Complexity of Searching for 2×2 Submatrices in Boolean Matrices

Daniel Průša¹ and Michael Wehar²(✉)

¹ Czech Technical University, Prague, Czech Republic

daniel.prusa@fel.cvut.cz

² Swarthmore College, Swarthmore, PA, USA

mwehar1@swarthmore.edu

Abstract. We study the problem of finding a given 2×2 matrix as a submatrix of a given Boolean matrix. Three variants are considered: search for a matching submatrix of any area, of minimum area, or of maximum area. The problem relates to 2D pattern matching, and to fields such as data mining, where the search for submatrices plays an important role. Besides these connections, the problem itself is very natural and its investigation helps to demonstrate differences between search tasks in one-dimensional and multidimensional topologies.

Our results reveal that the problem variants are of different complexities. First, we show that given an $m \times n$ Boolean matrix, the any variant can be solved in $\tilde{O}(mn)$ time for any given 2×2 matrix, but requires various strategies for different 2×2 matrices. This contrasts with the complexity of the task over matrices with entries from the set $\{0, 1, 2\}$, where the problem is Triangle Finding-hard and hence no algorithm with similar running time is known for it. Then, we show that the minimization variant in the case of Boolean matrices can also be solved in $\tilde{O}(mn)$ time. Finally, in contrast, we prove Triangle Finding-hardness for the maximization variant and show that there is a rectangular matrix multiplication-based algorithm solving it in $O(mn(\min\{m, n\})^{0.5302})$ time.

Keywords: Boolean matrix · Submatrices · Two-dimensional pattern matching · Local picture language · Triangle Finding-hard problem · Fast matrix multiplication

1 Introduction

We study the complexity of Four Corner Problems. A Four Corner Problem is concerned with finding a given 2×2 matrix as a submatrix of a given Boolean matrix. By submatrix, we mean a matrix that is formed by restricting the original matrix to a subset of its rows and columns. For a matrix $\mathbf{B} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, where $a, b, c, d \in \{0, 1\}$, we consider three kinds of Four Corner Problems:

1. In the input Boolean matrix $\mathbf{M}$, search for any 2×2 submatrix of $\mathbf{M}$ that matches $\mathbf{B}$ (we abbreviate this task as ANY $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$)

2. Search for a submatrix that matches $\mathbf{B}$ and encloses the minimum area of $\mathbf{M}$ (abbreviated as $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$)
3. Search for a submatrix that matches $\mathbf{B}$ and encloses the maximum area of $\mathbf{M}$ (abbreviated as $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$).

In general, the problem of finding a specific submatrix in a larger matrix is of importance in several computer science disciplines. For example Boolean matrices, and their associated submatrices of 1's, play a central role in *data mining* problems such as *frequent itemset mining* [13]. Moreover, finding a submatrix of 1's in the adjacency matrix of a graph G corresponds to finding a biclique of G [13]. As the maximum edge biclique problem is NP-complete [10], the complexity of searching for a $k \times k$ submatrix is expected to grow as k grows. In this paper, we deal with the simplest case when $k = 2$. An example of its use is as follows. Given m respondents answering n yes/no questions in a questionnaire, are there two respondents who answered yes on two of the same questions?

The above tasks $\text{ANY} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ can also be viewed as two-dimensional pattern matching: we search for any/min/max rectangular block of a matrix that matches a given template. In only one dimension, similar pattern matching problems can be described using *regular languages* [2]. In this case, all the any/min/max tasks are solvable by a finite-state automaton-based algorithm in time linear in the input length [8]. In two dimensions, these problems are easily definable via the notion of *local picture languages* [5]. This is a formalism defining sets of two-dimensional arrays (so called *pictures*) for which the membership problem can be determined by looking at a window of size 2×2 . These picture languages are a straightforward generalization of the well known *local (string) languages* [12], which form a proper subset of the family of regular languages.

We introduced in [8] a general algorithm solving two-dimensional pattern matching against local picture languages in time $O(mn \min\{m, n\})$ for $m \times n$ input matrices. Further, for a specific local picture language, we investigated the pattern matching problem which is precisely $\text{ANY} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and showed it to be solvable in linear time in the input matrix area. Here our goal is to propose more efficient algorithms for a specialized subclass of local picture language pattern matching problems over Boolean matrices called Four Corner Problems. In particular, we show that the problem $\text{ANY} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is solvable in $\tilde{O}(mn)$ time for any $a, b, c, d \in \{0, 1\}$ (Theorem 1). This result is surprising because it was proven in [8] that searching for a submatrix matching $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ in an $n \times n$ matrix over $\{0, 1, 2\}$ is Triangle Finding-hard. In other words, the proof introduced a *fine-grained reduction* [15] from Triangle Finding to the search problem for $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ over $\{0, 1, 2\}$ suggesting that Four Corner Problems are harder over larger alphabets.

The Triangle Finding problem is to decide whether a given undirected graph $G = (V, E)$ is triangle-free or not. It is a classic algorithmic problem which can be reduced to Boolean Matrix Multiplication (see [6]) and solved in time $O(n^\omega)$, where $n = |V|$ and $\omega < 2.373$ denotes the matrix multiplication exponent [14]. However, it is currently unknown whether Triangle Finding can be solved in

time $\tilde{O}(n^2)$. Note that conditional lower bounds based on Triangle Finding are known for several problems (see, e.g., [1, 7, 9, 11]).

We further investigate the minimization and maximization variants of the search problem over Boolean matrices. For the min variant, we improve on Theorem 1 by showing that the problem $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is solvable in $\tilde{O}(mn)$ time for any $a, b, c, d \in \{0, 1\}$ (Theorem 4). For the max variant, we prove that $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is Triangle Finding-hard for any $a, b, c, d \in \{0, 1\}$ (Theorem 5). Also, we present an algorithm that solves $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ in $O(mn(\min\{m, n\})^{0.5302})$ time (Theorem 6). This algorithm is based on computing a *minimum witness* for Boolean matrix multiplication [4]. However, it is likely impractical because it uses a fast rectangular matrix multiplication algorithm that involves a large constant factor.

The paper is structured as follows. Section 2 establishes some required notions. Then, Sects. 3, 4 and 5 gradually present results for the problems $\text{ANY} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ and $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

2 Preliminaries

$\mathbb{N} = \{0, 1, 2, \dots\}$ is the set of natural numbers and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ is the set of positive integers. For functions $f, g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, we write $f(m, n) = \tilde{O}(g(m, n))$ if and only if there are numbers $p, q \in \mathbb{N}$ such that $f(m, n) = O(g(m, n) \log^p(m) \log^q(n))$.

Let $\mathbf{M}$ be an $m \times n$ Boolean matrix. We write $\overline{\mathbf{M}}$ to denote the matrix obtained from $\mathbf{M}$ by negating its entries (i.e., we have $\overline{\mathbf{M}}_{i,j} = 1 - \mathbf{M}_{i,j}$ for every entry). We consider that rows and columns of $\mathbf{M}$ are indexed from 1 to m and n , respectively. A $k \times \ell$ (rectangular) *block* of $\mathbf{M}$ at a position (r, c) is denoted as $B = \mathbf{M}[r, c; k, \ell]$, where $1 \leq k \leq m$, $1 \leq \ell \leq n$, $1 \leq r \leq m - k + 1$, $1 \leq c \leq n - \ell + 1$. Its entries coincide with the entries of the submatrix obtained from $\mathbf{M}$ by deleting rows $1, \dots, r - 1$ and $r + k, \dots, m$, and columns $1, \dots, c - 1$ and $c + \ell, \dots, n$. We use $B_{i,j}$ to refer to the entry in the i -th row and j -th column of B . We have $B_{i,j} = \mathbf{M}_{r+i-1, c+j-1}$. We define the *area* of B as $\mathbf{a}(B) = k\ell$, and the 2×2 *corners submatrix* of B as

$$\varkappa(B) = \begin{pmatrix} B_{1,1} & B_{1,\ell} \\ B_{k,1} & B_{k,\ell} \end{pmatrix}.$$

The set of all blocks of $\mathbf{M}$ is denoted by $\mathcal{B}_{\mathbf{M}}$.

For $a, b, c, d \in \{0, 1\}$, we define the following search problems (also known as Four Corner Problems) for an input Boolean matrix $\mathbf{M}$.

- $\text{ANY} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$: find $B \in \mathcal{B}_{\mathbf{M}}$ such that $\varkappa(B) = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$,
- $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$: find $B \in \arg \min_{B \in \mathcal{B}_{\mathbf{M}}} \{\mathbf{a}(B) \mid \varkappa(B) = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\}$,
- $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$: find $B \in \arg \max_{B \in \mathcal{B}_{\mathbf{M}}} \{\mathbf{a}(B) \mid \varkappa(B) = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\}$.

3 Searching for Any Matching Submatrix

This section presents algorithms for $\text{ANY} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ that run in nearly linear time in the input matrix area, for every a, b, c , and d . In some cases an efficient

algorithm is achieved by using properties of the minimum matching submatrix, so these algorithms also solve the corresponding $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ problem (see Lemmas 2 and 3).

Out of all ANY $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ problems, ANY $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and ANY $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ are easiest to solve. It has already been shown in [8] that ANY $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ reduces to finding a four-cycle in a bipartite graph. Here we give a more straightforward algorithm.

Lemma 1. ANY $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ is solvable in time $O(mn)$ for m by n Boolean matrices.

Proof. Let an $m \times n$ Boolean matrix $\mathbf{M}$ be given. Without loss of generality, suppose that $m \geq n$. The algorithm is as follows. We create a set S of pairs of column indexes. Initially, the set is empty. The matrix is traversed row by row. For each row i , we find the set C_i of all column indexes j such that $\mathbf{M}_{i,j} = 1$. Then, for every pair $\{c_1, c_2\} \in \binom{C_i}{2}$, we check whether $\{c_1, c_2\}$ is in S . If not, it is added to S . Otherwise, a desired submatrix has been found.

The algorithm takes $O(mn + n^2)$ time because it visits each entry from $\mathbf{M}$ at most once and it adds at most $\binom{n}{2}$ pairs of column indexes into S . Because $m \geq n$, the total runtime is $O(mn)$. $\square$

Lemma 2. $\text{MIN} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ is solvable in time $O(mn)$ for m by n Boolean matrices.

Proof. Let $\mathbf{M}$ be an $m \times n$ Boolean matrix. The algorithm is based on the following claim: If $\mathbf{M}$ contains a block $B = \mathbf{M}[r, c; k, \ell]$ such that $\varkappa(B) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, then it contains a block $B' = \mathbf{M}[r', c'; k', \ell']$ such that $\varkappa(B') = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, $B'_{i,1} = 0$ for all $i = 2, \dots, k' - 1$ and $B'_{k',j} = 0$ for all $j = 2, \dots, \ell' - 1$ (i.e., the left and bottom edge of B' , excluding the corners, contain only 0 entries).

To see this, suppose without loss of generality that $B_{i,1} = 1$ for some $1 < i < k$. Let $B_1 = \mathbf{M}[r, c; i, \ell]$ and $B_2 = \mathbf{M}[r + i - 1, c; k - i + 1, \ell]$. Then, either $\varkappa(B_1) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ (if $B_{i,\ell} = 1$) or $\varkappa(B_2) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ (if $B_{i,\ell} = 0$). Since B_1 and B_2 are proper subsets of B , we have found a smaller block containing $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ as a submatrix.

Now, we present the algorithm. It creates a map σ where a key is a pair (i, j) such that $\mathbf{M}_{i,j} = 1$. The value associated with (i, j) is a pair (i', j') such that i' is the largest row index less than i such that $\mathbf{M}_{i',j} = 1$ (i.e., i' is the row index of the nearest entry 1 located upwards from the position (i, j)) and j' is the smallest column index greater than j such that $\mathbf{M}_{i,j'} = 1$ (i.e., the column index of the nearest entry 1 rightwards). Note that the value of i' or j' might be undefined if there is no such row index or column index, respectively.

It is possible to build σ in $O(mn)$ time by making two passes over $\mathbf{M}$. The first pass is to compute the i' 's. The matrix $\mathbf{M}$ is scanned column by column. Each column index j is scanned from top to bottom. Whenever entry 1 is detected at a position (i', j) , then i' is the first component of $\sigma(i, j)$ for the next detected entry 1 from position (i, j) . Analogously, the second pass, scanning $\mathbf{M}$ row by row, is to compute the j' 's.

Now, for each key (i, j) in the map σ , the algorithm takes its value (i', j') and checks if rows i, i' and columns j, j' form a desired submatrix matching $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$. By doing this, every existing block with 0 entries on the left and bottom edges is

checked. Among these blocks, a minimum-area block B such that $\varkappa(B) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ is returned as the result.

Assuming constant time map operations, the algorithm runs in $O(mn)$ time (note that the map σ can be implemented by using an $m \times n$ array). $\square$

Lemma 3. $\text{MIN} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is solvable in time $\tilde{O}(mn)$ for m by n Boolean matrices.

Proof. Case I (square matrices): Let an $n \times n$ Boolean matrix $\mathbf{M}$ be given. We present a divide and conquer strategy. If $t(n)$ denotes the runtime of searching for a desired minimum submatrix in an $n \times n$ matrix, then we show that

$$t(n) = 4 \cdot t\left(\frac{n}{2}\right) + O(n^2). \quad (1)$$

To accomplish this, we split $\mathbf{M}$ horizontally into matrices $\mathbf{M}_{\text{top}}$ and $\mathbf{M}_{\text{bottom}}$, where $\mathbf{M}_{\text{top}}$ is $\lceil \frac{n}{2} \rceil$ by n and $\mathbf{M}_{\text{bottom}}$ is $\lfloor \frac{n}{2} \rfloor$ by n . Next, we split $\mathbf{M}_{\text{top}}$ vertically into $\mathbf{M}_{\text{top, left}}$, which is $\lceil \frac{n}{2} \rceil$ by $\lceil \frac{n}{2} \rceil$, and $\mathbf{M}_{\text{top, right}}$, which is $\lceil \frac{n}{2} \rceil$ by $\lfloor \frac{n}{2} \rfloor$. Quite analogously, we split $\mathbf{M}_{\text{bottom}}$ vertically into $\mathbf{M}_{\text{bottom, left}}$ and $\mathbf{M}_{\text{bottom, right}}$. A desired minimum submatrix is either in one of the four $\frac{n}{2}$ by $\frac{n}{2}$ matrices or it spans the border between either $\mathbf{M}_{\text{top}}$ and $\mathbf{M}_{\text{bottom}}$, $\mathbf{M}_{\text{top, left}}$ and $\mathbf{M}_{\text{top, right}}$, or, $\mathbf{M}_{\text{bottom, left}}$ and $\mathbf{M}_{\text{bottom, right}}$. We propose a procedure running in $O(n^2)$ time that finds a desired minimum submatrix by the assumption that there is such a submatrix crossing the specified borders. Therefore, in $O(n^2)$ time, we reduce finding a desired submatrix in an n by n matrix $\mathbf{M}$ to finding a desired submatrix in one of four $\frac{n}{2}$ by $\frac{n}{2}$ matrices. If we solve recurrence (1), then we get $t(n) = O(n^2 \log(n))$ [3].

Without loss of generality, let us deal only with the border between $\mathbf{M}_{\text{top}}$ and $\mathbf{M}_{\text{bottom}}$. We claim: if $B = \mathbf{M}[r, c; k, \ell]$ is a minimum-area block of $\mathbf{M}$ such that $\varkappa(B) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, then $B_{i,1} = B_{i,\ell}$ for all $i = 2, \dots, k-1$. Indeed, $B_{i,1} \neq B_{i,\ell}$ would clearly contradict the minimality of B .

Based on the claim, we create maps σ_{top} and σ_{bottom} such that $\sigma_{\text{top}}(\{i, j\})$ is the largest row index such that columns i and j differ in $\mathbf{M}_{\text{top}}$, and, analogously, $\sigma_{\text{bottom}}(\{i, j\})$ is the smallest row index such that columns i and j differ in $\mathbf{M}_{\text{bottom}}$. Once we have constructed σ_{top} and σ_{bottom} we go through each pair of column indexes $\{i, j\}$ and check if rows $\sigma_{\text{top}}(\{i, j\})$, $\sigma_{\text{bottom}}(\{i, j\})$ and columns i, j together create a desired submatrix of $\mathbf{M}$. A minimum submatrix among the detected submatrices is the candidate for the resulting submatrix returned by the procedure.

It remains to explain how we obtain the maps. Let us first give a construction for σ_{top} . We create a set X of pairwise disjoint sets of column indexes. Initially, X contains one set containing all column indexes. We repeat the following process for each row of $\mathbf{M}_{\text{top}}$, starting at the bottommost one and proceeding upwards: Create two disjoint sets A_0 and A_1 where A_0 contains all column indexes that are 0's and A_1 contains all column indexes that are 1's in the current row. For each set S in X , split S into two disjoint subsets $S_0 = S \cap A_0$ and $S_1 = S \cap A_1$. For every $\{i, j\}$ such that $i \in S_0$ and $j \in S_1$, set $\sigma_{\text{top}}(\{i, j\})$ to the current row index. Then, update X by replacing S with S_0 and S_1 . Throw out any sets from

X that have less than two elements. Finish when X is empty or every row of $\mathbf{M}_{\text{top}}$ has been processed.

We similarly build σ_{bottom} , but we start at the top row of $\mathbf{M}_{\text{bottom}}$ going one row down at a time. It only takes $O(n^2)$ time to construct σ_{top} and σ_{bottom} because we do $O(n)$ work per row plus an additional constant amount of work for each pair of columns.

Case II (rectangular matrices): Let an $m \times n$ Boolean matrix $\mathbf{M}$ be given. Assume without loss of generality that $m > n$.

We perform horizontal splits to divide $\mathbf{M}$ into $d = \lceil \frac{m}{n} \rceil$ smaller matrices $\{\mathbf{M}_k\}_{k \in [d]}$ such that for each $k \in [d-1]$, $\mathbf{M}_k$ is n by n , and $\mathbf{M}_d$ is c by n for some $c \leq n$. A desired minimum submatrix is either in $\mathbf{M}_k$ for some $k \in [d]$ or it crosses the border between $\mathbf{M}_k$ and $\mathbf{M}_{k+1}$ for some $k \in [d-1]$. Then, the former cases in total take $O(\frac{m}{n} \cdot t(n))$ time. We claim that the latter cases take $O(mn)$ time. For each $k \in [d-1]$, we construct maps $\sigma_{k,\text{top}}$ and $\sigma_{k,\text{bottom}}$ such that $\sigma_{k,\text{top}}(\{i, j\})$ is the smallest row index such that columns i and j differ in $\mathbf{M}_k$, and $\sigma_{k,\text{bottom}}(\{i, j\})$ is the largest row index such that columns i and j differ in $\mathbf{M}_k$. Following the same approach as for the square matrix case, we can construct all maps in total time $O(mn)$. Then, for each pair of column indexes i, j we have up to d cases to check. This results in total time $O(mn)$. Note that if a map is not defined at $\{i, j\}$, then we try the next map and combine the cases together since this means a submatrix might span across multiple horizontal splits. In total, our algorithm takes $O(\frac{m}{n} \cdot t(n) + mn) = O(mn \log(n))$ time. $\square$

Lemma 4. ANY $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ is solvable in time $\tilde{O}(mn)$ for m by n Boolean matrices.

Proof. Let an $m \times n$ Boolean matrix $\mathbf{M}$ be given.

Case I (tall matrices): We consider the case when $m \geq n$. We proceed in a similar manner as in the proof of Lemma 1. We create a set S of pairs of column indexes. Initially, the set is empty. The matrix is traversed row by row. For each row, we do the following. We create a set R . Initially, R is empty, but we will add column indexes to R . We scan entries from left to right in the row. When we encounter a 1 entry at column index i , we add i to R . When we encounter a 0 entry at column index j , we go through each column index i from R . If (i, j) is in S , then we found a desired submatrix. Otherwise, we add (i, j) to S . Since $m \geq n$, this takes $O(mn + n^2) = O(mn)$ time.

Case II (short matrices): We consider the case when $m < n$. We perform vertical splits to divide $\mathbf{M}$ into $d = \lceil \frac{n}{m} \rceil$ smaller matrices $\{\mathbf{M}_k\}_{k \in [d]}$ such that for each $k \in [d-1]$, $\mathbf{M}_k$ is m by m , and $\mathbf{M}_d$ is m by c for some $c \leq m$. The matrix $\mathbf{M}$ contains a desired submatrix if and only if some $\mathbf{M}_k$ contains a minimal submatrix for some $k \in [d]$ or there is a minimal submatrix that crosses the border between $\mathbf{M}_k$ and $\mathbf{M}_{k+1}$ for some $k \in [d-1]$.

Consider the former condition. Checking if a given $\mathbf{M}_k$ matrix contains a minimal desired submatrix takes $O(m^2)$ time by applying the approach from the first case. Checking all of the matrices in $\{\mathbf{M}_k\}_{k \in [d]}$ takes $O(d \cdot m^2) = O(\frac{n}{m} \cdot m^2) = O(mn)$ time.

Now, we focus on checking the latter condition. For each $k \in [d - 1]$, we construct maps $\sigma_{k,\text{left}}$ and $\sigma_{k,\text{right}}$ such that $\sigma_{k,\text{left}}(\{i, j\})$ is the smallest column index such that rows i and j are equal in $\mathbf{M}_k$, and $\sigma_{k,\text{right}}(\{i, j\})$ is the largest column index such that rows i and j are equal in $\mathbf{M}_k$. Once we have constructed these maps, we consider each pair of rows i and j . We have up to $d - 1$ cases to check where each case considers the border between $\mathbf{M}_k$ and $\mathbf{M}_{k+1}$ for some $k \in [d - 1]$. We check each case by seeing if rows i and j along with columns $\sigma_{k,\text{right}}(\{i, j\})$ and $\sigma_{k+1,\text{left}}(\{i, j\})$ form a desired submatrix. It is sufficient to check these submatrices because we are only concerned with desired submatrices crossing the border that are minimal. Note if a map is not defined at $\{i, j\}$, then we try the next map and combine the cases together since this means a submatrix might span across multiple vertical splits. This takes $O(d \cdot m^2) = O(\frac{n}{m} \cdot m^2) = O(mn)$ time. It remains to describe how the maps are constructed. We claim that the maps can be constructed in $O(mn \log(m))$ time. Therefore, the total runtime is $O(mn \log(m))$.

Given $k \in [d - 1]$, we describe how to construct $\sigma_{k,\text{left}}$ for the matrix $\mathbf{M}_k$. For each $\ell \in [\log(m)]$, we construct a matrix $\mathbf{M}_{k,\ell}$. The matrix $\mathbf{M}_{k,\ell}$ is obtained from $\mathbf{M}_k$ by negating all bits in each row i such that i 's binary expansion has a 1 at position ℓ . Next, in a similar manner as described in the proof of Lemma 3, we construct a map σ_ℓ such that $\sigma_\ell(\{i, j\})$ is the smallest column index where rows i and j differ in $\mathbf{M}_{k,\ell}$. Now, we use these $\log(m)$ maps to construct $\sigma_{k,\text{left}}$. For each pair of rows i and j , there is some position ℓ in i and j 's binary expansions where they differ. The smallest column index where rows i and j differ in $\mathbf{M}_{k,\ell}$ is exactly the same as the smallest column index where rows i and j are equal in $\mathbf{M}_k$. Hence, we make $\sigma_{k,\text{left}}(\{i, j\}) = \sigma_\ell(\{i, j\})$. It takes $O(m^2 \log(m))$ time to construct $\sigma_{k,\text{left}}$. The map $\sigma_{k,\text{right}}$ can be constructed in a similar manner. In total, it takes $O(d \cdot m^2 \log(m)) = O(\frac{n}{m} \cdot m^2 \log(m)) = O(mn \log(m))$ time to construct all of the maps. $\square$

Theorem 1. *Problem ANY $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is solvable in time $\tilde{O}(mn)$ for m by n Boolean matrices and any $a, b, c, d \in \{0, 1\}$.*

Proof. Consider the set of matrices $S = \{(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}), (\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}), (\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}), (\begin{smallmatrix} 1 & 0 \\ 1 & 0 \end{smallmatrix})\}$. Every 2×2 Boolean matrix $\mathbf{A}$ is similar to a Boolean matrix $\mathbf{B} \in S$ in the sense that $\mathbf{B} = U(\mathbf{A})$ for an operation U that combines a rotation with an optional negation of all bits. Further, for every Boolean matrix $\mathbf{M}$, the matrix $\mathbf{M}$ contains $\mathbf{A}$ as a submatrix if and only if $U(\mathbf{M})$ contains $\mathbf{B}$ as a submatrix. Applying Lemmas 1, 2, 3, and 4, we can determine if $\mathbf{M}$ has $\mathbf{A}$ as a submatrix in $\tilde{O}(mn)$ time. $\square$

4 Searching for a Minimum 2-by-2 Submatrix of 1's

In the previous section, we presented fast algorithms for minimization problems $\text{MIN} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $\text{MIN} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Here, we use preceding results for $\text{ANY} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $\text{ANY} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ to also obtain fast algorithms for $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $\text{MIN} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$.

First, we introduce an algorithm for $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. The technique we apply requires several preparatory steps: a characterization of Boolean matrices that do not have $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ as a submatrix (Lemma 5), an algorithm solving $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ whose complexity depends on the number of pairs of 1's within the same rows (Lemma 6), and a fast algorithm solving $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ approximately (Lemma 8). Then, we can apply a similar approach to solve $\text{MIN} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$.

Lemma 5. *Let $\mathbf{A}$ be an m by n Boolean matrix. Let a_i denote the number of 1's in the i -th row of $\mathbf{A}$. If $\sum_{i=0}^m \binom{a_i}{2} > \binom{n}{2}$, then $\mathbf{A}$ must contain a block whose corners are 1's.*

Proof. $\sum_{i=0}^m \binom{a_i}{2}$ is the size of the set $T = \{(i, \{j, k\}) \mid j \neq k \wedge \mathbf{A}_{ij} = \mathbf{A}_{ik} = 1\}$. If $|T| > \binom{n}{2}$, then there are $(i_1, \{j, k\}), (i_2, \{j, k\}) \in T$, where $i_1 \neq i_2$. This means that rows i_1, i_2 and columns j, k form a submatrix $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. $\square$

Lemma 6. *Let $\mathbf{M}$ be an m by n Boolean matrix and $T(\mathbf{M}) = \{(i, \{j, k\}) \mid j \neq k \wedge \mathbf{M}_{i,j} = \mathbf{M}_{i,k} = 1\}$. There is an algorithm solving $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ in $O(|T(\mathbf{M})| + mn)$ time.*

Proof. The algorithm uses a map σ with keys $\{j, k\}$, where $j \neq k$ are column indexes. The value of $\sigma(\{j, k\})$ is a row index. Initially, the map is empty.

The input Boolean matrix $\mathbf{M}$ is processed row by row. In the i -th row, the following actions are performed for each $(i, \{j, k\}) \in T(\mathbf{M})$. First, it is checked whether $\sigma(\{j, k\})$ is defined. If it is not, then $\sigma(\{j, k\})$ is set to i . Otherwise, the algorithm finds out whether the rectangle formed by rows $i, \sigma(\{j, k\})$ and columns j, k is the minimum one so far. Then, $\sigma(\{j, k\})$ is updated to be i . $\square$

For convenience, for each Boolean matrix $\mathbf{M}$ considered now until the end of this section, assume that the number of rows and the number of columns of $\mathbf{M}$ are powers of 2. Since any matrix of a general size $m \times n$ can be extended to a $2^{\lceil \log_2 m \rceil} \times 2^{\lceil \log_2 n \rceil}$ matrix (with the added entries set to “undefined” value), the assumption will not have any impact on the generality and asymptotic time complexity of the presented algorithms.

For $p \in \mathbb{N}^+$, let $\mathcal{S}(p) = \{2^i \mid i = 1, 2, \dots, \lceil \log_2 p \rceil\}$ be the set of powers of two greater than 1 and not greater than p . Let $\mathbf{M}$ be an $m \times n$ Boolean matrix. For $k \in \mathcal{S}(m)$ and $\ell \in \mathcal{S}(n)$, let $\mathcal{R}_{\mathbf{M}}(k, \ell)$ denote the set of all $k \times \ell$ blocks of $\mathbf{M}$ whose top left corner is located in $\mathbf{M}$ at a position $(1 + a \cdot \frac{k}{2}, 1 + b \cdot \frac{\ell}{2})$ for some $a, b \in \mathbb{N}$. Let $\mathcal{R}_{\mathbf{M}} = \bigcup_{k \in \mathcal{S}(m), \ell \in \mathcal{S}(n)} \mathcal{R}_{\mathbf{M}}(k, \ell)$.

Lemma 7. *Let B be a p by q block of $\mathbf{M}$. There are powers of 2, denoted by k and ℓ , such that $k < 4p$, $\ell < 4q$ and B is included in a block from $\mathcal{R}_{\mathbf{M}}(k, \ell)$.*

Proof. Assume $B = \mathbf{M}[r, c; p, q]$. Let $k = \min\{m, 2 \cdot 2^{\lceil \log_2 p \rceil}\}$, $\ell = \min\{n, 2 \cdot 2^{\lceil \log_2 q \rceil}\}$,

$$a = \max\{x \mid x \in \mathbb{N} \wedge x \cdot \frac{k}{2} + k \leq m \wedge 1 + x \cdot \frac{k}{2} \leq r\}, \text{ and} \quad (2)$$

$$b = \max\{y \mid y \in \mathbb{N} \wedge y \cdot \frac{\ell}{2} + \ell \leq n \wedge 1 + y \cdot \frac{\ell}{2} \leq c\}. \quad (3)$$

Then, B is included in the block $\mathbf{M}[1 + a \cdot \frac{k}{2}, 1 + b \cdot \frac{\ell}{2}; k, \ell] \in \mathcal{R}_{\mathbf{M}}(k, \ell)$. This is proved as follows. The definition of a ensures that $1 + a \cdot \frac{k}{2} \leq r$. It is also needed to verify that

$$r + p - 1 \leq a \cdot \frac{k}{2} + k. \quad (4)$$

Observe that this inequality is trivially fulfilled when $a \cdot \frac{k}{2} + k = m$. Hence, assume that

$$a \cdot \frac{k}{2} + k < m. \quad (5)$$

Since m is divisible by $\frac{k}{2}$, inequality (5) implies that

$$(a + 1) \cdot \frac{k}{2} + k \leq m. \quad (6)$$

It must thus hold that

$$1 + (a + 1) \cdot \frac{k}{2} > r \quad (7)$$

(otherwise the right-hand side of (2) is greater than a). Now, it suffices to combine (7) and $p \leq 2^{\lceil \log_2 p \rceil} = \frac{k}{2}$ to obtain inequality (4).

Quite analogously, the definition of b ensures that $1 + b \cdot \frac{\ell}{2} \leq c$ and it can be proved that $c + q - 1 \leq b \cdot \frac{\ell}{2} + \ell$. $\square$

Lemma 7 and the defined set of blocks $\mathcal{R}_{\mathbf{M}}$ provide a basis for designing a fast algorithm that solves $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ approximately.

Lemma 8. *There is an algorithm that, for any m by n Boolean matrix $\mathbf{M}$, finds in $O(mn \log m \log n)$ time a block B of $\mathbf{M}$ such that $\varkappa(B) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ and $\text{a}(B) < 16 \cdot \text{a}(B_{\min})$, where $B_{\min}$ is a minimum-area block of $\mathbf{M}$ fulfilling $\varkappa(B_{\min}) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.*

Proof. The algorithm works as follows. For each block $B \in \mathcal{R}_{\mathbf{M}}$, it uses the algorithm of Lemma 1 to search inside B for a submatrix matching $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. Among all the detected submatrices, it outputs a minimal one.

By Lemma 7, $B_{\min}$ is a part of a block $B' \in \mathcal{R}_{\mathbf{M}}$ whose area is less than 16 times the area of $B_{\min}$, hence the algorithm of Lemma 1 running on B' finds a block of $\mathbf{M}$ fulfilling the lemma requirement.

For each $(k, \ell) \in \mathcal{S}(m) \times \mathcal{S}(n)$, the sum of the areas of the blocks in $\mathcal{R}_{\mathbf{M}}(k, \ell)$ is $O(mn)$, hence all these blocks are processed by the algorithm of Lemma 1 cumulatively in $O(mn)$ time. Finally, since $|\mathcal{S}(m) \times \mathcal{S}(n)| = O(\log m \log n)$, the proposed algorithm runs in $O(mn \log m \log n)$ time. $\square$

Theorem 2. $\text{MIN} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ is solvable in time $\tilde{O}(mn)$ for m by n Boolean matrices.

Proof. Let $\mathbf{M}$ be an input $m \times n$ Boolean matrix. Assume that the algorithm of Lemma 8 finds in $\mathbf{M}$ a block of an area S . The minimum area of a block of $\mathbf{M}$ containing $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ as a submatrix is in the range $(\frac{S}{16}, S]$.

Let us identify a suitable subset of blocks in $\mathcal{R}_{\mathbf{M}}$ such that each minimum-area block containing $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ as a submatrix is a part of a block from the subset. Define

$$P = \left\{ (k, \ell) \mid k \in \mathcal{S}(m) \wedge \ell = 2^{\lceil \log_2 \frac{16 \cdot S}{k} \rceil - 1} \wedge \ell \leq n \right\}, \text{ and} \\ \mathcal{R}'_{\mathbf{M}} = \bigcup_{(k, \ell) \in P} \mathcal{R}_{\mathbf{M}}(k, \ell).$$

The subset of blocks $\mathcal{R}'_{\mathbf{M}}$ satisfies the following properties.

Claim I: Every $k \times \ell$ block B in $\mathcal{R}'_{\mathbf{M}}$ is of area less than $16 \cdot S$. Indeed, we can derive

$$a(B) = k\ell = k \cdot 2^{\lceil \log_2 \frac{16 \cdot S}{k} \rceil - 1} < k \cdot 2^{\log_2 \frac{16 \cdot S}{k}} = 16 \cdot S.$$

Claim II: Every $p \times q$ block B of $\mathbf{M}$ such that $a(B) \leq S$ is a subset of a block in $\mathcal{R}'_{\mathbf{M}}$. This is proved as follows. By Lemma 7, there is a $k \times \ell$ block B_1 such that $4p > k \in \mathcal{S}(m)$, $4q > \ell \in \mathcal{S}(n)$, and B is included in B_1 . It holds that $k\ell < 16 \cdot pq \leq 16 \cdot S$, and hence $\ell < 2^{\lceil \log_2 \frac{16 \cdot S}{k} \rceil}$. Since ℓ is a power of 2, we can write $\ell \leq 2^{\lceil \log_2 \frac{16 \cdot S}{k} \rceil - 1} = \ell'$, which implies that there is a block $B_2 \in \mathcal{R}_{\mathbf{M}}(k, \ell') \subseteq \mathcal{R}'_{\mathbf{M}}$ that includes B_1 as well as B .

Claim III: Every $k \times \ell$ block B in $\mathcal{R}'_{\mathbf{M}}$ fulfills $\min\{|T(B)|, |T(B^T)|\} = O((\min\{k, \ell\})^2)$ (see Lemma 6 for the definition of $T(B)$). To show this, assume without loss of generality that $k \geq \ell$ and $k \geq 256$. Consider B to be split horizontally into 256 subblocks B_i of size $\frac{k}{256} \times \ell$. Hence, $a(B_i) = \frac{k\ell}{256}$. By Claim I, it holds that $k\ell < 16 \cdot S$, and hence $a(B_i) < \frac{16 \cdot S}{256} = \frac{S}{16}$. This means that B_i does not contain $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ as a submatrix, and hence Lemma 5 implies that $|T(B_i)| = O(\ell^2)$. Finally, we derive $|T(B)| = \sum_{i=1}^{256} |T(B_i)| = O(\ell^2)$.

Algorithm: We now have all prerequisites for describing the intended algorithm and deriving its time complexity. It works as follows. Call the algorithm of Lemma 8 to obtain S . For each $B \in \mathcal{R}'_{\mathbf{M}}$ of a size $k \times \ell$, call the algorithm of Lemma 6 either for B (if $k \geq \ell$) or B^T (if $k < \ell$) to find a minimum-area block within B containing $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ as a submatrix in time $O((\min\{k, \ell\})^2)$. A minimum-area block among all found blocks is returned as the final output.

For an ordered pair $(k, \ell) \in P$, the blocks of $\mathcal{R}_{\mathbf{M}}(k, \ell)$ are processed by the algorithm of Lemma 6 in cumulative time

$$O\left(\frac{mn}{k\ell} \cdot ((\min\{k, \ell\})^2 + k\ell)\right) = O(mn).$$

Hence, assuming without loss of generality that $m \leq n$, all the blocks of $\mathcal{R}'_{\mathbf{M}}$ are processed in total time

$$O\left(\sum_{(k, \ell) \in P} mn\right) = O(mn \log m).$$

Since the other stages of the algorithm run in $\tilde{O}(mn)$ time, the stated time complexity has been proven. $\square$

Theorem 3. $\text{MIN} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ is solvable in time $\tilde{O}(mn)$ for m by n Boolean matrices.

Theorem 4. $\text{MIN} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is solvable in time $\tilde{O}(mn)$ for m by n Boolean matrices and any $a, b, c, d \in \{0, 1\}$.

5 Searching for a Maximum Matching Submatrix

We first prove that the problem $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is Triangle Finding-hard for any $a, b, c, d \in \{0, 1\}$ (Theorem 5). Then, we show how $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ can be solved using rectangular matrix multiplication (Theorem 6).

Theorem 5. $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is Triangle Finding-hard for any $a, b, c, d \in \{0, 1\}$.

Proof. By the same reasoning given in the proof of Theorem 1, it suffices to prove Triangle Finding-hardness for problems $\text{MAX} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\text{MAX} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, $\text{MAX} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$, and $\text{MAX} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$. We first present a fine-grained reduction from Triangle Finding to $\text{MAX} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. We then adapt the reduction to the other three problems.

Let a graph $G = (V, E)$ be given. Let the set of vertices be $V = \{v_i \mid i \in \{1, \dots, n\}\}$. Let $\mathbf{A}$ be an $n \times n$ lower triangular Boolean matrix derived from the adjacency matrix of G as follows: $\mathbf{A}_{i,j} = 1$ if and only if $i > j$ and $\{v_i, v_j\} \in E$. Observe that $\{v_i, v_j, v_k\}$, where $i < j < k$, is a triangle in G if and only if $\mathbf{A}_{j,i} = \mathbf{A}_{k,i} = \mathbf{A}_{k,j} = 1$.

Define the matrix

$$\mathbf{M}_1 = \begin{pmatrix} \mathbf{A}_1 & \mathbf{O} & \mathbf{I} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{A}_2 & \mathbf{O} & \mathbf{A}_3 \end{pmatrix}$$

where $\mathbf{O}$ is the $n \times n$ zero matrix, $\mathbf{I}$ is the $n \times n$ identity matrix, and $\mathbf{A}_1 = \mathbf{A}_2 = \mathbf{A}_3 = \mathbf{A}$. An example of a graph G and the induced matrices $\mathbf{A}$ and $\mathbf{M}_1$ is given in Fig. 1.

Triangle Finding-hardness of $\text{MAX} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ is implied by the following property.

Claim: G has a triangle if and only if there is a block B of $\mathbf{M}$ such that $\varkappa(B) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ and $\text{a}(B) \geq 3n^2$.

To prove this, let us investigate which blocks can exist in $\mathbf{M}_1$ where all four corners are 1's. It is easy to see that there are three types of such blocks:

- A block B included in one of the matrices $\mathbf{A}_i$, $i \in \{1, 2, 3\}$. Its area is not greater than n^2 .
- A block B with two corners in $\mathbf{A}_2$ and the other two corners in either $\mathbf{A}_1$ or $\mathbf{A}_3$. Assume e.g. that the leftmost column of such a block is the k -th column of $\mathbf{M}_1$ and the rightmost column is in the ℓ -th column of $\mathbf{M}_1$, where $\ell > 2n$. The height of B is at most $n - k$ and it holds that $\ell < 3n$. Hence, $\text{a}(B)$ is upper bounded by $(\ell - k + 1)(n - k) \leq (3n - k)(n - k) < 3n^2$.
- A block B that has one corner in each of the matrices $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{I}$. Let the top left corner of B be in $\mathbf{M}_1$ at a position (k, ℓ) , and the bottom right corner of B be at a position (s, t) . Properties of the $\mathbf{A}_i$'s and $\mathbf{I}$ ensure that $|\ell - k| < n$, $t = 2n + k$, and $s > 2n + \ell$. Hence, $\text{a}(B) = (s - k + 1)(t - \ell + 1) > (2n + \ell - k)(2n + k - \ell) = 4n^2 - (\ell - k)^2 \geq 3n^2$.

Fig. 1. (a) A sample graph G on the set of vertices $\{1, 2, 3, 4\}$ where the subset $T = \{1, 2, 4\}$ forms a triangle. (b) The lower triangular adjacency matrix of G . The framed entries indicate the existence of the triangle T . (c) Triangle Finding in G reduced to $\text{MAX}_{\begin{bmatrix} 1 & 1 \end{bmatrix}}$.

It is not difficult to verify that there is a one-to-one correspondence between blocks of the third type and triples $i < j < k$ such that $\mathbf{A}_{j,i} = \mathbf{A}_{k,i} = \mathbf{A}_{k,j} = 1$, hence representing triangles $\{v_i, v_j, v_k\}$ of G .

To reduce Triangle Finding to $\text{MAX}_{\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}}$, $\text{MAX}_{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}$ and $\text{MAX}_{\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}}$, define the matrices $\mathbf{M}_2$, $\mathbf{M}_3$ and $\mathbf{M}_4$, respectively, as

$$\mathbf{M}_2 = \begin{pmatrix} \mathbf{A}_1 & \mathbf{O} & \bar{\mathbf{I}} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{A}_2 & \mathbf{O} & \mathbf{A}_3 \end{pmatrix}, \quad \mathbf{M}_3 = \begin{pmatrix} \mathbf{A}_1 & \mathbf{O} & \bar{\mathbf{I}} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \bar{\mathbf{A}}_2 & \mathbf{O} & \mathbf{A}_3 \end{pmatrix}, \quad \mathbf{M}_4 = \begin{pmatrix} \mathbf{A}_1 & \bar{\mathbf{O}} & \bar{\mathbf{I}} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{A}_2 & \mathbf{O} & \bar{\mathbf{A}}_3 \end{pmatrix}.$$

Recall that $\bar{\mathbf{C}}$ denotes the matrix created from a Boolean matrix $\mathbf{C}$ by negating its entries. Figure 2 shows the matrices $\mathbf{M}_3$ and $\mathbf{M}_4$ constructed for the graph G from Fig. 1.

One can again verify that G has a triangle if and only if each of the constructed matrices contains a block B such that $\mathbf{a}(B) \geq 3n^2$ and $\mathbf{x}(B)$ matches the desired 2×2 matrix. $\square$

Now, let us focus on approaches for solving the maximization problems. Given an $m \times n$ Boolean matrix $\mathbf{M}$ and $p, q \in \{0, 1\}$, let $\sigma_{p,q}^{\mathbf{M}}$ denote the map whose keys are pairs (i, j) , where i, j are row indexes of $\mathbf{M}$ such that $i < j$. For a key (i, j) , the map value is defined as the smallest column index c such that $\mathbf{M}_{i,c} = p$ and $\mathbf{M}_{j,c} = q$.

Let $\mathbf{M}'$ denote the matrix $\mathbf{M}$ flipped left to right. It is easy to see that every problem $\text{MAX}_{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}$, where $a, b, c, d \in \{0, 1\}$, can be solved based on the maps $\sigma_{a,c}^{\mathbf{M}}$ and $\sigma_{b,d}^{\mathbf{M}'}$. Conversely, this also shows that these maps are Triangle Finding-hard to build.

0 0 0 0	0 0 0 0	0 1 1 1	0 0 0 0	1 1 1 1	0 1 1 1
<u>1</u> 0 0 0	0 0 0 0	1 <u>0</u> 1 1	<u>1</u> 0 0 0	1 1 1 1	1 <u>0</u> 1 1
0 0 0 0	0 0 0 0	1 1 0 1	0 0 0 0	1 1 1 1	1 1 0 1
1 1 1 0	0 0 0 0	1 1 1 0	1 1 1 0	1 1 1 1	1 1 1 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1
0 1 1 1	0 0 0 0	1 0 0 0	0 0 0 0	0 0 0 0	0 1 1 1
1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1
<u>0</u> 0 0 1	0 0 0 0	1 <u>1</u> 1 0	<u>1</u> 1 1 0	0 0 0 0	0 <u>0</u> 0 1

(a)
(b)

Fig. 2. Triangle Finding reduced to (a) $\text{MAX} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and (b) $\text{MAX} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$, respectively.

The maps can be computed based on a minimum witness for Boolean matrix multiplication [4], and hence the time complexity of solving $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ in this way coincides with the time complexity in [4] for the minimum witness problem.

Lemma 9. *There is an algorithm that, for any m by n Boolean matrix $\mathbf{M}$ where $m \leq n$, and any $p, q \in \{0, 1\}$, builds $\sigma_{p,q}^{\mathbf{M}}$ in time $O(mn \cdot m^{0.5302})$.*

Theorem 6. *For any $a, b, c, d \in \{0, 1\}$, there is an algorithm solving $\text{MAX} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ in time $O(mn \cdot (\min\{m, n\})^{0.5302})$ for m by n Boolean matrices.*

6 Conclusion

We investigated the complexity of Four Corner Problems over Boolean matrices. A Four Corner Problem is concerned with searching for a given 2×2 submatrix in a given Boolean matrix. We demonstrated that minimum-area Four Corner Problems over Boolean matrices are solvable in nearly linear time in the input matrix area (Theorem 4) and maximum-area Four Corner Problems over Boolean matrices are Triangle Finding-hard (Theorem 5). The algorithms that we presented for the former problems might lead to efficient implementations, while the results achieved for the latter problems give rise to an interesting unresolved theoretical question: Are the maximum-area Four Corner Problems harder than the Triangle Finding problem? Going further, we suggest that a possible future direction is to investigate the complexity of Four Corner Problems over matrices with entries from larger alphabets.

Acknowledgment. We greatly appreciate all of the help and suggestions that we received. We would especially like to thank Joseph Swernofsky for helping us obtain some preliminary results, for contributing to a preliminary version of this work, and for providing valuable feedback. We also thank the Czech Science Foundation for supporting the first author (grant no. 19-09967S).

References

1. Abboud, A., Backurs, A., Williams, V.V.: If the current clique algorithms are optimal, so is Valiant's parser. In: Guruswami, V. (ed.) IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015), pp. 98–117. IEEE Computer Society (2015)
2. Aho, A.V.: Algorithms for finding patterns in strings. In: van Leeuwen, J. (ed.) Algorithms and Complexity, Handbook of Theoretical Computer Science, vol. A, pp. 255–300. The MIT Press, Cambridge (1990)
3. Bentley, J.L., Haken, D., Saxe, J.B.: A general method for solving divide-and-conquer recurrences. *SIGACT News* **12**(3), 36–44 (1980)
4. Cohen, K., Yuster, R.: On minimum witnesses for boolean matrix multiplication. *Algorithmica* **69**(2), 431–442 (2014)
5. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, pp. 215–267. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59126-6_4
6. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. In: 9th Annual ACM Symposium on Theory of Computing (STOC 1977), pp. 1–10. ACM, New York (1977)
7. Lee, L.: Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM* **49**(1), 1–15 (2002)
8. Mráz, F., Průša, D., Wehar, M.: Two-dimensional pattern matching against basic picture languages. In: Hospodár, M., Jirásková, G. (eds.) CIAA 2019. LNCS, vol. 11601, pp. 209–221. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23679-3_17
9. de Oliveira Oliveira, M., Wehar, M.: Intersection non-emptiness and hardness within polynomial time. In: Hoshi, M., Seki, S. (eds.) DLT 2018. LNCS, vol. 11088, pp. 282–290. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98654-8_23
10. Peeters, R.: The maximum edge biclique problem is NP-complete. *Discret. Appl. Math.* **131**(3), 651–654 (2003)
11. Potechin, A., Shallit, J.: Lengths of words accepted by nondeterministic finite automata. CoRR abs/1802.04708 (2018)
12. Salomaa, A.: *Jewels of Formal Language Theory*. Computer Science Press, Rockville (1981)
13. Sun, X., Nobel, A.B.: On the size and recovery of submatrices of ones in a random binary matrix. *J. Mach. Learn. Res.* **9**(Nov), 2431–2453 (2008)
14. Williams, V.V.: Multiplying matrices faster than Coppersmith-Winograd. In: 44th Annual ACM Symposium on Theory of Computing (STOC 2012), pp. 887–898. ACM, New York (2012)
15. Williams, V.V.: Hardness of easy problems: basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In: Husfeldt, T., Kanj, I.A. (eds.) 10th International Symposium on Parameterized and Exact Computation (IPEC 2015). LIPIcs, vol. 43, pp. 17–29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015)