# The Maximum Matrix Contraction problem

Dimitri Watel[1,2] and Pierre-Louis Poirion[1,3]

[1] CEDRIC-CNAM, 292 rue du faubourg Saint Martin, 75003, Paris, FRANCE
[2] ENSIIE, 1 Square de la résistance, Evry, FRANCE dimitri.watel@ensiie.fr,
[3] ENSTA Paristech pierre-louis.poirion@ensta-paristech.fr

**Abstract.** In this paper, we introduce the *Maximum Matrix Contraction problem*, where we aim to contract as much as possible a binary matrix in order to maximize its density. We study the complexity and the polynomial approximability of the problem. Especially, we prove this problem to be NP-Complete and that every algorithm solving this problem is at most an $\sqrt{n}$-approximation algorithm where $n$ is the number of ones in the matrix. We then focus on efficients algorithm to solve the problem: a linear program and three heuristics.
*Keywords:* Complexity, Approximation algorithm, Linear Programming

## 1 Introduction

In this paper, we define the following problem. We are given a two dimensional array in which some entries contain a dot and others are empty. Two lines $i$ and $i+1$ of the grid can be contracted by shifting up every dot of line $i+1$ and of every line after. Two columns $j$ and $j+1$ of the grid can be contracted by shifting left the corresponding dots. However, such a contraction is not allowed if two dots are brought into the same entry. The purpose is to ensure that the number of neighbor pairs of dots (including the diagonal ones) is maximized. Figure 1 illustrates examples of valid contractions and non-valid ones and an optimal solution.

*Motivation.* This problem have an application in optimal sizing of wind-farms [1] where we must first define, from a given set of wind-farms location, the neighborhood graph of this set, i.e. the graph such that two wind farms are connected if and only if their corresponding entries in the grid are neighbors. More precisely, given a set of points in the plane, we consider a first grid-embedding such that any two points are at least separated by one vertical line and one horizontal line. Then, we consider the problem of deciding which lines and columns to contract such that the derived embedding maximize the density of the grid, i.e., the number of edges in the corresponding neighbor graph.

*Contributions.* In this paper, we formally define the grid contraction problem as a binary matrix contraction problem in which every dot is a 1 and every other entry is 0. We study the complexity and the polynomial approximability of the problem. Especially, we prove this problem to be NP-Complete. Nonetheless, every algorithm solving this problem is at most an $\sqrt{n}$-approximation algorithm

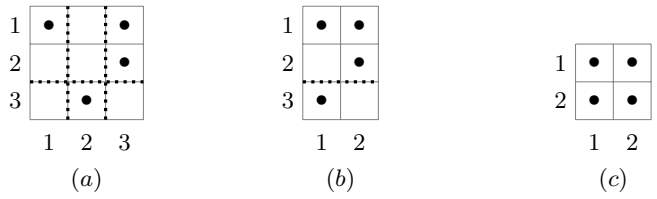


Fig. 1: In Figure 1.a, we give a $3 \times 3$ grid containing four dots. We can contract lines 2 and 3, columns 1 and 2 and columns 2 and 3 but it is not allowed to contract lines 1 and 2 because this would lead to put the two dots of coordinates (1;3) and (2;3) into the same entry (1;3). Valid contractions are represented by dotted lines and columns. Figure 1.b is the result of the contraction of columns 1 and 2 and Figure 1.c is the result of the contraction of lines 2 and 3. The number of neighbor pairs in each grid is respectively 2, 4 and 6.

where $n$ is the number of 1 in the matrix. We then focus on efficient algorithm to solve the problem. We first investigate the mathematical programming formulation of MMC. We give two formulations: a straightforward non-linear program and a linear program. Secondly, we describe three polynomial heuristics for the problem. We finally give numerical tests to compare the performances of the linear program and each algorithm.

In Section 2, we give a formal definition of the problem. In Section 3, we prove that the corresponding decision problem is NP-complete, then we give, in Section 4 some results about approximability of the problem. In Section 5 we derive a linear integer program for the model and run some experiments, then in the next section, we present and compare the three different heuristics.

## 2 Problem definition

The following definitions formalize the problem we want to solve with binary matrices. A binary matrix is a matrix with entries from $\{0, 1\}$. Such a matrix modelizes a grid in which each dot is a 1 in the matrix.

**Definition 1.** *Let $M$ be a binary matrix with $p$ lines and $q$ columns. For each $i \in [\![1; p-1]\!]$* and each $j \in [\![1; q-1]\!]$, we define the* line contraction matrix $L_i$ *and the* column contraction matrix $C_j$ *by*

*The meaning of $[\![p; q]\!]$ is the list $[p+1, p+2, \ldots, q]$.

$$L_i = \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & & i & & & & & p & & \end{array} \\ \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & \\ 0 & 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 & \\ 0 & 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 & \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & \end{pmatrix} \begin{array}{l} 1 \\ 2 \\ \\ \\ i \\ \\ \\ \\ p \end{array} \end{array} \quad C_j = \begin{array}{c} \begin{array}{ccccccccccc} 1 & 2 & & j & & & & & q & & \end{array} \\ \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \\ 0 & 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & \end{pmatrix} \begin{array}{l} 1 \\ 2 \\ \\ \\ \\ j \\ \\ \\ \\ q \end{array} \end{array}.$$

*The size of $L_i$ is $p \times p$ and the size of $C_j$ is $q \times q$.*

**Definition 2.** *Let $M$ be a binary matrix of size $p \times q$, $I = [i_1, i_2, \ldots, i_{|I|}]$ a sublist of $[\![1; p-1]\!]$ and $J = [j_1, j_2, \ldots, j_{|I|}]$ a sublist of $[\![1; q-1]\!]$. We assume $I$ and $J$ are sorted. We define the* contraction $C(M, I, J)$ of the lines $I$ and the columns $J$ of $M$ *by the following matrix*

$$C(M, I, J) = \left( \prod_{k=1}^{|I|} L_{i_k} \right) \cdot M \cdot \left( \prod_{k=|J|}^{1} C_{j_k} \right).$$

*Example 1.* Let $M$ be the matrix corresponding to the grid $(a)$ of Figure 1. The following contraction gives the grid $(c)$ :

$$C(M, [2], [1]) = L_2 \cdot M \ldots C_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

**Definition 3.** *A contraction $C(M, I, J)$ is said* valid *if and only if $C(M, I, J)$ is a binary matrix.*

*Example 2.* The following contraction is not valid :

$$C(M, [1, 2]) = L_1 \cdot L_2 \cdot M = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

**Definition 4.** *Let $M$ be a binary matrix of size $p \times q$. The* density *is the number of neighbor pairs of 1 in the matrix (including the diagonal pairs). This value may be computed with the following formula :*

$$d(M) = \frac{1}{2} \cdot \sum_{i,j} \left( M_{i,j} \cdot \left( \sum_{\delta=-1}^{1} \sum_{\gamma=-1}^{1} M_{i+\delta, j+\gamma} \right) - 1 \right)$$

*where we define that $M_{i,j} = 0$ if $(i, j) \notin [\![1; p-1]\!] \times [\![1; q-1]\!]$*

*Problem 1. Maximum Matrix Contraction problem* (MMC). Given a binary matrix $M$ of size $p \times q$ such that $n$ entries equal 1 and $p \cdot q - n$ entries equal 0, the Maximum Matrix Contraction problem consists in the search for two sublists $I$ of $[\![1; p-1]\!]$ and $J$ of $[\![1; q-1]\!]$ such that the contraction $C(M, I, J)$ is valid and maximizes $d(C(M, I, J))$.

We study in the next two sections the complexity and the approximability of this problem.

## 3 Complexity

This section is dedicated to proving the NP-Completeness of the problem.

**Theorem 1.** *The decision version of (MMC) is NP-Complete.*

*Proof.* Lets $M$ be an instance of MMC. Given an integer $K$, a sublist $I$ of $[\![1; p-1]\!]$ and a sublist $J$ of $[\![1; q-1]\!]$, one can compute in polynomial time the matrix $C(M, I, J)$ and checks if $d(C(M, I, J)) \geq K$. This proves the problem belongs to NP.

In order to prove the NP-Hardness, we describe a polynomial reduction from the NP-Complete Maximum Clique problem [2]. Lets $G(V, E)$ be an instance of the Maximum Clique problem, we build an instance $M$ of MMC with $p = q = (4|V| + 6)$. We arbitrarily number the nodes of $G : V = \{v_1, v_2, \dots v_{|V|}\}$.

Let $l_i$ and $c_i$ be respectively the $6+4(i-1)+1$-th line and the $6+4(i-1)+1$-th column. We associate the four lines $l_i, l_i+1, l_i+2, l_i+3$ and the four columns $c_i, c_i+1, c_i+2, c_i+3$ to $v_i$. The key idea of the reduction is that each node $v$ is associated with two 1 of the matrix. If we choose the node $v$ to be in the clique, then, firstly, the two 1 associated with are moved next to each other and this increases the density by one; and secondly, for every node $w$ such that $(v, w) \notin E$, the two 1 associated to cannot be moved anymore.

We use three gadgets in this reduction, described in Figure 2. The gadget $(\alpha_i)$ is added at the intersection of the lines and the columns associated with the node $v_i$. The gadget $(\beta_{i,j})$ is added at the intersections of the lines of $v_i$ and the columns of $v_j$ such that $(v_i, v_j) \notin E$. If $v_i$ and $v_j$ are linked with an edge in $G$, we do not add the gadget and all the entries equal 0. We add the gadget $(\gamma_i)$ for each node $v_i$ at line 1, and a transposed gadget at column 1. Finally, we set $M_{1,j} = M_{j,1} = 1$ for each $j \in [\![1; 6]\!]$. A complete example is given in Figure 3.

The initial density in this matrix is $d_0 = 11+6|V|+(|V|(|V|-1)-2|E|)$. Note that, firstly, only the contractions of the line $l_i$ and the column $c_i$ for $i \in [\![1; n]\!]$ are valid and, secondly, we cannot increase the number of neighbors of the 1's that are not contained in the gadgets $(\alpha_i)$. As a consequence, in order to add one to the density of the matrix we must choose a node $v_i$ and contract the column $c_i$ and the line $l_i$. If the column $c_i$ is contracted and if $(v_i, v_j) \notin E$, the two 1's of the gadget $(\beta_{i,j})$ are moved on the same column. Similarly, if the line $l_i$ is contracted, the two 1's of the gadget $(\beta_{j,i})$ are moved on the same column. This prohibits the contraction of the line $l_j$ and the column $c_j$. Consequently, in

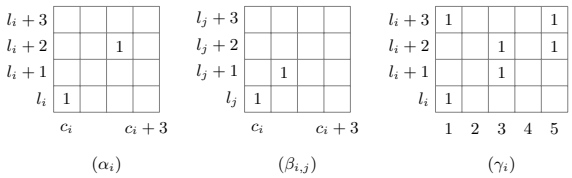


Fig. 2: The three gadgets of the reduction. We do not show the 0 entries of the matrices for readability. We specify on each gadget the indexes of its lines and columns.

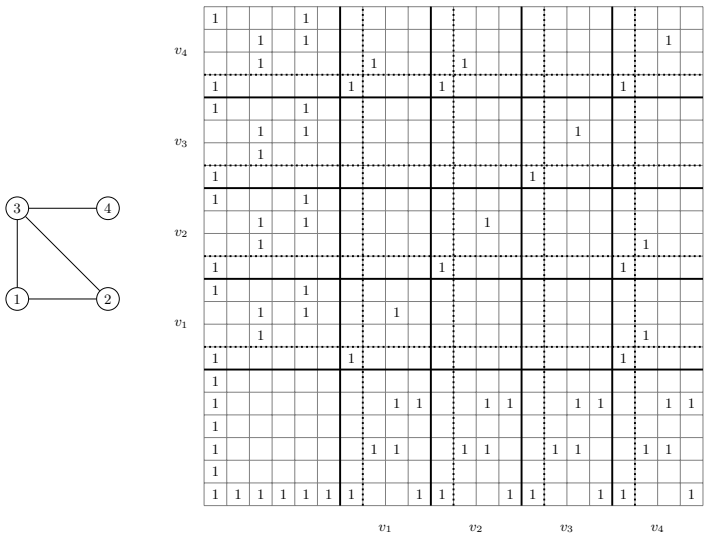


Fig. 3: This figure illustrates, on the left, a graph in which we search for a maximum clique and, on the right, the matrix obtained built with the reduction. We do not show the 0 entries of the matrix for readability. Each dotted line and column represent the valid contractions.

order to add $C$ to the density, we must find a clique of size $C$ in the graph and contract every line and column associated with the nodes of that clique.

Thus, there is a clique of size $K$ if and only if there is a feasible solution for $M$ of density $d_0 + K$. This concludes the proof of NP-Completeness.

□

The Maximum Clique problem cannot be approximated to within $|V|^{\frac{1}{2}-\varepsilon}$ in polynomial time unless P = NP [3]. Unfortunately, the previous reduction cannot be used to prove a negative approximability result occurs for MMC. Indeed, the density of any feasible solution of the MMC instance we produce is between $d_0+1$ and $d_0+|V|$, with $d_0 = O(|V|^2 - |E|)$. Consequently, the optimal density is at most $(1 + 1/|V|)$ times the worst density. A way to prove a high inapproximability ratio for MMC would be to modify the reduction such that the gap between the optimal solution and another feasible solution increase.

In the next section, we prove that a $n^{\frac{1}{2}-\varepsilon}$ harness ratio would almost tight the approximability of MMC as there exists an $2\sqrt{n}$-approximation algorithm.

## 4 Approximability

In this section we define the notion of maximal feasible solution and prove that every algorithm returning a maximal feasible solution is a $2\sqrt{n}$-approximation where $n$ is the number of 1 in the matrix.

**Definition 5.** *We say a feasible solution is* maximal *if it is not strictly included in another feasible solution. In other words, when all the lines and columns of that solution are contracted, no contraction of any other line or column is valid.*

**Lemma 1.** *Let $M$ be an instance of MMC, $(I, J)$ be a maximal feasible solution and $M' = C(M, I, J)$ then $2\sqrt{n} \leq d(M') \leq 4n$.*

*Proof.* A 1 in a matrix cannot have more than 8 neighbors, thus the density of $M'$ is no more than $4n$.

Let $p'$ and $q'$ be respectively $p-|I|$ and $q-|J|$. For each line $i \in [\![1, p'-1]\!]$ of $M'$, there is a column $j$ such that $M'_{i,j} = M'_{i+1,j} = 1$, otherwise we could contract line $i$ and $(I, J)$ would not be maximal. Similarly for each column $j \in [\![1, q'-1]\!]$. Thus $d(M') \geq p' + q'$ where $p' \times q'$ is the size of $M'$. From the inequality of arithmetic and geometric means, we have $p' + q' \geq 2\sqrt{p' \cdot q'}$ and, as $M'$ contains $n$ entries such that $M'_{i,j} = 1$, $p' \cdot q' \geq n$. Thus $p' + q' \geq 2\sqrt{n}$. □

From the upper bound and the lower bound given in the previous lemma, we can immediately prove the following theorem.

**Theorem 2.** *An algorithm returning any maximal solution of an instance of MMC is a $2\sqrt{n}$-approximation.*

Theorem 2 proves a default ratio for every algorithm trying to solve the problem. Note that there are instances in which the ratio between an optimal density and the lowest density of a maximal solution is $O(\sqrt{n})$. An example is given in Appendix A. In Section 6, we describe three natural heuristics to solve the problem. For two of them returns, the instance of Appendix A may be adapted to show their approximability ratio is $O(\sqrt{n})$.

Determining if MMC can be approximated to within a constant factor is an open question. As it was already pointed at the end of Section 3, the problem may possibly be not approximable to within $n^{\frac{1}{2}-\varepsilon}$ and this would almost tight the approximability of MMC.

The next two sections focus on efficient algorithms to solve the problem. The next section is dedicated to the mathematical programming methods.

## 5 Linear integer programming

For $i \in [\![1; p-1]\!]$ (resp. $j \in [\![1; q-1]\!]$ ), let $x_i$ (resp. $y_j$) be the binary variable such that $x_i = 1$ (resp. $y_j = 1$) if and only if line $i$ is contracted, i.e. $i \in I$ (resp. column $j$ is contracted, i.e. $j \in J$). From the definitions of Section 2, we can model the MMC problem by the following non-linear binary program:

$$(*) \begin{cases} \max\limits_{x,y} & d(A) \\ & A = \prod\limits_{i=1}^{p-1} ((L_i - I_p)x_i + I_p)M \prod\limits_{j=q-1}^{1} ((C_j - I_q)y_j + I_q) \\ & A_{i,j} \leq 1, \quad \forall (i,j) \in [\![1; p-1]\!] \times [\![1; q-1]\!] \\ & x_i, y_j \in \{0,1\} \end{cases}$$

where $I_p, I_q$ denotes the identity matrix and where the formula of $d(A)$ is the one given in Definition 4.

Although this formulation is very convenient to write the mathematical model, it is intractable as we would need to add an exponential number of linearizations: for all subset $I, J \subseteq [\![1; p-1]\!] \times [\![1; q-1]\!]$ we would need a variable $x_I = \prod\limits_{i \in I} x_i$ and $y_I = \prod\limits_{j \in J} y_j$.

We now present a linear integer programming model for the MMC problem: instead of linearizing the products $\prod\limits_{i \in I} x_i$ and $\prod\limits_{j \in J} y_j$, we cut the product

$A = \prod\limits_{i=1}^{p-1} ((L_i - I_p)x_i + I_p)M \prod\limits_{j=q-1}^{1} ((C_j - I_q)y_j + I_q)$ in $T = p + q - 1$ time-steps.

More precisely, define $A^{(1)} = M$; for all $t = 2, ..., p$, we define by $A^{(t)}$ the matrix which is computed after deciding the value of $y_j$ for $j \geq p - t + 1$; similarly, for all $p + 1 \leq t \leq T$, $A^{(t)}$ is determined by the value of $y_j$ for all $j$ and by the value of $x_i$ for $i \geq q - t + p$. We obtain the following program:

$$(P)\begin{cases}\max\limits_{x,y} & d(A^{(T)}) \\ & A^{(t+1)} = ((L_{p-t} - I_p)x_{p-t} + I_p)A^{(t)} & \forall 1 \leq t \leq p-1 \\ & A^{(t+1)} = A^{(t)}((C_{q-t+p} - I_q)y_{q-t+p} + I_q) & \forall p \leq t \leq T \\ & A^{(t)}_{i,j} \leq 1, \quad \forall (i,j,t) \in [\![1; p-1]\!] \times [\![1; q-1]\!] \times [\![2; T]\!] \\ & x_i, y_j \in \{0,1\}\end{cases}$$

We can easily linearize the model above by introducing, for all $(i,j,t) \in [\![1; p-1]\!] \times [\![1; q-1]\!] \times [\![2; T]\!]$ $r_{i,j,t} = A^{(t)}_{i,j} * x_{p-t}$ if $1 \leq t \leq p-1$ and $r_{i,j,t} = A^{(t)}_{i,j} * y_{q-t+p}$ if $p+1 \leq t \leq T$, noticing that the variables $A^{(t)}_{i,j}, x_t, y_t$ are all binary. Finally, after linearizing the product $A^{(T)}_{i,j} A^{(T)}_{k,l}$ in the objective function, $d(A^{(T)})$, we obtain a polynomial size integer programming formulation of the MMC problem.

### 5.1 Numerical results

We test the proposed model using IBM ILOG CPLEX 12.6. The experiments are performed on an Intel i7 CPU at 2.70GHz with 16.0 GB RAM. The models are implemented in Julia using JuMP [4]. The algorithm is run on random squared matrices. Given a size $p$ and a probability $r$, we produce a random binary matrix $M$ of size $p \times p$ such that $Pr(M_{i,j} = 1) = r$. The expected value of $n$ is then $r \cdot p^2$. We test the model for $n \in \{6, 9, 12\}$ for a probability $r \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$ and we report the optimal value $d^*$ and the time to find the optimal solution. For each value of $p$ and $r$, 10 random instances are created, whose averages are reported on Table 1. We notice that the integer programming model is not very

Table 1: Test of random instances for the linear program model.

| | r | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0.1 | | 0.15 | | 0.20 | | 0.25 | | 0.3 | |
| | $d^*$ | time (s) | $d^*$ | time (s) | $d^*$ | time (s) | $d^*$ | time (s) | $d^*$ | time (s) |
| (p,q)=(6,6) | 6.0 | 0.3 | 4.1 | 0.26 | 12.1 | 0.2 | 15.3 | 0.28 | 22.0 | 0.15 |
| (p,q)=(9,9) | 15.1 | 5.3 | 22.1 | 5.1 | 32.3 | 7.8 | 36.5 | 7.0 | 44.5 | 3.4 |
| (p,q)=(12,12) | 30.8 | 171.6 | 48.0 | 281.2 | 55.0 | 183.0 | 64.4 | 101.0 | 71.0 | 95.1 |

efficient to solve the problem. For $p = 15$ , in most of the cases, CPLEX need to run more than 2 hours to solve the model.

## 6 Polynomial heuristics

In this section, we describe three heuristics for MMC : two first-come-first-served algorithm and two greedy algorithms.

### 6.1 The LCL heuristic

This algorithm is a first-come-first-served algorithm. It is divided into two parts: the Line-Column (LC) part and the Column-Line (CL) part.

The LC part computes and returns a maximal feasible solution $M^{LC}$ by, firstly, contracting a maximal set of lines $I^{LC}$ and, then, by contracting a maximal set of columns $J^{LC}$. The algorithm builds $I^{LC}$ as follows: it checks for each line from $p-1$ down to 1 if the contraction of that line is valid. In that case, the contraction is done and the algorithm goes on. $J^{LC}$ is built the same way.

The CL part computes and returns a maximal feasible solution $M^{CL}$ by starting with the columns and ending with the lines. The LCL algorithm then returns the solution with the maximum density.

The advantage of such an algorithm is its small time complexity.

**Theorem 3.** *The time complexity of the LCL algorithm is $O(p \cdot q)$.*

*Proof.* The four sets $I^{LC}$, $J^{LC}$, $I^{CL}$ and $J^{CL}$ can be implemented in time $O(p \cdot q)$ using an auxiliary matrix $M'$. The proof is given for the first one, the implementation of the three other ones is similar. At first, we copy $M$ into $M'$. For each line $i$ from $p-1$ to 1 of $M'$, we check with $2q$ comparison if there is a column $j$ such that $M'_{i,j} = M'_{i+1,j} = 1$. In that case, we do nothing. Otherwise, we add $i$ to $I^{LC}$ and we replace line $i$ with the sum of the $i$-th and the $i+1$-th lines.

Finally, given a matrix $M$ and a set of lines $I$, one can compute $C(M, I, \emptyset)$ in time $O(p \cdot q)$ by, firstly, computing in time $O(p)$ an array $A$ of size $p$ such that $A_i$ is the number of lines in $I$ strictly lower than $i$ and, secondly, returning a matrix $C$ of size $p - |I| \times q$ such that $C_{i-A_i,j} = Mi, j$. □

*Remark 1.* Note that, if there is at most one 1 per line of the matrix of the matrix, the LCL algorithm is asymptotically a 4-approximation when $n$ approaches infinity. Indeed, the LC part returns a line matrix in which each entry is a 1. The density of this solution is $n-1$. As the maximum density is $4n$ by Lemma 1, the ratio is $4\frac{n}{n-1}$. On the contrary, the example given in Appendix A can be adapted to prove this algorithm is, in the worst case, at least a $\sqrt{n}$-approximation.

### 6.2 The greedy algorithm

The greedy algorithm tries to maximize the density at each iteration. The algorithm computes $d(C(M, \{i\}, \emptyset))$ and $d(C(M, \emptyset, \{j\}))$ for each line $i$ and each column $j$ if the contraction is valid. It then chooses the line or the column maximizing the density. It starts again until the solution is maximal.

**Theorem 4.** *The time complexity of the Greedy algorithm is $O(p^2 \cdot q^2)$.*

*Proof.* There are at most $p \cdot q$ iterations. At each iteration, we computes one density per line $i$ and one density per column $j$. The density of $C(M, \{i\}, \emptyset)$ is the density of $M$ plus the number of new neighbor pairs of 1 due to the contraction of lines $i$ and $i+1$. The increment can be computing in time $O(q)$ as

there are at most three new neighbors for each of the $q$ entries of the four lines $i-1$ to $i+2$. Similarly, the density of $C(M, \emptyset, \{j\})$ can be computing in time $O(p)$. Thus one iteration takes $O(p \cdot q)$ iterations. □

*Remark 2.* As for the LCL algorithm, we can also adapt the instance of Appendix 4 in order to lure the greedy algorithm and prove it is at least a $\sqrt{n}$-approximation algorithm.

### 6.3 The neighborization algorithm

The neighborization algorithm is a greedy algorithm trying to maximize, at each iteration, the number of couple of entries that can be moved next to each other with a contraction. This algorithm is designed to avoid the traps in which the LCL algorithm and the Greedy algorithm fall in by never contracting lines and columns that could prevent some 1 entries to gain a neighbor.

We define a function $N$ from $(\llbracket 0; p-1 \rrbracket \times \llbracket 0; q-1 \rrbracket)^2$ to $\{0,1\}$. For each couple $c = ((i,j),(i',j'))$ such that $M_{i,j} = 0$ or $M_{i',j'} = 0$, $N(c) = 0$. Otherwise, $N(c) = 1$ if and only if there is a sublist of lines $I$ and a sublist of columns $J$ such that $C(M, I, J)$ is valid and such that the two entries are moved next to each other with this contraction. Finally, we define $N(M)$ as the sum of all the values $N((i,j),(i',j'))$. The algorithm computes $N(C(M, \{i\}, \emptyset))$ and $N(C(M, \emptyset, \{j\}))$ for each line $i$ and each column $j$ if the contraction is valid. It chooses the line or the column maximizing the result and starts again until the solution is maximal.

**Theorem 5.** *The time complexity of the Greedy algorithm is* $O(n^2 \cdot p^3 \cdot q^3 \cdot (p+q))$.

*Proof.* Let $M$ be a binary matrix, we first determine the time complexity we need to compute $N(M)$. Let $((i,j),(i',j'))$ be two coordinates such that $M_{i,j} = M_{i',j'} = 1$. We assume $i < i'$ and $j < j'$. The two entries may be moved next to each other if $i' - i - 1$ of the $i' - i$ lines and $j' - j - 1$ of the $j' - j$ columns between the two entries may be contracted and this can be done in time $O(p \cdot q \cdot (j' - j) \cdot (i' - i)) = O(p^2 \cdot q^2)$. As there are at most $n^2$ entries satisfying $M_{i,j} = M_{i',j'} = 1$, we need $O(n^2 \cdot p^2 \cdot q^2)$ operations to compute $N(M)$.

As there are at most $p \cdot q$ iterations. At each iteration, we computes one value per line $i$ and one value per column $j$ in time $O(n^2 \cdot p^2 \cdot q^2)$. The time complexity is then $O(n^2 \cdot p^3 \cdot q^3 \cdot (p+q))$. □

### 6.4 Numerical results

In this last subsection, we give numerical results of the three algorithms in order to evaluate their performances.

The experiments are performed on an Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz processor with 8Go of RAM. The algorithms are implemented with Java 8[§]. The algorithms are run on random squared matrices. Given a size $p$ and a probability $r$, we produce a random binary matrix $M$ of size $p \times p$ such that

[§]The implementations can be found at https://github.com/mouton5000/MMCCode.

$Pr(M_{i,j} = 1) = r$. The expected value of $n$ is then $r \cdot p^2$. Before executing each algorithm, we first reduce the size of each instance by removing every column and line with no 1.

*Small instances.* We first test the three algorithm on small instances on which we can compute an exact brute-force algorithm. This algorithm exhaustively enumerates every subset of lines and columns for which the contraction is valid and returns the solution with maximum density. The results are summarized on Table 2 and Table 3.

Table 2: This table details the results for each algorithm. For each values of $p$ and $r$, the algorithms are executed on 50 instances. We give for each heuristic the mean running time in milliseconds, the mean ratio between the optimal density $d^*$ and returned density $d$ and the number of instances for which the ratio is 1.

| | | Exact | LCL | | | Greedy | | | Neigh. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $p$ | $r$ | time (ms) | time (ms) | $\frac{d^*}{d}$ | $d = d^*$ | time (ms) | $\frac{d^*}{d}$ | $d = d^*$ | time (ms) | $\frac{d^*}{d}$ | $d = d^*$ |
| 5 | 0.01 | $< 1$ ms | $< 1$ ms | 1 | 50 | $< 1$ ms | 1 | 50 | $< 1$ ms | 1 | 50 |
| | 0.02 | $< 1$ ms | $< 1$ ms | 1.00 | 49 | $< 1$ ms | 1 | 50 | $< 1$ ms | 1.00 | 49 |
| | 0.03 | $< 1$ ms | $< 1$ ms | 1 | 50 | $< 1$ ms | 1 | 50 | $< 1$ ms | 1 | 50 |
| | 0.04 | $< 1$ ms | $< 1$ ms | 1.00 | 48 | $< 1$ ms | 1 | 50 | $< 1$ ms | 1.00 | 47 |
| | 0.05 | $< 1$ ms | $< 1$ ms | 1.00 | 48 | $< 1$ ms | 1.00 | 49 | $< 1$ ms | 1.00 | 48 |
| | 0.1 | $< 1$ ms | $< 1$ ms | 1.00 | 48 | $< 1$ ms | 1.00 | 48 | $< 1$ ms | 1.00 | 43 |
| | 0.2 | $< 1$ ms | $< 1$ ms | 1.00 | 45 | $< 1$ ms | 1.00 | 41 | $< 1$ ms | 1.00 | 37 |
| | 0.3 | $< 1$ ms | $< 1$ ms | 1.00 | 44 | $< 1$ ms | 1.00 | 44 | 1.24 | 1.00 | 39 |
| 10 | 0.01 | $< 1$ ms | $< 1$ ms | 1.00 | 46 | $< 1$ ms | 1 | 50 | $< 1$ ms | 1.00 | 46 |
| | 0.02 | $< 1$ ms | $< 1$ ms | 1.02 | 41 | $< 1$ ms | 1.00 | 47 | $< 1$ ms | 1.00 | 40 |
| | 0.03 | $< 1$ ms | $< 1$ ms | 1.08 | 36 | $< 1$ ms | 1.00 | 41 | $< 1$ ms | 1.00 | 38 |
| | 0.04 | $< 1$ ms | $< 1$ ms | 1.02 | 31 | $< 1$ ms | 1.00 | 42 | 2.24 | 1.00 | 29 |
| | 0.05 | $< 1$ ms | $< 1$ ms | 1.02 | 25 | $< 1$ ms | 1.00 | 29 | 3.42 | 1.00 | 25 |
| | 0.1 | $< 1$ ms | $< 1$ ms | 1.00 | 20 | $< 1$ ms | 1.00 | 23 | 17.90 | 1.00 | 22 |
| | 0.2 | 1.14 | $< 1$ ms | 1.00 | 27 | $< 1$ ms | 1.00 | 19 | 64.58 | 1.00 | 19 |
| | 0.3 | $< 1$ ms | $< 1$ ms | 1.00 | 30 | $< 1$ ms | 1.00 | 33 | 65.52 | 1.00 | 29 |
| 15 | 0.01 | $< 1$ ms | $< 1$ ms | 1.08 | 35 | $< 1$ ms | 1.00 | 45 | $< 1$ ms | 1.00 | 39 |
| | 0.02 | $< 1$ ms | $< 1$ ms | 1.14 | 29 | $< 1$ ms | 1.00 | 31 | 1.28 | 1.00 | 31 |
| | 0.03 | 3.40 | $< 1$ ms | 1.10 | 20 | $< 1$ ms | 1.00 | 21 | 8.40 | 1.00 | 22 |
| | 0.04 | 10.30 | $< 1$ ms | 1.04 | 12 | $< 1$ ms | 1.00 | 19 | 17.12 | 1.00 | 26 |
| | 0.05 | 21.20 | $< 1$ ms | 1.04 | 13 | $< 1$ ms | 1.00 | 9 | 53.92 | 1.00 | 15 |
| | 0.1 | 239.82 | $< 1$ ms | 1.00 | 9 | $< 1$ ms | 1.00 | 12 | 354.32 | 1.00 | 10 |
| | 0.2 | 53.30 | $< 1$ ms | 1.00 | 14 | $< 1$ ms | 1.00 | 18 | 997.60 | 1.00 | 17 |
| | 0.3 | $< 1$ ms | $< 1$ ms | 1.00 | 37 | $< 1$ ms | 1.00 | 35 | 574.74 | 1.00 | 20 |
| 20 | 0.01 | $< 1$ ms | $< 1$ ms | 1.14 | 24 | $< 1$ ms | 1.00 | 34 | 1.30 | 1.00 | 29 |
| | 0.02 | 68.36 | $< 1$ ms | 1.06 | 13 | $< 1$ ms | 1.00 | 13 | 21.02 | 1.00 | 19 |
| | 0.03 | 181.40 | $< 1$ ms | 1.02 | 10 | $< 1$ ms | 1.00 | 2 | 83.62 | 1.00 | 14 |
| | 0.04 | 3227.42 | $< 1$ ms | 1.00 | 5 | $< 1$ ms | 1.00 | 4 | 272.66 | 1.00 | 9 |
| | 0.05 | 27227.76 | $< 1$ ms | 1.00 | 3 | $< 1$ ms | 1.00 | 3 | 589.32 | 1.00 | 5 |
| | 0.1 | 78579.56 | $< 1$ ms | 1.00 | 3 | $< 1$ ms | 1.00 | 1 | 3499.12 | 1.00 | 6 |
| | 0.2 | 532.92 | $< 1$ ms | 1.00 | 14 | $< 1$ ms | 1.00 | 20 | 5382.82 | 1.00 | 13 |
| | 0.3 | 1.28 | $< 1$ ms | 1.00 | 35 | $< 1$ ms | 1.00 | 42 | 2060.16 | 1.00 | 31 |

We can observe from Table 2 that the running time first increases when $r$ grows and then decreases. Similarly, the number of times the heuristics returns an

Table 3: Each entry of this table details, for each couple of heuristics, the number of instances of Table 2 (there are 1600 instances) for which the line heuristic gives a strictly better results than the column heuristic.

| | LCL | Greedy | Neigh. |
|---|---|---|---|
| LCL | - | 344 | 343 |
| Greedy | 451 | - | 443 |
| Neigh | 396 | 415 | - |

optimal solution first decreases and then increases. The first behavior is explained by the fact that the size of instances with small values of $n$ can be reduced. On the other hand, if $r$ is high, the number of lines and columns of which the contraction is not valid increases and, then, the search space of the algorithms is shortened. Considering the running times, as it was predicted by the time complexities, the LCL and the greedy heuristics are the fastest algorithms. We can observe that the neighborization algorithm can be slower than the exact algorithm on small instances because the running time of the former is more influenced by $n$ than the latter. However, we do not exclude the fact the implementation of the neighborization algorithm may be improved. Considering the quality of the solutions returned by the algorithms, the Greedy and the neighborization heuristics show better performances than the LCL algorithm.

*Big instances.* We then test the two fastest algorithms LCL and Greedy on bigger instances. The results are given on Table 4. Four interesting differences with Table 2 emerges from Table 4. Firstly, the LCL algorithm is faster than the greedy algorithm. This is coherent with the time complexities. Secondly, the LCL algorithm does not follows the same behavior as the exact algorihtm and the neighborization heuristic for small instances: the running time increases with $r$ even if the search space is shortened. Indeed, contrary to the three other algorithms, the implementation does not depends on this search space. Thirdly, the running time of the greedy algorithm first increases with $r$, then decreases and and finally slowly increases again. This last increase is due to the computation time of the density and the line and columns that can be contracted. Finally, the solution returned by the LCL algorithm seems to be better for small values of $r$ and, on the other hand, the greedy algorithm returns better densities for middle values. The two algorithm are equivalent for high values of $r$ because those instances can probably not be contracted.

## 7 Conclusion

In this paper, we introduced the Maximum Matrix Contraction problem (MMC). We proved this problem is NP-Complete. However, we also proved that every algorithm which solve this problem is an $O(\sqrt{n})$-approximation algorithm. Considering that the NP-Completeness was derived from the Maximum Clique problem, and that this problem cannot be polynomially approximated to within $n^{\frac{1}{2}-\varepsilon}$,

Table 4: This table details the results for the LCL algorithm and the greedy alorithm. For each values of $p$ and $r$, the algorithms are executed on 50 instances. We give for each heuristic the mean running time in milliseconds and how many times the returned density is strictly better than the density returned by the other algorithm.

| | | LCL | | Greedy | | | | LCL | | Greedy | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $p$ | $r$ | time (ms) | $d_L > d_G$ | time (ms) | $d_L < d_G$ | $p$ | $r$ | time (ms) | $d_L > d_G$ | time (ms) | $d_L < d_G$ |
| 200 | 0.01 | $< 1$ ms | 49 | 17.78 | 1 | 1000 | 0.01 | 12.00 | 50 | 2832.52 | 0 |
| | 0.02 | $< 1$ ms | 48 | 22.58 | 2 | | 0.02 | 14.04 | 21 | 1890.40 | 29 |
| | 0.03 | $< 1$ ms | 43 | 21.82 | 5 | | 0.03 | 16.34 | 1 | 1099.38 | 49 |
| | 0.04 | $< 1$ ms | 31 | 19.26 | 18 | | 0.04 | 17.72 | 1 | 553.90 | 49 |
| | 0.05 | $< 1$ ms | 21 | 16.76 | 29 | | 0.05 | 18.74 | 5 | 233.70 | 45 |
| | 0.1 | 1.28 | 10 | 5.18 | 40 | | 0.1 | 24.72 | 0 | 7.82 | 0 |
| | 0.2 | 1.92 | 0 | $< 1$ ms | 0 | | 0.2 | 41.50 | 0 | 12.62 | 0 |
| | 0.3 | 2.58 | 0 | $< 1$ ms | 0 | | 0.3 | 59.18 | 0 | 18.36 | 0 |
| 500 | 0.01 | 3.28 | 50 | 382.06 | 0 | 2000 | 0.01 | 53.54 | 49 | 22068.00 | 1 |
| | 0.02 | 3.58 | 44 | 321.30 | 6 | | 0.02 | 59.96 | 0 | 10664.44 | 50 |
| | 0.03 | 3.92 | 17 | 237.06 | 33 | | 0.03 | 65.66 | 0 | 3914.08 | 50 |
| | 0.04 | 4.56 | 10 | 164.82 | 40 | | 0.04 | 71.68 | 6 | 1049.00 | 44 |
| | 0.05 | 4.88 | 4 | 104.48 | 46 | | 0.05 | 76.36 | 0 | 186.04 | 10 |
| | 0.1 | 6.80 | 0 | 4.70 | 2 | | 0.1 | 100.16 | 0 | 28.88 | 0 |
| | 0.2 | 10.66 | 0 | 3.34 | 0 | | 0.2 | 167.42 | 0 | 50.46 | 0 |
| | 0.3 | 15.06 | 0 | 4.58 | 0 | | 0.3 | 237.54 | 0 | 72.88 | 0 |

MMC is very likely to not being approximable to within the same ratio. Such a result would almost tight the approximability of MMC.

Moreover, we studied four algorithms to solve the problem, a linear program, a first-come-first-served algorithm and two greedy algorithms, and gave numerical results. It appears firstly that linear programming is not adapted to MMC while the three other heuristics returns really good quality solutions in short amount of time even for large instances. Those results seems to disconfirm the $n^{\frac{1}{2}-\varepsilon}$ inapproximability ratio. It would be interesting to deepen the study of those algorithm in order to produce a constant-factor polynomial approximation algorithm or a polynomial-time approximation scheme if such an algorithm exists.

## References


1. Pillai, A., Chick, J., Johanning, L., Khorasanchi, M., de Laleu, V.: Offshore wind farm electrical cable layout optimization. Engineering Optimization **47**(12) (2015) 1689–1708
2. Karp, R.: Reducibility among combinatorial problems. In: Complexity of Computer Computations. Springer (1972) 85–103
3. Håstad, J.: Clique is hard to approximate within n^(1-$\epsilon$). Acta Mathematica **182**(1) (mar 1999) 105–142
4. Lubin, M., Dunning, I.: Computing in operations research using Julia. INFORMS Journal on Computing **27**(2) (2015) 238–248

# A An instance with an $O(\sqrt{n})$ gap between the worst and the best solution.

Theorem 2 of Section 4 proves a default $2\sqrt{n}$ upper bound of the approximation ratio for every algorithm returning a maximal solution.

We give, in this appendix, in Figure 4, an instance in which the ratio between an optimal density and the lowest density of a maximal solution is at least $4\sqrt{n}$.

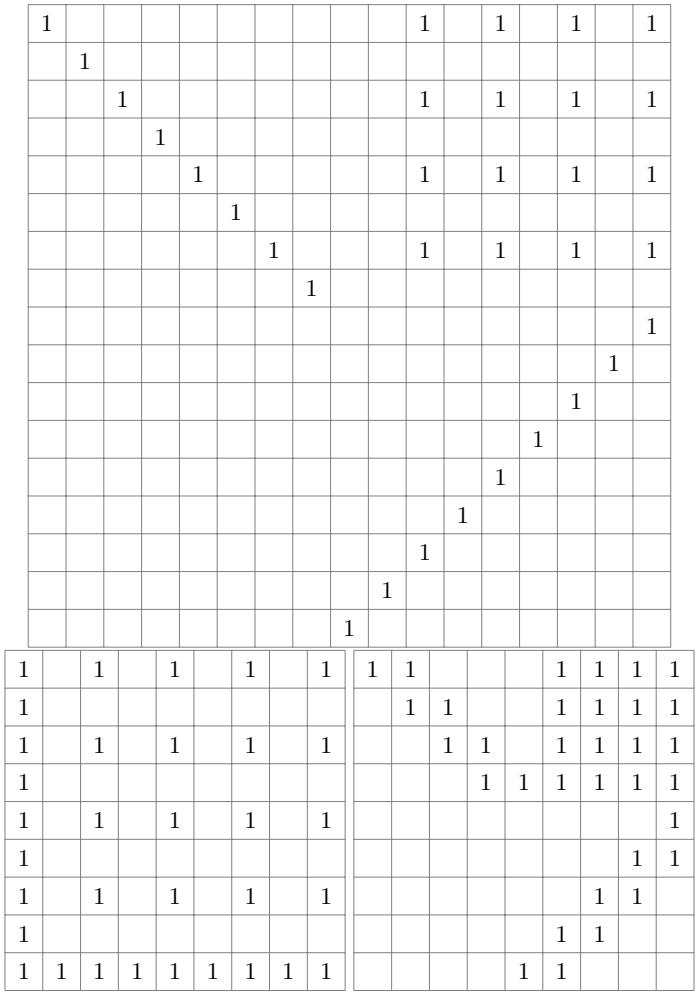

Fig. 4: In this instance, the density of an optimal solution is $O(n)$ and there is a maximal solution of density $O(\sqrt{n})$.

We highly believe this instance is the worst case that can happen and that the density of a maximal solution is always higher than $4\sqrt{n}$. The result of Theorem 2 may then possibly be updated to the following conjecture.

*Conjecture 1.* An algorithm returning any maximal solution of an instance of MMC is a $\sqrt{n}$-approximation.