A Permutation-Based Algorithm for Computing Covers from Matchings*

Ariel Fernandez[1], Ryszard Janicki[1] and Michael Soltys[2]
[1]Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada L8S 4L8
{fernanag,janicki}@mcmaster.ca
[2]Department of Computer Science, California State University at Channel Islands
One University Drive, Camarillo, CA 93012, U.S.A.
michael.soltys@csuci.edu

Abstract

We present a matrix permutation algorithm for computing a minimal vertex cover from a maximal matching in a bipartite graph. Our algorithm is linear time and linear space, and provides an interesting perspective on a well known problem. Unlike most algorithms, it does not use the concept of alternating paths, and it is formulated entirely in terms of combinatorial operations on a binary matrix.

keywords: minimal vertex cover, bipartite graph, maximal matching, König's Mini-Max theorem

1 Introduction

In this paper we provide a new better solution to an old problem.

Suppose that we are given a bipartite graph $G = (V = V_1 \cup V_2, E)$, i.e., a graph where $V_1 \cap V_2 = \emptyset$ and $E \subseteq V_1 \times V_2$. Let A_G be the adjacency matrix of G, of size $|V_1| \times |V_2|$, and with 0-1 entries: $(i,j) \in E$ iff $(A_G)_{ij} = 1$. 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. We can represent a matching as a set of pairs of nodes of V, i.e., $M \subseteq E$, or as an adjacency matrix. A matching M is maximal if $|M|$ is as large as possible, i.e., when $|M|$ is maximum. We talk of bipartite graphs and their adjacency matrix representations interchangeably.

A vertex cover of a graph $G = (V,E)$ is a set of vertices $C \subseteq V$ such that each edge of the graph is incident to at least one vertex of the set C, i.e. for each $e = (v_1, v_2) \in E$, $\{v_1, v_2\} \cap C \neq \emptyset$. A vertex cover C is minimal if $|C|$ is minimum.

It is well known that given a general graph, a maximal matching can be computed with the classical Edmond's blossom algorithm ([4]) in $O(|V|^4)$ time, or the more complex $O(|V|^{\frac{1}{2}}|E|)$ algorithm by Micali and Vazirani [13]. For bipartite graphs, the easiest solution is to use the Ford-Fulkerson algorithm for flows [6], either directly, or its modified version based on the concept of alternating paths, i.e., paths that alternate between edges that are in the matching and edges that are not in the matching, (c.f. [1]), both run in $O(|V||E|)$ time; or we can use the more efficient (especially for sparse graphs) Hopcroft-Karp algorithm [8] which again runs in $O(|V|^{\frac{1}{2}}|E|)$, or, for dense graphs, the algorithm of [2] which runs in $O\left(|V|^{1.5}\sqrt{\frac{|E|}{\log|V|}}\right)$.

On the other hand, for general graphs, the problem of computing minimal vertex covers is **NP**-hard [9]. For bipartite graphs, due to König's Mini-Max Theorem [10, 11], minimal vertex covers can be derived from maximal matchings in $O(|V|+|E|)$ time (c.f. [14, Lemma 3]). This means that for all algorithms known so far, the time complexity of computing a minimal vertex cover for bipartite graphs is the same as time complexity of computing an appropriate maximal matching. All widely known derivation methods use the idea of alternating paths (or equivalent concepts) (c.f. [7, 15]).

In this paper we use a different approach. We start with a matrix version of König's Mini-Max Theorem, instead of its more popular graph theory version. In fact we use graph theory terminology for readability only, as they are not really needed to present and implement our solution. All operations are simple matrix operations which can be implemented very efficiently.

Our algorithm is linear in both time and space with respect to the size of a binary matrix that represents a given *bipartite* graph. No assumptions are made about the method for computing maximal matchings.

2 Problem Formulation and König's Mini-Max Theorems

Let $M_G = \mathsf{MA}(G)$ be the output of running an algorithm MA that computes a maximal matching for a given bipartite graph $G = (V_1 \cup V_2, E)$, i.e., M_G is the adjacency matrix of a

*This research was partially supported by NSERC Grant of Canada.

978-1-943436-06-4 / copyright ISCA, CATA 2017
March 20-22, 2017, Honolulu, Hawaii, USA

maximal matching produced by the algorithm MA. Let A_G be an adjacency matrix that defines the graph G, and let M_G denote the adjacency matrix for a maximal matching of G. Note that both A_G and M_G are of size $|V_1| \times |V_2|$, and thus four times smaller than a standard $|V| \times |V|$ representation. In what follows we will assume bipartite graphs to be represented by $|V_1| \times |V_2|$ adjacency matrices.

We find it useful to give two equivalent formulations of König's theorem. The first one is the standard formulation that uses the language of graphs.

Theorem 1 (König's Mini-Max version I)**.** *Given a bipartite graph G, if ρ_G is the size of the maximal matching of G and ρ'_G is the size of the minimal vertex cover of G, then $\rho_G = \rho'_G$.*

The proof of Theorem 1 (c.f. [3]) furnishes the basic ideas that will be used later in Section 3 to transform maximal matchings into minimal covers.

Given an $m \times n$ 0-1 matrix A, let S_A be a set of pairs of indices, i.e., $S_A = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\} \subseteq \mathbb{N} \times \mathbb{N}$, 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_A 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_A is called the *term rank* of A ([3]). Notice that if A_G is the adjacency matrix of a bipartite graph G, then the term rank equals the size of a maximal matching in G.

On the other hand, given a 0-1 matrix A of size $m \times n$, 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 minimal vertex cover corresponds to the minimal cover of A_G.

We now restate König's theorem but using the language of matrices.

Theorem 2 (König's Mini-Max version II)**.** *Let A be a 0-1 matrix of size $m \times n$. 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 the same line.*

Since a bipartite graph G can be identified with its 0-1 matrix representation A_G of size $|V_1| \times |V_2|$, we may write $M_{A_G} = \mathsf{MA}(A_G)$ instead of $M_G = \mathsf{MA}(G)$.

Our goal is to design an algorithm, which we call PERALG, that takes an input $\langle A_G, M_{A_G} \rangle$ and produces a set of lines C_{A_G} that form a minimal cover of A_G. We want PERALG to compute C_{A_G} in $O(|A_G|)$, where $|A_G|$ is the size of the matrix A_G.

3 Preliminaries to our algorithm

Our permutation-based algorithm, PERALG, runs in time $O(|V_1||V_2|)$. Since $|V_1||V_2| = |A_G|$, the size of the matrix representing G, PERALG runs in linear time in the size of $|A_G|$, assuming a model of computation (such as RAM) where the matrix entries can be accessed at cost $O(1)$.

The main idea behind PERALG is that, given a maximal matching M of a bipartite G, the minimal vertex cover C can be constructed by taking, for each $e \in M$, one of e's end point nodes. Of course, not all $2^{|M|}$ selections of end-points work, but at least one selection of end-points works. We show the details in Lemma 1.

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

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

A cover is a set of lines $C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}} = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\}$, with *orientation* $\boldsymbol{o} = o_1 o_2 \ldots o_k$, and $\boldsymbol{i} = i_1, i_2, \ldots, i_k$, $\boldsymbol{j} = j_1, j_2, \ldots, j_k$, and it is such that any 1 in A is covered by (at least) 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 $l^{o_p}_{(i_p,j_p)} \in C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$ and $[i = i_p \wedge o_p = 0] \vee [j = j_p \wedge o_p = 1]$.

If M_A is a maximal matching, the Mini-Max theorem says that there exist $\boldsymbol{o}, \boldsymbol{i}, \boldsymbol{j}$ of length $k = |M_A|$ such that $C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$ is a cover. Recall that M_A represents a maximal matching as a 0-1 matrix, 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 $C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$ must have lines through all the 1s of M_A; further, any such line cannot cover more than a single 1. Since we know that the size of $C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$ is the size of M_A, each 1 of M_A claims exactly one line. Hence $\boldsymbol{i}, \boldsymbol{j}$ are directly determined by M_A, but $\boldsymbol{o}$ needs to be computed.

Lemma 1. *Suppose that $G = (V = V_1 \cup V_2, E)$ is a bipartite graph, A its adjacency matrix, and M_A a maximal 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

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

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

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 Mini-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. □

Given permutations $\pi : [m] \longrightarrow [m]$ and $\tau : [n] \longrightarrow [n]$, let P_π and Q_τ be the corresponding permutation matrices. The matrices P_π and Q_τ are obtained from the identity matrix by exchanging the rows according to π and τ, respectively. Then: $(P_\pi M_A Q_\tau)_{ij} = (M_A)_{\pi^{-1}(i)\tau^{-1}(j)}$.

Given an $m \times n$ matrix A, and given a maximal matching M_A, which we represent as $M_A = \{(i_1,j_1),(i_2,j_2),\ldots,(i_k,j_k)\}$, where $i_1 < i_2 < \cdots < i_k$, then the pair of permutations:

$$\begin{array}{ccc|ccc} & \pi & & & \tau & \\ i_1 & \mapsto & 1 & j_1 & \mapsto & 1 \\ i_2 & \mapsto & 2 & j_2 & \mapsto & 2 \\ & \vdots & & & \vdots & \\ i_k & \mapsto & k & j_k & \mapsto & k \end{array}$$

are *order preserving permutations according to rows (for M_A)*. Note that the indices that are not specified are left fixed by π, τ.

That is, $P_\pi M_A Q_\tau$ place the 1s on the main diagonal, in the original order of the rows of M_A. Notice also that:

$$P_\pi A Q_\tau = \left[\begin{array}{c|c} T & A_1 \\ \hline A_2 & 0_{(m-k)\times(n-k)} \end{array}\right], \qquad (1)$$

where $T = [t_{ij}]_{k\times k}$ and $t_{ii} = 1$ while $t_{ij} = 0$ for $i \neq j$. That is, the 1s in M_A are permuted to be on the diagonal of the upper-left $k \times k$ quadrant T. 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 the Min-Max theorem: all the lines in $C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$ pass through a 1 in T; none of these lines can possibly touch this lower-right quadrant, so it must be full of zeros.

For example, suppose that we have a graph G as in Figure 1; examine the values of M_A and $P_\pi A Q_\tau$.

The next Lemma relates the covering of the original A to the covering of permuted A, i.e., $P_\pi A Q_\tau$.

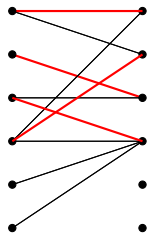
Lemma 2. *Suppose that π, τ are order preserving permutations according to rows. Then, if*

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

is a covering of A, then

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

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



(a) Graph G

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

(b) $A = A_G$

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

(c) $M_A = M_{A_G}$

$$\begin{bmatrix} \mathbf{1} & 0 & 0 & 1 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & \mathbf{1} & 0 & 0 & 0 \\ 1 & 0 & 1 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(d) $P_\pi A_G Q_\tau$

Figure 1: Then, π is the identity permutation, and τ fixes 1, moves 2 to 4, 3 to 2, 4 to 3, and fixes 5 and 6. The lines in red in (a) indicate a maximal matching; the red ones in (d) indicate the corresponding lines, now placed on diagonal. Note that the 2×2 lower-right submatrix is zero as it should.

Proof. Suppose that $C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}} = \{l^{o_1}_{(i_1,j_1)}, l^{o_2}_{(i_2,j_2)}, \ldots, l^{o_k}_{(i_k,j_k)}\}$ 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 $r_{\pi^{-1}(p)} \in C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$ or $c_{\tau^{-1}(q)} \in C_A^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}$. This last statement means that there exists an $a \in [k]$ such that at least one of the following two statements is true:

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

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

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

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

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, $C^{\boldsymbol{o},\pi(\boldsymbol{i}),\tau(\boldsymbol{j})}_{P_\pi A Q_\tau}$ is indeed a covering for $P_\pi A Q_\tau$. □

The purpose of Lemma 2 is to show that given A_G, we can reorder its rows and columns at will — which corresponds to a relabelling of V_1 and V_2 — and the resulting matrix has a maximal matching and minimal vertex cover of the same size. Furthermore, we can easily compute the maximal matching and vertex cover for the resulting matrix from the original one. Note that we assumed in Lemma 2 that the permutations are order preserving permutations (according to rows), and hence the orientation vector $\boldsymbol{o}$ is not affected. If the permutations are not order preserving, then we can still recompute the maximal matching and minimal vertex cover, but we must apply the corresponding permutation to the orientations. This is summarized in Corollary 1 below.

Corollary 1. *Suppose that π,τ are order preserving permutations according to rows, and that the diagonal 1s have been reordered by μ. Then, if $C^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}_A$ is a covering of A, then*

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

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

4 Our algorithm

On input $\langle A, M_A\rangle$, where M_A is a maximal matching for A, PERALG computes a minimal cover $C^{\boldsymbol{o},\boldsymbol{i},\boldsymbol{j}}_A$. More precisely, as was shown in Lemma 1, given M_A we know *a priori* that:

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

is a minimal covering for some $\boldsymbol{o}$, 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 $\boldsymbol{o} = o_1 o_2 \ldots o_k$.

The analogy in the graph theoretic setting is the following: given a bipartite graph G and a maximal matching M the minimal vertex cover can be selected from M. This selection takes place 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 PERALG for computing the orientations.

PERALG:
Input: A, M_A, $m \times n$ 0-1 matrices, where M_A is a maximal matching for A:

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

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

Step 2 Else, $k = |M_A| < \min\{m,n\}$, compute order preserving permutations π,τ that diagonalize M_A, so $P_\pi A Q_\tau = \left[\begin{array}{c|c} T & A_1 \\ \hline A_2 & 0_{(m-k)\times(n-k)} \end{array}\right]$ (Recall (1).)

Step 2a If $A_1 = 0 \vee A_2 = 0$:

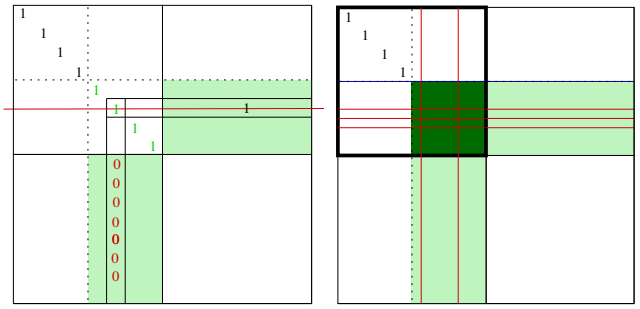
- If $A_1 = 0$ then return $\boldsymbol{o} = 0^k$ and exit
- If $A_2 = 0$ then return $\boldsymbol{o} = 1^k$ and exit

Step 2b Else, $A_1 \neq 0 \wedge A_2 \neq 0$. Group the 1s on the diagonal of T into two sets, the black and the green, of sizes k_1 and k_2, respectively, with $k = k_1 + k_2$. A black 1 in position (i,i) has the property that both row i of A_1 and column i of A_2 consist of zeros. The green 1s do not have this property. 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 $o_j = 0$.

Let T' be T where the 1s under the lines covering the green 1s have been zeroed out (see Figure 2). In order to compute the orientations of the black 1s, repeat recursively Step 2a on T'.

Output orientations $\boldsymbol{o}$



(a) Step 2b (b) Repeat Step 2 with T'

Figure 2: In (b), T' is 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.

Note that the situation, as represented in Figure 2(a), 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 2(b), but that would require in general a permutation that is not order preserving; this could still be done by Corollary 1, but it introduces a technical overhead, as the orientations would no longer match (but we can

always recover the original orientations by inverting the permutation).

Also note that in Step 2b, 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 $k = k_2$.

Lemma 3. *Algorithm* PERALG *is correct. That is, given A and M_A as input, it computes $\mathbf{o}$ so that $C_A^{\mathbf{o},i,j}$ is a vertex cover (of size $|M_A|$).*

Proof. By placing 1s on the diagonal of T, we ensure that each such 1 requires at least one line to be covered. By Lemma 1 we can conclude that exactly one line per 1 on the diagonal of T is sufficient to ensure a cover.

In Step 2b, if there is a 1 in row j of A_1, then we let $o_j = 1$, and otherwise we let $o_j = 0$. We know that this works because each 1 in T claims exactly one line in the cover. So it follows 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).

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. □

In the following lemma, we show how to compute order preserving π, τ in Step 2. It is clear that it can be done in linear space, that is, in space $O(|A|)$. Except for the computation of order preserving π, τ once in Step 2, the recursive computation of the orientations is done inside A, with constantly many registers indexing A (space $O(\log(|A|))$) and hence also in space $O(|A|)$. Thus, PERALG requires linear space.

Lemma 4. *Algorithm* PERALG *runs in time $|A| = m \times n$.*

Proof. We show first the details of computing order preserving π, τ in Step 2. We initialize $r = 1$ and $q = 1$, and we also initialize two integer arrays i, j of size n. For every $p \in [n]$, if row p of M_A is not zero, we let $i[q] = p$, and let $q = q + 1$. On the other hand, if row p of M_A is zero, we let $j[r] = p$, and let $r = r + 1$.

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

$$i[1] \mapsto 1; \ldots; i[k] \mapsto k; j[1] \mapsto k+1; j[2] \mapsto k+2; \ldots; j[n-k] \mapsto n,$$

From π we construct P, where P has 1s in positions:

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

and zeros everywhere else. The permutation matrix Q is constructed in a similar manner from τ. This can be clearly done in time and space proportional to $|M_A|$, i.e., in linear time and space.

Once we obtain $P_\pi A Q_\tau$, we work, recursively, with this matrix, starting at each level of the recursion in Step 2a, in order 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 $k = |M_A| \leq \min\{m, n\}$ many times. Note therefore that if there are many green 1s, the procedure has fewer recursive calls; if there are few green 1s, the procedure has more recursive calls.

We make this argument a little bit more precise; let $R(n)$ be the maximum number of steps, in the worst-case, that our procedure takes on a matrix of size n (let n denote here $\max\{m, n\}$). Let a "step" be a single "atomic operation" which for us is one of the following two: check what is the value in position (i, j) of some matrix, and change the value in position (i, j) of some matrix to 0 or 1.

Suppose that k is the number of black 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:

$$R(n+1) = \max_{1<k<n} \{k(n+1-k) + R(k)\}. \tag{2}$$

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; 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 (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 (2); it should really be $2k(n + 1 - k)$, but this 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 (2).

We now show by induction that $R(n) \leq n^2$, where R is initialized with $R(0) = R(1) = 1$. For $k < n + 1$, by inductive assumption $R(k) < k^2$, and so by (2):

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

Finally, we show how PERALG keeps track of the orientations $\boldsymbol{o}$ in each step of the procedure. As was pointed out following the presentation of PERALG, the situation, as represented in Figure 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. This means that some orientations are computed in a given step, and some are not: $\boldsymbol{o} = o_1 o_2 \dots o_k$, where $o_{i_1} o_{i_2} \dots o_{i_{k_1}}$ correspond to the black 1s and are not yet computed, and $o_{j_1} o_{j_2} \dots o_{j_{k_2}}$ correspond to the green 1s, and have just been computed. Note that the o_{i_p}'s and o_{j_q}'s are interspersed in $\boldsymbol{o}$, and $k = k_1 + k_2$. But it is easy to keep track, because the order is preserved: let $\boldsymbol{o}$ be a string of 0s, 1s, and unset values. The unset values always appear in the same order as the black 1s on the diagonal to be dealt with in the next step. □

From Lemma 3 and Lemma 4 we get the main result.

Theorem 3. *Given a bipartite graph $G = (V = V_1 \cup V_2, E)$, and a maximal matching M_G,* PERALG *runs in time and space $O(|A_G|)$ to compute a minimal cover C_G. That is,* PERALG *runs in linear time and space to compute a minimal cover from a maximal matching.*

5 Conclusion

PERALG is a matrix permutation based algorithm that runs in time $O(|V_1||V_2|)$ to transform a maximal matching of a bipartite graph into a minimal vertex cover. In particular, PERALG runs in linear time and space (as its input is a binary matrix of size $|V_1| \times |V_2|$), and using the properties in the famous König's Mini-Max theorem, it performs basic counting of zeros and ones to compute a minimal cover. Our algorithm is very simple, and does not employ the usual graph theoretic properties that are the foundation of classical algorithms in this area.

Our algorithm is one of many applications of König's Mini-Max theorem, which has also several equivalent formulations: as *Menger's Theorem*, counting disjoint paths; as *Hall's Theorem*, giving necessary and sufficient conditions for the existence of a "system of distinct representatives" of a collection of sets; as *Dilworth's Theorem*, counting the number of disjoint chains in a poset. It has been shown in [5] that these different formulations are equivalent, and the equivalence can be proven in weak fragments of arithmetic. A surveys of Mini-Max results can be found in [12].

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[2] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O\left(n^{1.5}\sqrt{\frac{m}{\log n}}\right)$. *Information Processing Letters*, 37:92–99, 1991.

[3] R. A. Brualdi and H. J. Ryser. *Combinatorial Matrix Theory*. Cambridge University Press, 1991.

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

[5] A. G. Fernández and M. Soltys. Feasible combinatorial matrix theory. In K. Chatterjee and J. Sgall, editors, *Mathematical Foundations of Computer Science 2013*, volume 8087 of *Lecture Notes in Computer Science*, pages 777–788. Springer, 2013.

[6] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Prinston Univ. Press, 1962.

[7] F. Gavril. Testing for equality between maximum matching and minimum node covering. *Information Processing Letters*, 6(6):199–202, 1977.

[8] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4), December 1973.

[9] 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.

[10] D. König. Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére. *Matematikai és Természettudományi Értesítö*, 34:104–119, 1916.

[11] D. König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4), 1916.

[12] L. Lovász and M. D. Plummer. Matching theory. In *Annals of Discrete Mathematics*. North-Holland, 1986.

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

[14] S. Mishra, V. Raman, S. Saurabh, S. Sikdar, and C. R. Subramnian. The Complexity of König Subgraph Problems and Above-Guarantee Vertex Cover. *Algorithmica*, 61(4):857–881, 2011.

[15] J. A. Storer. *An Introduction to Data Structures and Algorithms*. Progress in Computer Science and Applied Logic Series. Springer, 2001.