith column E, the ith column of A_2 is $\neq 0$? If yes, the orientation vector of e_{ii} will be vertical, otherwise e_{ii} correspond to a a black 1 an we continue with the next diagonal element.

Basically, what the Algorithm 6 does is to define which entry among the k-th main diagonal ones, is black or green according to the exploration of A_1, and eventually, if the line of A_1 was full of zeros, explore A_2. So, the algorithm is like follows:

Pre-condition: the structure of PA_GQ is like in A.3.1.24.

Algorithm 6: Computing Orientation vector $\vec{O}$

input : Matrix PA_GQ

output : Orientation Vector $\vec{O}$

```
1  k ← row(E);                                   /* E^{k×k} submatrix of PA_GQ */
2  Ō[1...k] ← 0;                  /* Initialize to 0 ks position of Ō */
3  p ← 1;          /* p range over the first k diagonal elements.  */
4  while p ≤ k do
5      if row_p(A_1) ≠ 0 then
6          Ō[p] ← 1;                                  /* Horizontal-Green */
7      else
8          if col_p(A_2) ≠ 0 then
9              Ō[p] ← 0;                               /* Vertical-Green */
10         else
11             Ō[p] ← 2;                                        /* Balck */
12     p ← p + 1;
```

Post-condition: The orientation vector $\vec{O} = \{o_1, o_2, \ldots, o_k\}$ is

$$o_p = \begin{cases} 0 & \text{if } \exists\ q \leq k \text{ such that } col_q(A_2) \neq 0 \\ 1 & \text{if } \exists\ r \leq k \text{ such that } row_r(A_1) \neq 0 \\ 2 & \text{if } \exists\ i \leq k \text{ such that } row_i(A_1) = col_i(A_2) = 0 \end{cases} \tag{A.3.1.25}$$

that is, we obtain a vector which will express in which position of the main diagonal are black 1's or green 1's. Recall that we use indistinguishably o_p and $\vec{O}[p]$ for the vector entries.

Now, having our matrix PA_GQ such that we know by the orientation vector $\vec{O}$ which diagonal element of E are green 1s or black 1s, our next step is to proceed to split it into two groups, all the green 1s on the lower right corner of E and the black 1s on the top left corner of E (see Chapter 3 Figure 4.2). To do the splitting process we compute permutation μ, explained in the follow section.

A.3.1.4 Computing μ from PA_GQ and $\vec{O}$

In this section we are going to discuss essentially how the permutation μ works, and how we obtain our main goal, that is, the orientation vector $\vec{O}$ denoting the Minimum Vertex Cover related to a given bipartite graph G.

Remember that μ will be computed if we are in **Step 2b** of our algorithm (Chapter 3 Section 3.3), i.e., $(A_1 \neq 0 \wedge A_2 \neq 0)$, so, the first task to be computed is to group the diagonal elements on green 1s and black 1s. Hence, we need to explore each entry of $\vec{O}$ to determine if a particular diagonal entry is green or not, in an specific instance. Recall that the values of each entry of $\vec{O}$ are 0 to denote green vertical line, 1 to denote green horizontal line, and 2 to denote black 1s.

Before continue with the algorithm to compute the process of split green 1s and black 1s, let us make some remarks about permutation μ:

- μ is a general permutation, no necessary and more probably, a non-order preserving permutation; but recall that Corollary 3.2.1:

Corollary *A general permutation of the 1s in PA_GQ, where we place them on the diagonal of a contiguous block as in Lemma 3.2.1, but we also* reorder *them according to some permutation μ still yields a corresponding covering where we account for μ as follows:*

$$C^{\mu(\vec{o}),\pi(\vec{i}),\tau(\vec{j})}_{R_\mu P_\pi A_G Q_\tau R_\mu} = \{l^{o_{\mu(1)}}_{(\mu(\pi(i_1)),\mu(\tau(j_1)))}, l^{o_{\mu(2)}}_{(\mu(\pi(i_2)),\mu(\tau(j_2)))}, \ldots, l^{o_{\mu(k)}}_{(\mu(\pi(i_k)),\mu(\tau(j_k)))}\}.$$

in which the point of these rather technical corollary is to show that we can preserve coverings under permutations; that is, we can permute PA_GQ at will in our algorithm, and recover the lines easily.

- another technical issue about μ correspond to the following fact: recall that we use to describe permutations, as bijections, π and τ, and as permutation matrices, P_π and Q_τ, that is, P_π and Q_τ permute the rows and columns, respectively, of PA_GQ, so, when we apply μ to PA_GQ, what we really do is to multiply[22] it from left by a permutation matrix R_μ of size $m \times m$ and from right by a permutation matrix R'_μ of size $n \times n$, resulting the product $R_\mu P_\pi A_G Q_\tau R'_\mu$. Also, in order to simplify and make more readable the computations, we shall adopt that $|R_\mu| = |R'_\mu|$. Because otherwise we have to pad with zeros the matrices in the following way, if our E submatrix is of size $k \times k$ and our PA_GQ matrix is

[22]We use cycle decomposition of the matrices to compute the multiplication on that linear representations. (For details see Section 3.4 and Appendix A.4)

of size $m \times n$ we will need pad with zeros R_μ with $(m-k)$ columns, and pad with zeros R'_μ with $(n-k)$ rows.

The main point here is that μ is a transposition of the k first diagonal elements of PA_GQ, this means the transposition of the entire row or column, and in order to swap the lines—rows or columns—we need to compute the permutation matrices R_μ and R'_μ, and put ones inside of both matrices according to the lines to be swapped. This involve the same process explained in detail in Section A.3.1 Subsection Computing P from M_{A_G} (A.3.1.1), and Subsection Computing Q from PM_{A_G} (A.3.1.2).

Therefore, the process of compute μ it is computational very simple and "similar" to the process that we detail in Sections A.3.1 subsection Computing P from M_{A_G} (A.3.1.1), and subsection Computing Q from PM_{A_G} (A.3.1.2), similar not only because instead of π and τ we compute another permutation μ, but also more simple because μ is a transposition function, and basically what we do is to put 1s on both R's matrices according to the entries of the orientation vector $\vec{O}$, to yield the transpositions.

In order to make the computation of μ, and in particular the computation of the orientation vector $\vec{O}$ more readable and simple, instead of yield the permutation matrices R_μ and R'_μ, we are going to use a function called `Swap` which interchange two rows, or two columns, of a particular matrix A, that is, `Swap`$(col_i(A), col_j(A))$ or `Swap`$(row_i(A), row_j(A))$. Note that this change does not produce any considerable change in the computational complexity associate to the algorithm that compute the orientation vector $\vec{O}$.

Then, we proceed with the Algorithm 7 to compute the process of split green 1s and black 1s in the main diagonal of PA_GQ, and compute the orientation vector $\vec{O}$

keeping track of every change on $\vec{O}$ provoked by what the splitting process does over the main diagonal elements.

In short, the Algorithm 7 group green 1s and black 1s of the main diagonal of E, and update the orientation vector $\vec{O}$ associate to them.

Pre-condition: the structure of PA_GQ is like in A.3.1.24, and the orientation

vector is initialized like A.3.1.25,i.e., with 0s,1s, and 2s.

Algorithm 7: Grouping ones on E, and updating $\vec{O}$

input : PA_GQ Matrix and orientation vector $\vec{O}$

output : PA_GQ Matrix with grouped green and black 1s, and updated version of $\vec{O}$

```
1  k ← row(E);                                   /* E^{k×k} submatrix of PA_GQ */
2  flag ← 1;
3  k_1 ← 0;                                      /* number of black 1s */
4  k_2 ← 0;                                      /* number of green 1s */
5  p ← 1;                                        /* p ranges over diagonal of E */
6  while p ≤ k do
7      if Ō[p] == 2 then
8          k_1 ← k_1 + 1;
9      else
10         k_2 ← k_2 + 1;
11         k̂ ← k;  /* aux var.  for ranges backwards over diagonal */
12         while flag ∧ (k̂ > p) do
13             if Ō[k̂] == 2 then
14                 UpdateO(Ō[k̂], Ō[p]);
15                 flag ← 0;
16             k̂ ← k̂ − 1;  /* p and k̂ green then goes backward looking
                           for a black 1.  */
17     p ← p + 1;
18     flag ← 1;
```

Post-condition: the internal structure of $\vec{O}$ corresponds to the orientation associated to the group of black 1s of the upper-left corner of E, and the group of green 1s to the lower-right corner E. That is, the post-condition is the updated version of $\vec{O}$ and the matrix PA_GQ has two groups—black and green—in the main diagonal of E.

Note that when we find a green one, i.e., the p variable, the *flag* variable is used to indicate which will be the position to go backwards looking for a black one to compute the transposition. When we swap lines, the *flag* variable is useful because, after swapping we need to move on the next value of p for which we do not know a priori if it is black or green, so, *flag* allow us to exit of the inner while loop in order to analyze what is in the p position—green or black—in the outer while loop. Otherwise, if we do not use the *flag* we need to add more conditions in the structure of the algorithm, in order to not count again the green one that was swapped and put in a further position in the main diagonal.

Before continue with details of function `Update`—Algorithm 8, we proof the correctness—partial correctness and termination—of Algorithm 7.

PROOF:**Correctness of Algorithm 7, Grouping ones on E, and updating $\vec{O}$.** The proof is going to be divided into three parts, the first one will be the correctness—partial correctness and termination—of the main loop, the second one the correctness of the inner loop, and the third one correspond to the correctness of the `Update` function—Algorithm 8.

Therefore, we start with the first part of the proof by proposing the following loop invariant for the outer while loop,

$$(k \geq k_1 + k_2) \wedge (p \geq 1). \tag{A.3.1.26}$$

where, k is the number of rows of submatrix E, k_1 and k_2 denote the number of black 1s and green 1s, respectively, and p is just an index of which row of E the Algorithm 7 is working.

We show that (A.3.1.26) holds after each iteration of the outer while loop.

Basis Case: (i.e., zero iterations of the outer loop) we are before line 6—Algorithm 7: $p = 1, k_1 = 0, k_2 = 0$, and $k = row(E)$, this imply that $p \geq 1$ and $k = row(E) \geq 1 \geq k_1 + k_2$, then A.3.1.26 holds.

Inductive step: suppose $k \geq k_1 + k_2$ and go once more through the outer loop, and let p', k_1', and k_2' be the new values of p, k_1, k_2, respectively. Since we execute the outer while loop once more time it follows that $p' \leq k$ and $p' > p \geq 1$, finally, by pre-condition[23] we have that either $k_1' > k_1$ or $k_2' > k_2$ which by induction hypothesis we have that $k \geq k_1' + k_2'$, and so still satisfy the loop invariant.

Now, we use the outer loop invariant to show that, if Algorithm 7 terminates and the pre-condition holds, then the post-condition holds. So, the loop terminates when is true that $\neg(p \leq k)$, i.e., is true $k < p$. On the other hand, A.3.1.26 holds on each iteration of the outer loop, in particular in the last one. Hence, at this point of the execution we have two cases,

Case 1: $p > k$ and $k > k_1 + k_2$, but this last inequality is false, because, otherwise contradict the maximality of the maximum matching of G, because by correctness of HK-Algorithm we know that the maximum size of $|M_{A_G}| = k$, but this k's elements correspond to the number of lines—rows and columns—of the submatrix E of size $k \times k$.

Case 2: $p > k$ and $k = k_1 + k_2$, where the inequality comes form our assumption

[23]Recall that we are in case $A_1 \neq 0 \wedge A_2 \neq 0$.

of termination and the equality comes by pre-condition, so we get our post-condition.

Until here we have partial correctness, so the next step is to show that the outer loop actually terminate, so, we use a non-negative monotone decreasing sequence related to the outer loop, defined:

$$v_p := (k - p) + 1 \tag{A.3.1.27}$$

By outer loop pre-condition we know that $p > 0$, and also we know that k is an fixed integer number such that $k \geq 1$. The sequence $v_1, v_2, v_3, \ldots$ is a decreasing sequence of positive integers because $p \leq k$, so by *LNP* is finite, and so the outer loop terminates.

At this point we proved **partial correctness and termination of the outer loop** inside Algorihtm 7.

Now, we are going to proceed with the correctness—partial correctness and termination—of the inner while loop, so, we propose the follow inner loop invariant,

$$p \leq \hat{k} \leq k \tag{A.3.1.28}$$

where $\hat{k}$ is a variable which going to be used to range between p—the actual green one the was find it—and $k = row(E)$; k denotes the possible—because must be a black 1—position to swap the 1 on p.

We show that A.3.1.28 holds after each iteration of the inner while loop.

Basis Case: (i.e., zero iterations of the inner while loop) we are before line 12—Algorithm 7, $\hat{k} = k$ by line 11, and $p \leq k$ because we are in a particular execution of the outer while loop, therefore, our inner loop invariant holds. Note that we are increasing k_2 by one—line 10 such that, if k_2' denote the actual execution, it is true that $k \geq k_1 + k_2' \geq \hat{k}$.

Inductive step: suppose $p \leq \hat{k} \leq k$ and go once more through the inner loop, and let $p', \hat{k}', k_1', k_2'$, and $flag'$ be the new values of $p, \hat{k}, k_1$, and k_2, respectively. Since we execute the inner while loop once more time if follows that $flag = 1$, and $\hat{k}' > p'$, note that p is fixed during every execution of the inner while loop, so, $p' = p$, and also note that $k = \hat{k} > \hat{k}'$ by lines 11, and 16, respectively, finally, by pre-condition and by case—we are in the case p is a green 1—we have that $k_1' = k_1$ and $k_2' = k_2$, putting all together and by induction hypothesis we have that $k \geq k_1' + k_2' \geq \hat{k} \geq p$, and our inner loop invariant is still satisfy.

Now, we use the inner loop invariant (A.3.1.28) to show that, if the inner while loop terminates, and the inner while pre-condition holds, the inner while post-condition holds. So, if the inner loop terminates we have either $flag == 0$, or $\hat{k} \leq p$. On the other hand, inner loop invariant, i.e., $p \leq \hat{k} \leq k$ holds on each iteration of the inner loop, in particular in the last one. Hence, at this point of the execution we have the following cases:

- $flag == 0$ and $p < \hat{k} \leq k$, where the first assertion holds by our assumption of termination and the the second one by invariant of inner while loop. So, we obtain that the inner while post-condition holds.
- $flag == 0$ and $p = \hat{k} \leq k$, where both assertions hold by our assumption of termination. Hence, our inner while loop post-condition holds.

- $(flag == 1 \wedge \hat{k} \leq p)$ and $p \leq \hat{k} \leq k$, where the only case that make both assertion true are the equalities, so, the first clause comes from pre-condition and the assumption of termination, and the second one comes from the inner loop invariant. Therefore, we obtain our inner post-condition.

So, putting altogether, the inner while loop post-condition

$$(flag == 0 \wedge p \leq \hat{k} \leq k) \vee (flag == 1 \wedge \hat{k} \leq p \leq k)$$

holds.

Until here we proved partial correctness, so the next step is to show that the inner loop actually terminate, so, we use a non-negative monotone decreasing sequence related to the inner while loop, defined:

$$v_p := \hat{k}_p \tag{A.3.1.29}$$

By inner loop pre-condition we know that $\hat{k} > p > 0$, and also we know that k is an fixed integer number such that $k \geq \hat{k}$. The sequence $v_1, v_2, v_3, \ldots$ is a decreasing sequence of positive integers because of line 16 of Algorithm 7, so by $\hat{k} \leq k$ and *LNP* it is finite, and hence, the inner while loop terminates.

At this point we proved **partial correctness and termination of the inner loop** inside Algorihtm 7.

Now we proceed to prove the correctness of the function `Update0`—Algorithm 8. Basically, the correctness of `Update0` is the correctness of an *If*-statement.

So, by the previous state of the *If*-statement, that is, just after line 13 of Algorithm 7—we have that $\vec{O}[\hat{k}] == 2$, that is, $(\hat{k}, \hat{k})$ is a black 1, otherwise we can not be

calling `Update0`, and (p, p) entry correspond to a green 1, which, clearly imply the *If* pre-condition of `Update0`, which is $\vec{O}[p] == 1 \vee \vec{O}[p] == 0$.

Also, after any of the two cases, we swap p and $\hat{k}$ rows of PA_GQ, or p and $\hat{k}$ columns of PA_GQ, respectively, and obtain an updated version of PA_GQ matrix, which make the *If* post-condition, i.e.,

$$\vec{O}[p] == 1 \wedge \texttt{swap}(row_p(PA_GQ), row_{\hat{k}}(PA_GQ)) \vee$$
$$\vec{O}[p] == o \wedge \texttt{swap}(col_p(PA_GQ), col_{\hat{k}}(PA_GQ))$$

holds.

Beside, on line 5 of `Update0`—Algorithm 8—we swap positions inside of the orientation vector $\vec{O}$ in order to obtain an updated of the vector $\vec{O}$ with respect to the updated matrix PA_GQ.

In line 6 we increase the number of black 1s because of the $(\hat{k}, \hat{k})$ black entry.

Finally, note that en line 7 of Algorithm 8, we decrease by 1 the number of green 1s, i.e., k_2, because when we find a black position we swap the green 1 on (p, p) and put it in a place that is going to be checked in next rounds, otherwise, we count again the same green 1 that we swapped.

Hence, we obtain the post-condition of `Update`, **finishing with the correctness of `Update0`—Algorithm 8**.

This finish **the correctness of the Algorithm 7**. □

Observe that we separate—in another algorithm—the updating process only for make more readable the whole process of grouping ones and updating the orientation vector.

Now, the `UpdateO` function not only updates the orientation vector $\vec{O}$ but also update the lines—rows and columns—in the square $k \times k$ quadrant of matrix PA_GQ, and note that this process update also A_1 and A_2, that is why we say that updates PA_GQ.

Algorithm 8: Updating PA_GQ and $\vec{O}$

input : $\vec{O}[\hat{k}]$ and $\vec{O}[p]$, i.e., position to be updated

output : Updated vector $\vec{O}$, and Updated matrix PA_GQ

```
1 if O[p] == 1 then
2 |   Swap(row_p(PA_GQ), row_k̂(PA_GQ));          /* Horizontal-Green */
3 else
4 |   Swap(col_p(PA_GQ), col_k̂(PA_GQ));          /* Vertical-Green */
5 Swap(O[k̂], O[p]);
6 k1 ← k1 + 1;
7 k2 ← k2 − 1;
8 Return;
```

Note that lines 2 and 4 of Algorithm 8, it updates the matrix PA_GQ and in line 5 of the same algorithm, we update the two entries of the orientation vector $\vec{O}$. Besides, is Algorithm 8 which make use of the distinction between horizontal and vertical line of the green 1s. Also, observe that p and $\hat{k}$ remain fixes during every execution of `UpdateO`.

In the Figure A.3 we show that basically, `Swap` function—in the context of bijections, this is our transposition μ—is a transposition of $i \mapsto i + l$ and $i + l \mapsto i$, the same to the coordinate j of an arbitrary entry (i, j) in PA_GQ. These are the basic steps in order to group black 1s and green 1s, that is, the transposition of each diagonal entry, to be more specific, we swap the whole line, not only the diagonal entry. Besides,

we compute the swapping corresponding to two entries of $\vec{O}$ accordingly with the swapping of PA_GQ in line 5 of Algorithm 8.

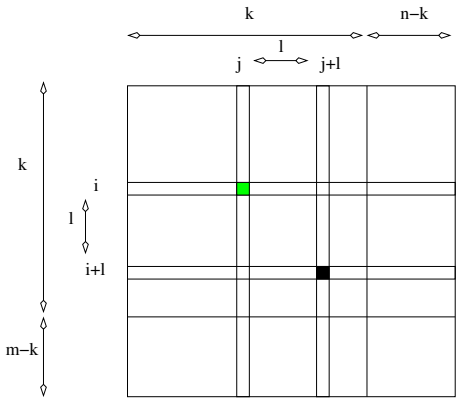


Figure A.3: Exchanging green 1s and black 1s in an arbitrary entry l. Swapping $(i, j) \mapsto (i + l, j + l)$ and the other way around.

Note that the Figure A.3 it is referencing to an arbitrary position l, so, to clarify the swapping and updating computation of Algorithm 7 and Algorithm 8, let us put graphically like Figure A.4:

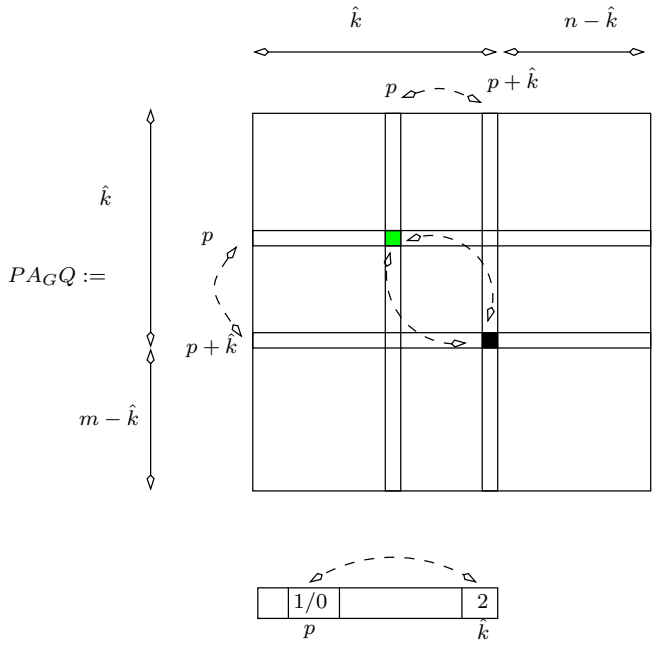


Figure A.4: Exchanging green 1s and black 1s. Swapping $(p, p) \mapsto (p + \hat{k}, p + \hat{k})$ and the other way around, also updating the entries of the orientation vector $\vec{O}[\hat{k}]$ and $\vec{O}[p]$.

Note that the arc dash line of inside of PA_GQ are denoting the swapping between the entries of the orientation vector $\vec{O}$. Also, see that the green entry in the vector $\vec{O}$ is a green horizontal line, i.e., $\vec{O}[p] := 1$ but could be a green vertical line, i.e., $\vec{O}[p] := 0$.

At this point of the analysis of our algorithm—see Chapter 3 Section 3.3—the next step is the **recursive call**. Before compute the recursive call, let us make some observations, the orientation vector $\vec{O}$ has in its entries the values according to the lines that goes over each green 1s that appear in the diagonal of lower right corner of E. Note that the orientations—horizontal or vertical—of each green one entry is represented graphically in Figure A.5 by red lines.

$$PA_GQ := \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0_{(m-k)\times(n-k)} \end{array}\right] =$$

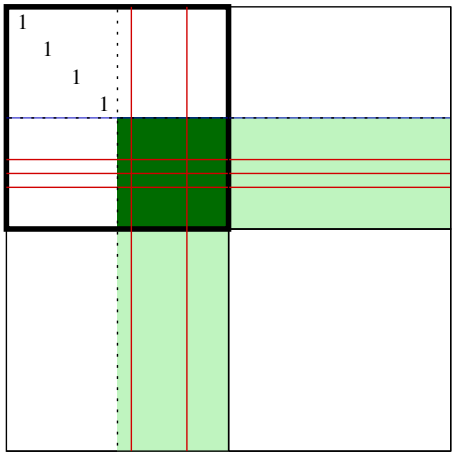


Figure A.5: Repeat Step 2 of our algorithm—Chapter 3 Section 3.3—with the upper-left quadrant, emphasized with a thicker border, with the "green square" zeroed out, as well as the entries under the red lines, arising from the cover of the "green square," zeroed out.

Here—in the recursive call—is when we make use of the updated version of the orientation vector $\vec{O}$, in order to define the red lines of Figure A.5 that pass over the green 1s, and zeroed out the lower right quadrant of E.

Recall that remain to compute the orientations of the lines covering the black 1s. We do so **recursively**, by repeating the procedure from **Step 2** of our algorithm—see Chapter 3 Section 3.3—with the matrix obtained from the $k \times k$ upper-left quadrant of PA_GQ, i.e., the upper-left quadrant of the matrix in Figure A.5, but with the following modifications:

the square enclosing the green 1s, represented in dark green, is zeroed out,

and we also place zeros wherever those lines crossed the rest of the quadrant. In Figure A.5, we place zeros in the upper-left quadrant wherever the horizontal red lines crosses.

We summarize—graphically speaking—the recursive call in Figure A.6. Let $k = k_1 + k_2$ be the size of the diagonal E, sucht that, we want to compute recursively over the submatrix $E^{k \times k}$, maintaining updated the permutation vector $\vec{O}$ according to the changes provoked inside of the Algorithm 7.

$$PA_GQ := \left[\begin{array}{c|c} E & A_1 \\ \hline A_2 & 0_{(m-k)\times(n-k)} \end{array}\right] =$$

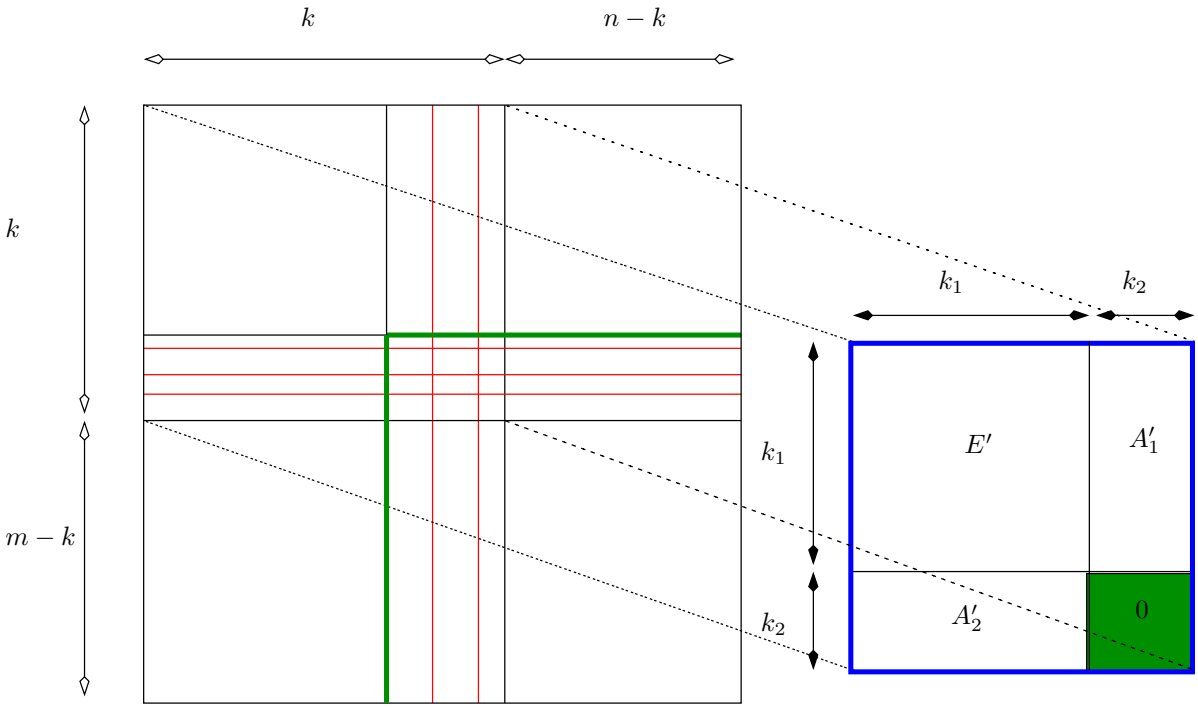


Figure A.6: Recursive Call Diagram. Where primes denote the next instance to be computed.

Observe, that after do the recursive call, the elements of the main diagonal—which at the beginning of this instance are all black 1s—if enter again in the case of **Step 2b**, that is, $(A_1 \neq 0 \wedge A_2 \neq 0)$, we know that some—for sure two—of the diagonal 1s which where black on the previous instance, are going to turn on green on the running recursive instance.

Finally, note that in order to know if an orientation line—if some one exists—is horizontal or vertical for a particular entry, we need *only* explore A_1 or A_2; this is because, in the step in which our algorithm—see Chapter 3 Section 3.3—is "checking" if an arbitrary diagonal entry is going to turn on green, say the entry e_{ii}, without loss of generality, let A_1 be the first to be checked, so, we range over the ith row of A_1 such that if it is not full of zeros, *choose horizontal AND turn on green*, and we do not need check A_2. Else, we range over the jth column of A_2 such that it is not full of zeros, *choose vertical AND turn on green*, otherwise goes to the next one. That is what we call "checking".

So, in this process, if we know that the previous black 1 turn on green 1, either horizontal or vertical, we do not need to range over the other orientation, that is the vertical or the horizontal, respectively.

Therefore we don't need more than $O(k(n-k))$ steps to set up the lines for the green 1s. For more details about the complexity of our algorithm see Chapter 3 Section 3.4.

A.3.2 Example

In this section we are going present an example in order to show how the permutations P, Q work, how the process of grouping green 1s and black 1s works, and finally,

how to yield the orientation vector $\vec{O}$ which correspond to the Minimum Vertex Cover of a given bipartite graph G.

The example is going to be divided into two parts, the first one correspond to the permutation matrices P and Q, and the second one, is going to be related with the grouping process—that refers to our permutation μ—and the orientation vector $\vec{O}$.

In each part we will use appropriate 0-1 matrices to make use of all the characteristics of our data structures. This is the reason why we use one set of 0-1 matrices in the first part, and another different in the second one.

So, our first part consist of computing the adequate P and Q permutation matrices. Therefore, consider the follow bipartite graph $G = (V = V_1 \cup V_2, E)$, where

$$V_1 = \{1, 2, 3, 4, 5, 6\} \quad , \quad V_2 = \{1', 2', 3', 4', 5', 6'\}$$

and

$$E = \{(1, 2'); (1, 6'); (2, 4'); (3, 3'); (4, 3'); (5, 1')\}$$

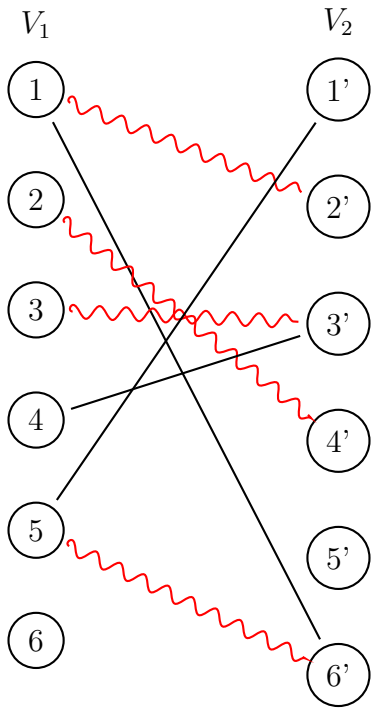
The adjacency matrix of G of size 6×6 is:

$$\mathbf{A_G} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Let M_{A_G} the output of $HK(A_G)$, for instance,

$$\mathbf{M_{A_G}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We denote those 1s that belong to the maximum matching with red color. So, the associated graph is:



where the snaked lines correspond to the maximum matching returned by Hopcroft-Karp algorithm on input $\langle A_G \rangle$.

In order to compute P, we compute the vectors $i[q]$ and $j[r]$ (see Algorithm 2) from M_{A_G}, where q and r denote the nonzero and full of zeros number of rows in M_{A_G},

respectively, that is, the entries of $i[q]$ represent the rows with a 1 , i.e., if $i[a] = b$ then the b-row of M_{A_G} has a red one on it, and the entries of vector $j[r]$ denote the rows of M_{A_G} full of zeros, so we have that rows $1, 2, 3$, and 5 of M_{A_G} are nonzero, hence, vectors i and j have the following values

$i[q]$			$j[r]$		
$i[1]$	$\leftarrow$	1	$j[1]$	$\leftarrow$	4
$i[2]$	$\leftarrow$	2	$j[2]$	$\leftarrow$	6
$i[3]$	$\leftarrow$	3			
$i[4]$	$\leftarrow$	5			

So, $q = 4$ representing the total number of nonzero rows, on the other hand, $r = 2$ represent the rows full of zeros, such that $r + q = row(M_{A_G})$. Therefore, in line 14 of the Algorithm 2 we call `FillPermRows`$(P, 4)$—see Algorithm 3—getting that P has 1's in position,

$$\{(1,1),(2,2),(3,3),(4,5),(5,4),(6,6)\}$$

In matrix form it is

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Then, by post-condition of Algorithm 2, and in order to obtain the form of equation A.3.1.17—all full of zero rows at the bottom of the matrix—we multiply the follow matrices

$$\mathbf{P}\cdot\mathbf{A_G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that we are using A_G instead of M_{A_G} like express equation A.3.1.17.

Besides, we have that

$$\mathbf{P}\cdot\mathbf{M_{A_G}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which correspond to the input of our next step, which is compute the permutation matrix Q, where we have $q = 4$ non-zeros rows.

Therefore, from Algorithm 4 we compute vectors $i[s]$ and $j[t]$, where s denote the nonzero number of rows in PM_{A_G}, and t denote the full of zeros columns in PM_{A_G}, so,

we have that columns $2, 3, 4$ and 6 are nonzero, and 1 and 5 are full of zeros. Hence, vectors i and j have the following values

$i[s]$	$j[t]$
$i[1] \leftarrow 2;$	$j[1] \leftarrow 1$
$i[2] \leftarrow 4;$	$j[2] \leftarrow 5$
$i[3] \leftarrow 3;$	
$i[4] \leftarrow 6;$	

Finally, in line 14 of the Algorithm 4 we call `FillPermCols`$(Q, 4)$ —see Algorithm 5—getting that Q has 1s in positions,

$$\{(2,1),(4,2),(3,3),(6,4),(1,5),(5,6)\}$$

In matrix form it is

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Then, by post-condition of Algorithm 4, and in order to obtain the form of equation A.3.1.23—all full of zero columns grouped to the right size—we multiply the

follow matrices but using A_G instead of M_{A_G},

$$(\mathbf{PA_G}) \cdot \mathbf{Q} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \left[\begin{array}{cccc|cc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Now, we continue with the second part of our example, in which we will to compute the grouping of green 1s and black 1s using PA_GQ, and we update the orientation vector $\vec{O}$.

Firts of all, we compute the orientation vector on input $\langle PA_GQ \rangle$, see Algorithm 6, obtaining the following $k = 4$ entries, $O[1] = 2$ because $row_1(A_1) == 0 \wedge col_1(A_2) == 0$, i.e., a black one, $O[2] = 2$ for identical reasons, $O[3] = 0$ because $col_3(A_2) \neq 0$, i.e., a vertical green one, and finally, $O[4] = 1$ because $row_4(A_2) \neq 0$. Therefore, the orientation vector is

$$\vec{O} := (O[1], O[2], O[3], O[4]) = (2, 2, 0, 1)$$

Note that the green 1s and the black 1s are already grouped, therefore this orientation vector is not the appropriate to show how the Algorithm 7—grouping ones on E and updating $\vec{O}$—works, so, we are going to choose the following—and more appropriate—orientation vector, $\vec{O} := \{2, 1, 2, 0\}$, which PA_GQ matrix could be the following,

$$\mathbf{PA_GQ} = \left[\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right]$$

where we have an horizontal line over the green 1 in the second row, and a vertical line over the green one in the forth row. So, with this matrix and the orientation vector $\vec{O}$ we proceed to group black 1s and green 1s, then exploring each p-entry of $\vec{O}$ we get:

- $\vec{O}[1] == 2$ from line 8 of Algorithm 7 that $k_1 = 1$, and increase p—the counter in the outer while loop–, that is, $p = 2$, and pass to next entry,
- $\vec{O}[2] == 1$ so, we assign $\hat{k} = 4$, i.e., the k value, in line 10 we increase k_2 by 1 ,i.e., $k_2 = 1$, and enter in the inner loop, because $flag == 1 \wedge \hat{k} > p$, then we ask if the $\hat{k}$-th entry of the main diagonal is black, in our case $\vec{O}[\hat{k}]$ is green, so in line 16 we goes backward decreasing $\hat{k}$ by 1, i.e., $\hat{k} = 3$, and pass to next inner while loop backwards,
- $\vec{O}[3] == 2$ a black one entry, and $flag == 1 \wedge \hat{k} > p$ is true because $flag$ does not change between the previous inner while loop, and $\hat{k} = 3$ and $p = 2$, then the line 13 is satisfied, and we call `Update`$(\vec{O}[\hat{k} = 3], \vec{O}[p = 2])$, inside of `Update`—see Algorithm 8 we proceed as follows:
  - we "swap" rows $2 \mapsto 3$ and $3 \mapsto 2$ in line 2 of `Update` because $\vec{O}[p]$ was an horizontal green one, so, this Updates PA_GQ. Besides, in line 5 we swap

the orientation vector entries $2 \mapsto 3$ and $3 \mapsto 2$, i.e., updating the vector $\vec{O}$, and in line 6 we increase k_1 by one, so, $k_1 = 2$ this is because we find a black 1 position to exchange and we are not going to explore up to the actual p, otherwise we loss of counting that black one. Finally, in line 7 we decrease the green 1 entry by 1, i.e., $k_2 = 0$, because otherwise we will count twice because we swap with a future entry which will be analyzed and counted by line 10 in the Algorithm 7.

So, the updated data structures are: $\vec{O} := (2, 2, 1, 0)$, and

$$\mathbf{PA_GQ} = \left[\begin{array}{cccc|cc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \tag{A.3.2.1}$$

Finally, $flag := 0$, and pass to the next inner loop condition evaluation, provoking exit of the inner while loop, so, increase p by 1, i.e., $p = 3$, set $flag := 1$, and pass to the next outer while loop condition evaluation,

- $\vec{O}[3] == 1$, so, increase green 1s by 1, i.e., $k_2 = 1$, and $\hat{k}$ restart to 4, and evaluate inner while condition—line 12, that is, $flag == 1 \wedge (\hat{k} = 4) > (p = 3)$, then enter and decrease $\hat{k}$ by one, i.e., $\hat{k} = 3$ because $\vec{O}[\hat{k}]$ is not black, it evaluates again the inner while loop condition, this time is false, and increase p by one, getting $p = 4$, and passing to the next outer while condition evaluation,

- $\vec{O}[4] == 0$ so, increase green 1s by 1, i.e., $k_2 = 2$, and restart $\hat{k} = 4$, and evaluate the condition of the inner while loop, given false, because of $(\hat{k} = 4) < (p = 4)$, then increase p by one then exit of the outer while loop.

After this process, we continue with the computation of the black ones, in a recursive way, then, the square enclosing the green 1s is zeroed out, and we also place zeros wherever those lines crossed the rest of the quadrant—remember Figure A.5.

So, we get the following matrix

$$\mathbf{PA_GQ} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

which correspond to the **Step 2a**, that is, $A_1 = 0 \vee A_2 = 0$, therefore, we choose $\vec{O} := 0^k$ for our k-first position of our orientation vector—in our case are the two first entries—which give us the output of our algorithm (that is, our Permutation-Based algorithm described in Chapter 3 Section 3.3), that is, $\vec{O} := (0, 0, 1, 0)$ which correspond to vertical, vertical, horizontal, and vertical line orientations for the four matching ones in the matrix A.3.2.1. This finish our example.

A.4 Permutations–A survey

This appendix has the purpose to show a very small review of concepts related to permutation which, in particular, we made use in Section 3.4. The concepts and notation related to this appendix are extracted from [Lan08, Chapter 2].

Let S be a set, a bijection $f : S \to S$ is called a *permutation* of S, let Perm(S) be the set of permutations of S.

Proposition A.4.1 *Perm(S) is a group, the law of composition being composition of mapping.*

So, let S_n be the permutation group S_n of $\{1, \ldots, n\} = J_n$ called *symmetric group.*

If $\sigma \in S_n$, then the inverse $\sigma^{-1} : J_n \to J_n$ is the permutation such that $\sigma^{-1}(k) = j$ where $j \in J_n$ is unique such that $\sigma(j) = k$. A *transposition* τ is a permutation which interchanges two position and leaves the rest fixed, i.e.,

$$\exists i, j \quad ((i \neq j) \wedge (\tau(i) = j \wedge \tau(j) = i)) \wedge \forall k((k \neq i \wedge k \neq j) \wedge (\tau(k) = k))$$

where $i, j, k \in J_n$.

Note that if τ is a transposition then $\tau^{-1} = \tau$ and $\tau^2 = I$, where I represent the identity mapping. Particularly, the inverse of a transposition is a transposition. Now we see an important theorem which express that the transpositions generate the permutation group.

Theorem A.4.1 *Every permutation of J_n can be expressed as a product of transpositions.*

A permutaiton σ of $\{1, \ldots, n\}$ is sometimes denoted by

$$\begin{bmatrix} 1 & \cdots & n \\ \sigma(1) & \cdots & \sigma(n) \end{bmatrix}$$

Let $i_1, \ldots, i_r$ be distinct integers in J_n, denoted $[i_1 \cdots i_r]$, meaning the permutation σ

such that

$$\sigma(i_1) = i_2, \sigma(i_2) = i_3, \ldots, \sigma(i_r) = i_1$$

such permutation is called *r-cycle*.

If $\sigma = [i_1 \cdots i_r]$ is a cycle, the one verifies at once that σ^{-1} is also a cycle, and that in fact[24] $\sigma^{-1} = [i_r \cdots i_1]$.

Two cycles $[i_1 \cdots i_n]$ and $[j_1 \cdots \jmath_n]$ in S_n are *disjoint* if no element of $\{1, \ldots, n\}$ is moved by both cycles. This enable us to make use of cycles like follows:

Theorem A.4.2 *Every n-permutation can be written as a product of disjoint cycles.*

We shall see an example to illustrate this theorem:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 3 & 5 & 1 & 7 & 6 \end{bmatrix} = [1245][3][67] = [1245][67]$$

To calculate this one start with a position, say 1, and follow the images round until we get back to, in our choose, 1. Afterwards, we start with the next symbol not accounted for.

Finally, the product of cycles is easily determined, like the following example shows,

$$[1324][23] = [124][3] = [124]$$

because, by definition, if $\sigma = [1324]$ and $\tau = [23]$, then by composition $\sigma \circ \tau$ follows our result. Note that a 2-cycle $\tau = [23]$ is just a transposition such that $3 \mapsto 2$ and $2 \mapsto 3$.

[24] A cycle notation is a more efficient notation to describe permutation because we can misses out the positions that are not moved by the permutation.

A.5 LA Theory–A survey

A complete treatment of this theory can be found in [Sol01].

The logical theory **LA** is strong enough to prove all the ring properties of matrices such as $A(BC) = (AB)C$ and $A + B = B + A$, but weak enough so that the theorems of **LA** translate into propositional tautologies with short Frege proofs. **LA** has three sorts of object: *indices* (i.e., natural numbers), *ring elements*, and *matrices*, where the corresponding variables are denoted $i, j, k, \ldots$; $a, b, c, \ldots$; and $A, B, C, \ldots$, respectively. The semantic assumes that objects of type ring are from a fixed but arbitrary ring (for the purpose of this thesis we are only interested in the ring $\mathbb{Z}$), and objects of type matrix have entries from that ring.

Terms and formulas are built from the following function and predicate symbols, which together comprise the language $\mathcal{L}_{\mathbf{LA}}$:

$$\begin{aligned} &0_{\text{index}}, 1_{\text{index}}, +_{\text{index}}, *_{\text{index}}, -_{\text{index}}, \mathtt{div}, \mathtt{rem}, \\ &0_{\text{ring}}, 1_{\text{ring}}, +_{\text{ring}}, *_{\text{ring}}, -_{\text{ring}}, {}^{-1}, \mathtt{r}, \mathtt{c}, \mathtt{e}, \Sigma, \\ &\leq_{\text{index}}, =_{\text{index}}, =_{\text{ring}}, =_{\text{matrix}}, \mathtt{cond}_{\text{index}}, \mathtt{cond}_{\text{ring}} \end{aligned} \tag{A.5.0.2}$$

The intended meaning should be clear, except in the case of $-_{\text{index}}$, cut-off subtraction, defined as $i - j = 0$ if $i < j$. For a matrix A: $\mathtt{r}(A), \mathtt{c}(A)$ are the number of rows and columns in A, $\mathtt{e}(A, i, j)$ is the ring element A_{ij} (where $A_{ij} = 0$ if $i = 0$ or $j = 0$ or $i > \mathtt{r}(A)$ or $j > \mathtt{c}(A)$), $\Sigma(A)$ is the sum of the elements in A. Also $\mathtt{cond}(\alpha, t_1, t_2)$ is interpreted **if** α **then** t_1 **else** t_2, where α is a formula all of whose atomic sub-formulas have the form $m \leq n$ or $m = n$, where m, n are terms of type index, and t_1, t_2 are terms either both of type index or both of type ring. The subscripts $_{\text{index}}$, $_{\text{ring}}$, and

$_{\text{matrix}}$ are usually omitted, since they ought to be clear from the context.

We use n, m for terms of type index, t, u for terms of type ring, and T, U for terms of type matrix. Terms of all three types are constructed from variables and the symbols above in the usual way, except that terms of type matrix are either variables $A, B, C, ...$ or λ-terms $\lambda ij\langle m, n, t\rangle$. Here i and j are variables of type index bound by the λ operator, intended to range over the rows and columns of the matrix. Also m, n are terms of type index *not* containing i, j (representing the number of rows and columns of the matrix) and t is a term of type ring (representing the matrix element in position (i, j)).

Atomic formulas are of the form $m \leq n, m = n, t = u$ and $T = U$, where the three occurrences of $=$ formally have subscripts $_{\text{index}}$,$_{\text{ring}}$,$_{\text{matrix}}$, respectively. General formulas are built from atomic formulas using the propositional connectives $\neg, \vee, \wedge$ and quantifiers $\forall, \exists$.

A.5.1 Defined terms

The λ terms allow us to construct the sum, product, transpose, etc., of matrices. As usual, $:=$ denotes a definition—in this context this amounts to an abbreviations for terms.

Integer maximum

$$\mathtt{max}\{i, j\} := \mathtt{cond}(i \leq j, j, i)$$

Matrix sum

$$\begin{aligned} &A + B := \\ &\lambda ij\langle \mathtt{max}\{\mathtt{r}(A), \mathtt{r}(B)\}, \mathtt{max}\{\mathtt{c}(A), \mathtt{c}(B)\}, A_{ij} + B_{ij}\rangle \end{aligned} \tag{A.5.1.1}$$

Note that $A + B$ is well defined even if A and B are incompatible in size, because of our convention that out-of-bound entries are 0.

Scalar product

$$aA := \lambda ij\langle \mathtt{r}(A), \mathtt{c}(A), a * A_{ij}\rangle \tag{A.5.1.2}$$

Matrix transpose

$$A^t := \lambda ij\langle \mathtt{c}(A), \mathtt{r}(A), A_{ji}\rangle \tag{A.5.1.3}$$

Zero and Identity matrices

$$\begin{aligned} 0_{kl} &:= \lambda ij\langle k, l, 0\rangle \\ I_k &:= \lambda ij\langle k, k, \mathtt{cond}(i = j, 1, 0)\rangle \end{aligned} \tag{A.5.1.4}$$

Sometimes we will just write 0 and I when the sizes are clear from the context.

Matrix trace

$$\mathrm{tr}(A) := \Sigma\lambda ij\langle \mathtt{r}(A), 1, A_{ii}\rangle \tag{A.5.1.5}$$

Dot product

$$\begin{aligned} &A \cdot B := \\ &\Sigma\lambda ij\langle \mathtt{max}\{\mathtt{r}(A), \mathtt{r}(B)\}, \mathtt{max}\{\mathtt{c}(A), \mathtt{c}(B)\}, A_{ij} * B_{ij}\rangle \end{aligned} \tag{A.5.1.6}$$

Matrix product

$$\begin{aligned} &A * B := \\ &\lambda ij\langle \mathtt{r}(A), \mathtt{c}(B), \\ &\lambda kl\langle \mathtt{c}(A), 1, \mathtt{e}(A, i, k)\rangle \cdot \lambda kl\langle \mathtt{r}(B), 1, \mathtt{e}(B, k, j)\rangle\rangle \end{aligned} \tag{A.5.1.7}$$

Finally, the following decomposition of an $n \times n$ matrix A will be used in our axioms defining $\Sigma(S)$:

$$A = \begin{pmatrix} a_{11} & R \\ S & M \end{pmatrix} \tag{A.5.1.8}$$

where a_{11} is the $(1,1)$ entry of A, and R, S are $1 \times (n-1)$, $(n-1) \times 1$ submatrices, respectively, and M is the principal submatrix of A. Therefore, we make the following precise definitions:

$$\begin{aligned} \mathtt{R}(A) &:= \lambda ij\langle 1, \mathtt{c}(A) - 1, \mathtt{e}(A, 1, i+1)\rangle, \\ \mathtt{S}(A) &:= \lambda ij\langle \mathtt{r}(A) - 1, 1, \mathtt{e}(A, i+1, 1)\rangle, \\ \mathtt{M}(A) &:= \lambda ij\langle \mathtt{r}(A) - 1, \mathtt{c}(A) - 1, \mathtt{e}(A, i+1, j+1)\rangle. \end{aligned} \tag{A.5.1.9}$$

A.5.2 Axioms and rules of LA

For each axiom listed below, every legal substitution of terms for free variables is an axiom of **LA**. Note that in a λ term $\lambda ij\langle m, n, t\rangle$ the variables i, j are bound. Substitution instances must respect the usual rules which prevent free variables from being caught by the binding operator λij. The bound variables i, j may be renamed to any new distinct pair of variables.

A.5.2.1 Equality Axioms

These are the usual equality axioms, generalized to apply to the three-sorted theory **LA**. Here $=$ can be any of the three equality symbols, x, y, z are variables of any of the three sorts (as long as the formulas are syntactically correct). In A4, the symbol

f can be any of the non-constant function symbols of **LA**. However A5 applies only to $\leq$, since this in the only predicate symbol of **LA** other than $=$.

A1 $x = x$

A2 $x = y \rightarrow y = x$

A3 $(x = y \wedge y = z) \rightarrow x = z$

A4 $x_1 = y_1, ..., x_n = y_n \rightarrow fx_1...x_n = fy_1...y_n$

A5 $i_1 = j_1, i_2 = j_2, i_1 \leq i_2 \rightarrow j_1 \leq j_2$

A.5.2.2 Axioms for indices

These are the axioms that govern the behavior of index elements. The index elements are used to access the entries of matrices, and so we need to define some basic number theoretic operations.

A6 $i + 1 \neq 0$

A7 $i * (j + 1) = (i * j) + i$

A8 $i + 1 = j + 1 \rightarrow i = j$

A9 $i \leq i + j$

A10 $i + 0 = i$

A11 $i \leq j \wedge j \leq i$

A12 $i + (j + 1) = (i + j) + 1$

A13 $[i \leq j \wedge j \leq i] \rightarrow i = j$

A14 $i * 0 = 0$

A15 $[i \leq j \wedge i + k = j] \rightarrow j - i = k$

A16 $\neg(i \leq j) \rightarrow j - i = 0$

A17 $[\alpha \rightarrow \text{cond}(\alpha, i, j) = i] \wedge [\neg\alpha \rightarrow \texttt{cond}(\alpha, i, j) = j]$

A.5.2.3 Axioms for a ring

These are the axioms that govern the behavior for ring elements; addition and multiplication, as well as additive inverses. We do not need multiplicative inverses.

A18 $0 \neq 1 \wedge a + 0 = a$

A19 $a + (-a) = 0$

A20 $1 * a = a$

A21 $a + b = b + a$

A22 $a * b = b * a$

A23 $a + (b + c) = (a + b) + c$

A24 $a * (b * c) = (a * b) * c$

A25 $a * (b + c) = a * b + a * c$

A26 $[\alpha \rightarrow \mathtt{cond}(\alpha, a, b) = a] \wedge [\neg\alpha \rightarrow \mathtt{cond}(\alpha, a, b) = b]$

A.5.2.4 Axioms for matrices

Axiom A27 states that $\mathtt{e}(A, i, j)$ is zero when i, j are outside the size of A. Axiom A28 defines the behavior of constructed matrices. Axioms A29-A32 define the function Σ recursively by first defining it for row vectors, then column vectors ($A^t := \lambda ij\langle \mathtt{c}(A), \mathtt{r}(A), A_{ji}\rangle$), and then in general using the decomposition (A.5.2.1). Finally, axiom A33 takes care of empty matrices.

A27 $(i = 0 \vee \mathtt{r}(A) < i \vee j = 0 \vee \mathtt{c}(A) < j) \rightarrow \mathtt{e}(A, i, j) = 0$

A28 $\mathtt{r}(\lambda ij\langle m, n, t\rangle) = m \wedge \mathtt{c}(\lambda ij\langle m, n, t\rangle) = n \wedge [1 \leq i \wedge i \leq m \wedge 1 \leq j \wedge j \leq n] \rightarrow$
$\rightarrow \mathtt{e}(\lambda ij\langle m, n, t\rangle, i, j) = t$

A29 $\mathtt{r}(A) = 1, \mathtt{c}(A) = 1 \rightarrow \Sigma(A) = \mathtt{e}(A, 1, 1)$

A30 $\mathtt{r}(A) = 1 \wedge 1 < \mathtt{c}(A) \to \Sigma(A) = \Sigma(\lambda ij\langle 1, \mathtt{c}(A) - 1, A_{ij}\rangle) + A_{1\mathtt{c}(A)}$

A31 $\mathtt{c}(A) = 1 \to \Sigma(A) = \Sigma(A^t)$

A32 $1 < \mathtt{r}(A) \wedge 1 < \mathtt{c}(A) \to \Sigma(A) = \mathtt{e}(A, 1, 1) + \Sigma(\mathtt{R}(A)) + \Sigma(\mathtt{S}(A)) + \Sigma(\mathtt{M}(A))$

A33 $\mathtt{r}(A) = 0 \vee \mathtt{c}(A) = 0 \to \Sigma A = 0$

Where

$$\begin{aligned} \mathtt{R}(A) &:= \lambda ij\langle 1, \mathtt{c}(A) - 1, \mathtt{e}(A, 1, i+1)\rangle, \\ \mathtt{S}(A) &:= \lambda ij\langle \mathtt{r}(A) - 1, 1, \mathtt{e}(A, i+1, 1)\rangle, \\ \mathtt{M}(A) &:= \lambda ij\langle \mathtt{r}(A) - 1, \mathtt{c}(A) - 1, \mathtt{e}(A, i+1, j+1)\rangle. \end{aligned} \tag{A.5.2.1}$$

A.5.2.5 Rules for LA

In addition to all the axioms just presented, **LA** has two rules: matrix equality and induction.

Matrix equality rule

From the premises: $\mathtt{e}(T, i, j) = \mathtt{e}(U, i, j)$, $\mathtt{r}(T) = \mathtt{r}(U)$ and $\mathtt{c}(T) = \mathtt{c}(U)$, we conclude $T = U$.

The only restriction is that the variables i, j may not occur free in $T = U$; other than that, T and U can be arbitrary matrix terms. Our semantics implies that i and j are implicitly universally quantified in the top formula. The rule allows us to conclude $T = U$, provided that T and U have the same number of rows and columns, and corresponding entries are equal.

Induction rule $\alpha(i) \to \alpha(i+1)$ implies $\alpha(0) \to \alpha(n)$.

Here $\alpha(i)$ is any formula, n is any term of type index, and $\alpha(n)$ indicates n is substituted for free occurrences of i in $\alpha(i)$. (Similarly for $\alpha(0)$.) Note that in **LA** we only allow induction over Σ_0^B formulas (no matrix quantifiers), whereas in $\exists$**LA** we allow induction over Σ_1^B formulas (a single block of bounded existential matrix quantifiers when α is put in prenex form). This completes the description of **LA**. We finish this section by observing the substitution property in the lemma below. We say that a formula S' of **LA** is a *substitution instance* of a formula S of **LA** provided that S' results by substituting terms for free variables of S. Of course each term must have the same sort as the variable it replaces, and bound variables must be renamed as appropriate.

Lemma A.5.1 *Every substitution instance of a theorem of* **LA** *is a theorem of* **LA**.

This follows by straightforward induction on **LA** proofs. The base case follows from the fact that every substitution instance of an **LA** axiom is an **LA** axiom.

Bibliography

[Aha83] R. Aharoni. *Menger's theorem for graphs containing no infinite paths. Europe J. Combinatorics*, 4, pp. 201–204, 1983.

[Aha84] R. Aharoni. *König's Duality Theorem for infinite bipartite graphs. J. London Math. Soc. (3)*, 29,pp. 1–12, 1984.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AHU83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[Als99] M. H. Alsuwaiyel, *Algorithms*. World Scientific, 1999.

[AB09] S. Arora, B. Barak. *Computational Complexity: A modern approach*. Cambridge University Press, 2009.

[ADH98] A. S. Asratian, T. M. J. Denley, and R. Häggkvist, *Bipartite graphs and their applications*. Cambridge University Press, 1998.

[BP98] Paul Beame, and Toniann Pitassi. Propositional proof complexity: Past, present, and future. *Bulletin of the EATACS*, TR98-067, 1998.

[Ber57] Berge C., Two theorems in graph theory. *Proc. Nat. Acad. Sci. U.S.A.*, 43:842–844, 1998.

[Ber84] Berkowitz S. J., On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984.

[BR91] Richard A. Brualdi and Herbert J. Ryser. *Combinatorial Matrix Theory.* Cambridge University Press, 1991.

[Bus86] Sam R. Buss. *Bounded Arithmetic.* Bibliopolis, Naples, Italy, 1986.

[Bus90] Samuel R. Buss. Axiomatizations and conservations results for fragments of Bounded Arithmetic. *AMS Contemporary Mathematics*, 106:57–84, 1990.

[Bus98] S. R. Buss. *An introduction to proof theory,* In S. R. Buss (eds.), *Handbook of Proof Theory.* North Holland, 1998, pages 1–78.

[Coo75] Stephen A. Cook, Feasibly constructive proofs and the propositional calculus. *Proc. 7th ACM Symposium on the Theory of Computation*, 1975, pages 83–97.

[Coo85] Stephen A. Cook, A taxonomy of problems with fast parallel algorithms. *Information and Computation*, 64(13):2–22, 1985.

[CR79] Stephen A. Cook, and Robert A. Reckhow. The relative efficiency of propositional proof systems. *JSL*, 44:36–50, 1979.

[SC04] Stephen A. Cook, M. Soltys. The Proof Complexity of Linear Algebra.

In *Annals of Pure and Applied Logic, Vol. 130, Num. 1–3*, 2004, pages 277–323.

[CN10] Stephen A. Cook, P. Nguyen. *Logical Foundations of proof complexity*. Cambridge University Press, 2010.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill Book Company, 2009. Second Edition.

[CD12] D. T. M. Lê and Stephen A. Cook, "Formalizing randomized matching algorithms," *Logical Methods in Computer Science*, vol. 8, pp. 1–25, 2012.

[DFS88] E. W. Dijkstra, W. H. J. Feijen, J. Sterringa *A Method of Programming*. Addison-Wesley, 1988.

[Dij76] Edsger W. Dijkstra *A Discipline of Programming*. Prentice Hall, 1976.

[Dil50] R. P. Dilworth, "A decomposition theorem for partially ordered sets," *Annals of Mathematics*, vol. 51, no. 1, pp. pp. 161–166, January 1950. [Online]. Available: http://www.jstor.org/stable/1969503

[DM58a] A. L. Dulmage and N. S. Mendelsohn. Coverings of Bipartite Graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

[DM58b] A. L. Dulmage and N. S. Mendelsohn. Some generalizations of the porblem of distinct representatives. *Canadian Journal of Mathematics*, 10:230–241, 1958.

[Edm65] J. Edmond, "Paths, tree, and flowers". *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965.

[EW49] C. J. Everett and G. Whaples, "Representations of sequences of sets," *American Journal of Mathematics*, vol. 71, no. 2, pp. pp. 287–293, April 1949. [Online]. Available: http://www.jstor.org/stable/2372244

[GJ79] M. R. Garey and D. S. Johnson, *Computers and Intractability*. W.H.Freeman and Company, San Francisco, 1979.

[Gon84] G. H. Gonnet. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1984.

[Go00] F. Göring, "Short proof of Menger's theorem," *Discrete Mathematics*, vol. 219, pp. 295–296, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0012365X00000881

[Gup67] R. P .Gupta. "A decomposition theorem for bipartite graphs (results)" in *Theory of Graphs (International Symposium, Rome, 1966)*, Ed. P. Rosenstiehl, Gordon and Breach, New York, 1967, pp. 135–138.

[Hal56] M. Hall, Jr. "An algorithm for distinct representatives". *Amer. Math, Monthly*, pp. 716–717, 1956.

[Hal87] P. Hall, "On representatives of subsets," in *Classic Papers in Combinatorics*, ser. Modern Birkhäuser Classics, I. Gessel and G.-C. Rota, Eds. Birkhäuser Boston, 1987, pp. 58–62.

[HV50] P. R. Halmos and H. E. Vaughan, "The marriage problem," *American*

Journal of Mathematics, vol. 72, no. 1, pp. pp. 214–215, Januar 1950. [Online]. Available: http://www.jstor.org/stable/2372148

[HS86] J. R. Hindley, J. P. Seldin. *Introduction to combinators and lambda calculus*. Cambridge University Press, 1986.

[HT11] P. Hrubes and I. Tzameret, "Short proofs for the determinant identities," *CoRR*, vol. abs/1112.6265, December 2011.

[Je05] E. Jeřábek, "Weak pigeonhole principle, and randomized computation," Ph.D. dissertation, Faculty of Mathematics and Physics, Charles University, Prague, 2005.

[JEH73] Richard M. Karp John E. Hopcroft. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4), December 1973.

[Kar72] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Kön16a] Dénes König. Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére. *Matematikai és Természettudományi Értesítö*, 34:104–119, 1916.

[Kön16b] Dénes König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4), 1916.

[Kön36] Dénes König. *Theorie der endlichen und unendlichen Graphen*. Akademischen Verlags-gesellschaft, Leipzig, 1936.

[Kra95] J. Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, 1995.

[Kra10] J. Krajíček. From feasible proofs to feasible computations. In *Computer Science logic.* LNCS 6247, Springer, Berlin, 2010, pages 22–31.

[Kuh55] H. W. Kuhn, "The Hungarian method for the assignment problem". *Naval Res. Logist. Quart.*, 2, pp. 83–97, 1955.

[Lan08] S. Lang. *Undergraduate Algebra*, Springer-Verlag, 2008 .

[LP86] L. Lovász and M.D. Plummer. *Matching Theory.* North-Holland, 1986.

[Meh84] K. Mehlhorn. *Graph Algorithms and NP-Completeness*, vol. 2 of *Data Structure and Algorithms* Springer-Verlag, 1984.

[Meng27] K. Menger, "Zur allgemeinen kurventheorie," *Fund. Math*, vol. 10, no. 95-115, 1927.

[Per63] M. A. Perles, "A proof of dilworth's decomposition theorem for partially ordered sets," *Israel Journal of Mathematics*, vol. 1, pp. 105–107, 1963.

[Pym96] J. S. Pym, "A proof of Menger's theorem," *Monatshefte für Mathematik*, vol. 73, no. 1, pp. 81–83, 1969.

[Sim92] S. G. Simpson. *On the strength of Kónig's duality theorem for countable bipartite graphs.* Journal of Symbolic Logic, 59, 1994, pp. 113-123.

[Sim99] S. Simpson. *Subsystems of Second Order Arithmetic.* Springer, 1999.

[Sch82] A. Schrijver. Min-max relations for directed graphs. In *Bonn Workshop on Combinatorial Optimization*, volume 16 of *Ann. Discrete Math.*, pages 261–280̇North-Holland, 1982.

[Sol01] M. Soltys. *The complexity of derivations of matrix identities*. PhD thesis, University of Toronto, 2001.

[SU04] M. Soltys, A. Urquhart. *Matrix identities and the pigeonhole principle*. Archive for Mathematical Logic 43(3), April 2004, pages 351–357.

[Sol04] M. Soltys, "LA, permutations, and the Hajós calculus," *Theoretical Computer Science*, vol. 348, no. 2–3, pp. 321–333, December 2005.

[Sol09] M. Soltys. *An introduction to Computational Complexity*. Jagiellonian University Press, 2009.

[Sol11] M. Soltys, "Feasible proofs of Szpilrajn's theorem: a proof-complexity framework for concurrent automata," *Journal of Automata, Languages and Combinatorics (JALC)*, vol. 16, no. 1, pp. 27–38, 2011.

[Sol12] M. Soltys. *An introduction to the Analysis of Algorithms*. World Scientific Press, 2012.

[SF12] M. Soltys and A. Fernandez, "A linear-time algorithm for computing minimum vertex covers from maximum matchings, October 2012.

[We01] D. B. West. *Graph Theory*, Second Edition. Prince Hall, 2001.

[Zwi09] U. Zwick. *Lecture notes on maximum matching in bipartite graphs and non-bipartite graphs*, 2009.

Index