INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2021) 1–16
DOI: 10.1111/itor.13065

Exponential neighborhood search for consecutive block minimization

Salim Haddadi*

LabSTIC, Université 8 Mai 1945, P.O. Box 401, Guelma 24000, Algeria
E-mail: haddadi.salim@univ-guelma.dz [Haddadi]

Received 15 May 2020; received in revised form 16 May 2021; accepted 16 September 2021

Abstract

Given binary matrix A, the term $A^{(\pi)}$ refers to the matrix obtained by permuting the columns of A according to permutation π. A 1-block in a binary matrix is any maximal sequence of consecutive 1s situated on the same row. Given binary matrix A, consecutive block minimization (CBM) is a combinatorial optimization problem that seeks a permutation π of A columns such that the number of 1-blocks in $A^{(\pi)}$ is minimum. Since CBM is NP-hard, we solve it by iterating local search where the neighborhood is exponential in the instance size and is always constructed in the vicinity of the best current solution. Seeking a local optimum in the exponential neighborhood amounts to solving a small traveling salesman problem (TSP). Instead, we obtain a good near local optimal solution using an implementation of Lin–Kernighan heuristic. Our method is compared with a recently published iterated local search metaheuristic as well as with two TSP solvers. The comparison shows that the proposed method provides better quality solutions.

Keywords: consecutive block minimization; exponential neighborhood search; local search; TSP

1. Introduction

Let A be a binary $m \times n$ matrix and let π be a permutation of $1, 2, \ldots, n$. The term $A^{(\pi)}$ refers to the matrix obtained by permuting the columns of A according to permutation π. For example, according to permutation $\pi = (2413)$, we have

$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \quad \text{and} \quad A^{(\pi)} = \begin{array}{c} \begin{matrix} 2 & 4 & 1 & 3 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array}. \tag{1}$$

Clearly, $A = A^{(\iota)}$, where $\iota = (1, 2, \ldots, n)$ is the identity permutation.

* Corressponding author.

A 1-block in a binary matrix is any maximal sequence of consecutive 1s situated on the same row. For instance, matrix A in (1) contains six 1-blocks. Given binary $m \times n$ matrix A, consecutive block minimization (CBM) seeks a permutation π such that the number of 1-blocks in $A^{(\pi)}$ is minimized. Let $b(A^{(\pi)})$ denote the number of 1-blocks in $A^{(\pi)}$, CBM can be regarded as the combinatorial optimization problem:

$$\min_{\pi} b(A^{(\pi)})$$

whose domain of solutions contains $n!$ possible permutations.

Given positive integer k, the decision version of CBM, called CBM decision, inquires whether there exists a permutation π such that the number of 1-blocks in $A^{(\pi)}$ is k. CBM-decision is NP-complete (Kou, 1977). Haddadi (2002) showed that CBM is NP-hard even on $(*, 2)$ matrices (i.e., matrices with at most two 1s per row and an arbitrary number of 1s per column). A good news is that CBM can be approximated to within at most 50% of the optimum (Haddadi and Layouni, 2008).

CBM may model any situation where we have to arrange a set of objects with the constraint that objects in any given subset must appear as linearly as possible. CBM occurs in many practical applications such as scheduling (Linhares and Yanasse, 2002; Paiva and Carvalho, 2017), production (Becceneri et al., 2004; Carvalho and Soma, 2015; Lima and Carvalho, 2017), glass industry (Dyson and Gregory, 1974; Madsen, 1979, 1988), and data compression (Johnson et al., 2004; Lemire and Kaser, 2011).

Surprisingly, the only computational method we know for CBM is a recently published iterated local search (ILS) metaheuristic (Soares et al., 2020). This is perhaps explained by the fact that CBM can be polynomially transformed to traveling salesman problem (TSP) so that any method for the latter can be used for solving the former. Johnson et al. (2004), for example, used very simple local search methods for solving CBM on huge binary matrices with hundreds of thousands of rows and columns. Similarly, Madsen (1988) used heuristics for TSP to solve a pattern-allocation problem in the glass industry modeled as CBM. The recent article by Bernardino and Paias (2021), published online, contains many interesting ideas that are worth exploring in the context of CBM.

If we confront such a long string of applications to the need for computational methods, we can conclude that a new good metaheuristic in this field would be noteworthy. This is our main motivation.

In this study, for solving CBM we propose a simple local search method where the size of the neighborhood is exponential in the instance size. Such a method is referred to as exponential neighborhood search (ENS). The last decade has seen an impressive number of research papers proposing to solve hard problems using ENS. Clearly, exponential neighborhoods give rise to better local optima but, in general, searching an exponential size neighborhood is time-consuming. Hence, ENS is pertinent only if there is an efficient method for searching the proposed exponential size neighborhood, or if we do not insist on finding the local optimum and we content ourselves with a near local optimal solution using some heuristic while scanning the neighborhood. The last choice is adopted in this study.

The proposed method is tested on a benchmark dataset, the same that has been used by Soares et al. (2020). If we compare our results with those obtained by the last authors, we find that our method is faster and provides better quality solutions since it always finds better configurations

more quickly. However, simply comparing approximate solutions without knowing how they are far from optimal is meaningless. Fortunately, since CBM is polynomially transformed to TSP, we can use a solver such as `Concorde` (Applegate et al., 2006a, 2006b) for getting an optimal solution. At the same time, we can use a good and well-known metaheuristic for TSP, the chained Lin–Kernighan method (Applegate et al., 2003), giving ourselves the opportunity to compare it with ours.

The article is organized as follows. Section 2 details the proposed method and Section 3 deals with the experimental phase and with the comparison of our method with competitive approaches.

2. Methods

We first recall in Section 2.1 that CBM can be solved exactly via a polynomial-time transformation to TSP. In Section 2.2, we present a small-sized neighborhood (Soares et al., 2020), then we propose its generalization to an exponential size neighborhood in Section 2.3. Section 2.4 explains how the neighborhood is constructed; then Section 2.5 summarizes the proposed method in pseudo-code and provides a useful illustrating example. The short sections, Section 2.6 and 2.7, provide some technical details. The first section shows how randomness is introduced in the binary matrix partition for escaping from local optima and the second section proposes to start our method from a good solution.

There is no loss of generality in assuming that binary matrices have no null columns and no null rows, and that their columns are distinct. Furthermore, in our context, rows with a single 1 are irrelevant because they lead to one 1-block regardless of the columns permutation. Hence, we will assume that binary matrices have at least two 1s per row.

2.1. Transforming CBM to TSP

Let A be a binary $m \times n$ matrix and let B be the $m \times (n+1)$ matrix obtained by concatenating to A a null column in position 0 (i.e., on the left of the first column). Let U be the $m \times (n+1)$ matrix whose entries are all 1s. Consider the symmetric matrix of order $n+1$:

$$W = B^T \cdot (U - B) + (U - B)^T \cdot B. \tag{2}$$

The value w_{ij} is a kind of " distance" between columns a_i and a_j of A, the smaller the value, the greater our wish to put the two columns side by side. It has been shown in Haddadi and Layouni (2008) that the symmetric matrix W in (2) has some nice properties. It satisfies the triangle inequality and the length of any cycle is even. Furthermore, since there are no identical columns, any value w_{ij} is less than m.

Solve TSP with instance matrix W and let Π be an optimal tour. Remove node 0 (which corresponds to the column labeled 0 concatenated to A) to obtain a path π on n nodes (which can be seen as a permutation of A columns).

Claim 1 (Haddadi and Layouni, 2008)**.** *There exists a permutation π of the columns of A giving a minimum number μ of 1-blocks in $A^{(\pi)}$ if and only if TSP on instance W has an optimal tour Π of length 2μ.*

Binary matrix A is C1P, if there exists a permutation π of its columns such that the 1s occur consecutively in each row of $A^{(\pi)}$. For instance, matrix A in (1) is C1P. As a corollary of Claim 1, we have a new characterization of C1P binary matrices.

Claim 2. *A is C1P if and only if TSP on instance W has an optimal tour of length $2m$.*

Proof. According to our assumption, A has no null rows. Hence, it is C1P if and only if there exists a permutation of its columns giving m 1-blocks. From Claim 1, the binary matrix has m 1-blocks if and only if TSP has an optimal tour of length $2m$. □

Unfortunately, we are unable to find any polynomial-time algorithm for recognizing C1P matrices based on the above characterization.

2.2. An $O(n^2)$-size neighborhood

Suppose we are given a binary matrix and suppose for simplicity that the actual solution is the identity permutation $\pi = (1, \ldots, n)$ giving b 1-blocks. Hence, π is the solution in the vicinity of which we want to define a neighborhood for local search. We consider a small neighborhood where a move

(i) considers two distinct columns, a_j for some $j \in \{1, \ldots, n\}$ and a_k for some $k \in \{1, \ldots, n\}$, $k \neq j$ (without loss of generality assume $j < k$, otherwise we can easily find the appropriate rule);
(ii) computes $\sigma = w_{j-1,j} + w_{k,k+1} - w_{j-1,k} - w_{j,k+1}$ (it is proven in Haddadi and Layouni, 2008, that σ is always even);
(iii) if $\sigma > 0$, reverse completely the order of the columns of the submatrix which starts in column a_j and ends in column a_k in such a way that column a_k becomes the first and column a_j becomes the last.

By doing so, the new better solution is $\pi' = (1, \ldots, a_{j-1}, a_k, a_{k-1}, \ldots, a_{j+1}, a_j, a_{k+1}, \ldots, a_n)$ giving a smaller number of 1-blocks, which is $b - \sigma/2$. Since there are $O(n^2)$ ways of choosing two distinct columns, the size of the considered neighborhood is $O(n^2)$. Furthermore, since σ is evaluated in $O(1)$, exploring the neighborhood takes time $O(n^2)$.

Let us illustrate the discussion above. Consider matrix A in (3), which has eight 1-blocks and let $j = 2$ and $k = 4$. Compute $\sigma = w_{12} + w_{45} - w_{14} - w_{25} = 2 + 4 - 2 - 2 = 2$. Hence, it would be advantageous to reverse the order of the columns of the submatrix that starts in column 2 and ends in column 4. The resulting configuration is shown in matrix A', which has only seven 1-blocks:

$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \quad A' = \begin{array}{c} \begin{matrix} 1 & 4 & 3 & 2 & 5 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}. \tag{3}$$

2.3. An exponential size neighborhood

The idea of reversing a single submatrix in the previous neighborhood can be generalized to any number $p \geq 3$ of submatrices. Suppose that the given binary matrix A is partitioned into p submatrices, the first beginning in column a_1 and ending in column a_{j_1}, the second beginning in column a_{j_1+1} and ending in column a_{j_2}, and so on until the last and pth submatrix starting with column $a_{j_{p-1}+1}$ and ending in column a_n:

$$\begin{array}{cccccccccc} 1 & \cdots & j_1 & j_1+1 & \cdots & j_2 & \cdots & j_{p-1}+1 & \cdots & n \\ (a_1 & \cdots & a_{j_1} & a_{j_1+1} & \cdots & a_{j_2} & \cdots & a_{j_{p-1}+1} & \cdots & a_n). \end{array} \tag{4}$$

Anyone of the p submatrices can be moved anywhere between two other submatrices, either with the given order of its columns or with the reversed order. The important requirement here is that the order of the internal columns of any submatrix should not be altered.

Consider the 4×8 matrix A that is partitioned into three submatrices intentionally separated. Matrix A has 11 1-blocks:

$$A = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \end{array}$$

We give in (5) an example of a regular move: the second submatrix is reversed and put before the first and the third submatrix is reversed. Observe that the new configuration contains 14 1-blocks:

$$\begin{array}{c} \begin{array}{cccccccc} 5 & 4 & 1 & 2 & 3 & 8 & 7 & 6 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \end{array} \tag{5}$$

Let us return to the partition of A into p submatrices presented in (4). We consider the neighborhood called `Submatrices_moving`. Since there are p submatrices and since each submatrix can be moved either with the given order of its columns or flipped, the neighborhood contains $p! \times 2^p$ possible configurations. Since p is chosen in the order of n, `Submatrices_moving` has an exponential size.

Note that the internal columns of any submatrix do not play any role. Hence, we need only to consider the following matrix, call it D, with $2p + 1$ columns (remember the null column in position 0):

$$\begin{array}{ccccccccc} 0 & 1 & j_1 & j_1+1 & j_2 & \cdots & j_{p-1}+1 & n \\ (\mathbf{0} & a_1 & a_{j_1} & a_{j_1+1} & a_{j_2} & \cdots & a_{j_{p-1}+1} & a_n). \end{array}$$

The local optimum in `Submatrices_moving` can be obtained directly by solving a small TSP on $2p + 1$ cities. Formally, we construct a simple and complete graph with $2p + 1$ nodes, node 0, and

nodes l_i and r_i, $i = 1, \ldots, p$, where l_i and r_i represent, respectively, the label of the leftmost column and the rightmost one of the ith submatrix. The symmetric length matrix is

$$\Delta = D^T \cdot (U - D) + (U - D)^T \cdot D, \tag{6}$$

where U is the matrix having the same dimension as D with all of its coefficients being 1s. Finally, we enforce nodes l_i and r_i to be adjacent in any optimal tour by setting $\Delta_{l_i,r_i} = -M$ for all $i =, 1, \ldots, p$, where M is an appropriate big positive integer value.

2.4. How the exponential neighborhood is constructed ?

In other words, given positive integer p as parameter, how the binary matrix is partitioned into p submatrices ? Suppose that the current best solution at our disposal is $\pi = (\pi_1, \ldots, \pi_n)$. Recall the distance matrix defined in (2) between columns and consider the distances between adjacent columns, which are the components of the $n - 1$ vector $(w_{\pi_1\pi_2}, w_{\pi_2\pi_3}, \ldots, w_{\pi_{n-1}\pi_n})$. Since we wish pairs of columns to be separated by a small distance, a good idea is find $p - 1$ pairs of adjacent columns with the largest distances and partition the given matrix according to these pairs.

As an illustrative example, consider binary matrix A with $n = 8$ columns, which we want to partition into $p = 3$ submatrices:

$$A = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Hence, we need to choose $p - 1 = 2$ pairs of adjacent columns separated by the largest distance. The two rows below provide, respectively, the labels of the eight columns and the distances separating them:

$$\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ & 2 & 3 & 1 & 2 & 3 & 2 & 2 \end{array}.$$

Since column 2 is "far away from" column 3, and column 5 from column 6, our idea is to partition the matrix according to these two pairs:

$$A = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & & 3 & 4 & 5 & & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 1 & 1 & & 0 & 0 & 1 & & 0 & 0 & 1 \\ 0 & 1 & & 0 & 1 & 1 & & 1 & 0 & 1 \\ 0 & 0 & & 1 & 1 & 0 & & 1 & 1 & 0 \\ 1 & 0 & & 0 & 0 & 0 & & 1 & 0 & 1 \end{pmatrix} \end{array}. \tag{7}$$

Note that it is imperative to avoid the degenerate case where one of the submatrices of the partition has a single column.

Algorithm 1. ENS_CBM

Input: Binary matrix A, parameters p and $NBITER$
Output: Permutation π^* and b^* ▷ b^* is the number of 1-blocks in the best configuration π^*

```
1:  π* ← (1, 2, ..., n)                                   ▷ Let b* be the number of 1-blocks in A^(π*)
2:  iter ← 0
3:  repeat
4:     Partition matrix A^(π*) into p submatrices as explained in Section 2.4
5:     Compute length matrix Δ as in (6)
6:     Solve TSP with length matrix Δ                      ▷ Let Π be an optimal tour
7:     Remove node 0 from tour Π to obtain path π on n nodes
8:     Compute the number b of 1-blocks in A^(π)
9:     if b* > b then
10:       b* ← b
11:       π* ← π
12:    iter ← iter + 1
13: until iter = NBITER
14: output π* and b*
```

2.5. The ENS algorithm pseudo-code and an illustrative example

The proposed algorithm, called ENS_CBM, is set in pseudo-code in Algorithm 1. Observe that the neighborhood is always defined in the vicinity of the best solution at hand, as recommended in Line 4.

The two positive integer numbers, p and $NBITER$, are parameters of the algorithm. The first controls the size of the binary matrix partition and at the same time the size of the small TSP defined by the symmetric length matrix Δ, while the second controls the number of iterations.

The proposed method is iterated $NBITER$ times. If we consider a single iteration, we can see that the computational burden happens in Line 6 and consists of solving a small TSP on $2p + 1$ cities. If we insist on an optimal solution, the proposed method would be time-consuming. So, instead of solving the small TSP exactly, we content ourselves with a good approximate solution obtained using an implementation of Lin–Kernighan heuristic. Hence, the exponential neighborhood is not exhaustively explored since we accept a near local optimal solution. Clearly, Lin–Kernighan heuristic is intended here only as a problem-specific heuristic optimization algorithm. In fact, any other heuristic, either deterministic or stochastic, can be used for this purpose.

Continuing our illustrative example where the partition is proposed in (7), matrices D and Δ are as follows. We can take $M = 200$ as a reasonable penalty:

$$D = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 5 & 6 & 8 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \tag{8}$$

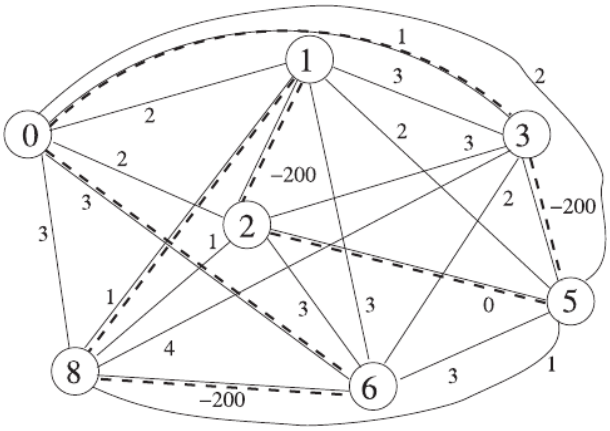

Fig. 1. Optimal tour in dotted lines.

$$\Delta = \begin{pmatrix} 0 & 2 & 2 & 1 & 2 & 3 & 3 \\ & 0 & -200 & 3 & 2 & 3 & 1 \\ & & 0 & 3 & 0 & 3 & 1 \\ & & & 0 & -200 & 2 & 4 \\ & & & & 0 & 3 & 1 \\ & & & & & 0 & -200 \\ & & & & & & 0 \end{pmatrix}.$$

By solving TSP with length matrix Δ (see Fig. 1), we get an optimal tour (0,3,5,2,1,8,6) (dotted lines in Fig. 1). By removing node 0, we obtain the path (3, 5, 2, 1, 8, 6), which indicates how to order the columns of matrix A is such a way that the number of 1-blocks is locally minimized. In our example, the local optimal solution provides a matrix configuration with only eight 1-blocks:

$$\begin{matrix} 3 & 4 & 5 & 2 & 1 & 8 & 7 & 6 \end{matrix}$$
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

2.6. *Introducing randomness to avoid getting stuck*

If we just follow our rule for the neighborhood construction in Section 2.4, the algorithm will get stuck in a local optimal solution after a small number of iterations. Indeed, once it arrives at an iteration where it is unable to find a better local optimum, it enters an infinite loop because the next neighborhood will be same as the current one. To avoid such a situation, we introduce randomness in this fashion: in each even iteration we use the rule of Section 2.4, and in each odd iteration we randomly select $p - 1$ positions where the given binary matrix is cut into p submatrices (obviously

with the requirement that no submatrix with a single column is generated). Hence, random and deterministic rules of construction of the neighborhood are followed. Intuitively the deterministic rules are the most appropriate but the random ones permit to escape from local optima.

2.7. Starting from a good solution

In our description in Section 2.5, algorithm `ENS_CBM` starts with the binary matrix as it is given, which is generally a bad configuration very far from optimal. A more reasonable approach is to start the search from a good configuration using some construction or improvement heuristic. As an alternative to the algorithm described in Section 2.5, we start with the matrix configuration induced by the chained Lin–Kernighan tour. Later we will compare the two alternative methods.

3. Experimental results

Our goal in this section is to demonstrate the efficiency of our methodology by empirical testing. However, before any experimentation, we can assert that our methodology is simple and easy to implement.

Section 3.1 presents the experimental setup. Using the polynomial-time transformation of CBM to TSP, Section 3.2 reports our experience in solving CBM exactly or in approximating it using Lin–Kernighan heuristic. At the same time, we present the results of Soares et al.'s ILS method. The detailed results of two versions of `ENS_CBM` are proposed in Section 3.3. Finally, Section 3.4 proposes to compare our heuristic with the competition and Section 3.5 tries to gain insight into CBM and the proposed approach.

3.1. Experimental setup

Our code is written in `C` on a Linux environment (`OpenSuse 13.2`) and compiled with `gcc`. The code is freely downloadable from the website https://github.com/salimhaddadi/ENS-_for_CBM. It is run on a `Dell Optiplex` i3-2120 @3.30 GHz with 2 GB RAM and tested on a benchmark dataset. This set of instances was constructed by Haddadi et al. (2015) by fixing the number of rows, the number of columns, and the matrix density, with the requirement that each column contains at least one nonzero entry and each row has at least two 1s (see the table below for more details). There are five instances of each of the nine types. It should be noted that binary matrices are sparse in practice with density no greater than 10 in general (set covering, nurse rostering, etc.). To the best of our knowledge, this dataset has been used only in the above reference as well as in Soares et al. (2020).

Instance type	A	B	C	D	E	F	G	H	I
Density (%)	2	5	10	2	5	10	2	5	10
$m \times n$		100×200			100×500			100×1000	

The computer code of Soares et al.'s ILS method (Carvalho, 2020) as well as two solvers, `Concorde` and `Linkern` (Applegate et al., 2003, 2006a, 2006b), are also run on our computer, so that all the competing methods are run on the same machine that permits a consistent comparison of computing times, all expressed in seconds.

3.2. *Using* `Concorde`, `Linkern`, *and Soares et al.'s metaheuristic to solve CBM*

As already stated in Claim 1, CBM can be polynomially transformed to TSP with symmetric length matrix W defined in (2) and can be solved exactly. The solver `Concorde` (Applegate et al., 2006a) is used for this purpose. A time limit of 7200 seconds is imposed to the solver. Table 1 provides the results where the first three columns provide, respectively, the instance name, the optimal number of 1-blocks, and the computing time. We observe that the solver runs generally quickly except for five instances for which it spends large computing times and for four instances for which it attains the time limit without finishing. We will see later that `Concorde` is in average more than 12 times slower than the proposed method.

The solver `Linkern` is also used to obtain a heuristic solution for each instance. The solver is run 20 times on each instance. Columns 4–7 of Table 1 are self-explanatory and show the results.

Soares et al. (2020) presented an ILS method exploring three $O(n^2)$-sized neighborhoods. They developed a greedy heuristic that they used for constructing a starting solution to their ILS metaheuristic. This greedy heuristic is based on three simple ingredients: a graph representation similar to what has been used in Fulkerson and Gross (1965), a depth-first search strategy, and a greedy column sequencing. Soares et al.'s metaheuristic is run 20 times on each instance. The last four columns of Table 1 provide the results, the first of which provides the number of 1-blocks in the starting solution constructed by the greedy heuristic. The results of this heuristic are very far from optimum (deviation of about 40%). The computing times of the construction heuristic are imperceptible because they are dominated by the $O(m \times n)$ time complexity of the greedy column sequencing. Finally, Soares et al.'s method is controlled with a time limit presented in the last column of Table 1.

By looking at Table 1 it appears that the solver `Linkern` is better than Soares et al.'s method from solution quality as well as from computation time point of view. The reason is that `Linkern` is also an ILS but it intelligently explores a far larger neighborhood, has variable strength perturbations and backtracking, and is implemented extremely efficiently to be able to solve very large TSP instances. In comparison, Soares et al.'s method is fairly naive. Computational results show that the solver remains a powerful and artfully designed piece of code.

3.3. *Results of two versions of ENS_CBM*

The first parameter in the metaheuristic `ENS_CBM` is the number p of submatrices in the partition of the given binary matrix. We found that the value $p = n/10$ is a good compromise between solution quality and computing time. With this choice, we will have to solve TSPs with 201 cities for the largest instances with $n = 1000$. Instead of solving them exactly, we will be content with approximate solutions obtained by applying the solver `Linkern`.

Table 1
Results of `Concorde` (Lim = 7200 seconds), `Linkern` and Soares et al.'s method

	Using `Concorde` for solving CBM		Using `Linkern` for approximating CBM (20 runs)				Soares et al.'s method (20 runs)			
Instance	Optimal solution	Computing time	Best solution	Average solution	Best time	Average time	Initial solution	Best solution	Average solution	Computing time
A1	210	1819.68	210	210.00	0.48	0.52	292	215	216.05	100
A2	216	29.57	216	216.00	0.52	0.55	296	218	221.75	
A3	204	763.44	204	204.00	0.41	0.45	278	208	209.70	
A4	233	0.88	233	233.00	0.49	0.52	305	237	238.00	
A5	226	19.49	226	226.00	0.46	0.49	311	231	232.50	
B1	632	3.59	633	634.45	0.92	0.95	817	641	642.65	
B2	629	3.64	631	632.40	0.73	0.76	816	637	640.45	
B3	619	1.73	620	622.65	0.62	0.66	793	628	630.45	
B4	638	350.15	639	640.15	0.80	0.85	830	648	649.85	
B5	613	1.51	614	614.90	0.68	0.72	812	621	622.55	
C1	1245	1.61	1248	1250.30	0.71	0.79	1630	1259	1265.55	
C2	1242	2.80	1245	1247.00	0.83	0.76	1584	1259	1262.05	
C3	1247	1366.75	1250	1251.65	0.83	0.66	1605	1262	1268.60	
C4	1253	1.41	1256	1257.75	0.79	0.85	1654	1270	1275.50	
C5	1311	1.68	1312	1312.80	0.68	0.72	1739	1328	1330.85	
D1	491	4.59	491	491.00	3.08	3.19	672	495	498.70	250
D2	502	8.89	502	502.00	3.20	3.25	672	505	507.85	
D3	445	4.68	445	445.00	3.36	3.51	574	446	450.60	
D4	472	4.46	472	472.05	3.17	3.21	648	477	480.20	
D5	454	7.75	454	454.00	3.26	3.41	600	459	461.75	
E1	1412	10.63	1423	1425.50	3.10	3.26	1951	1445	1453.95	
E2	1397	7.83	1409	1412.00	3.28	3.37	1939	1434	1438.85	
E3	1394	14.63	1404	1405.95	3.07	3.15	1948	1420	1431.95	
E4	1386	13.35	1394	1395.65	3.04	3.15	1933	1417	1424.50	
E5	1430	33.42	1442	1444.80	3.01	3.10	2012	1465	1472.95	
F1	2955	11.98	2962	2965.30	3.23	3.35	4100	3006	3018.25	
F2	3002	4.75	3013	3015.00	3.23	3.36	4121	3052	3070.50	
F3	2999	4.52	3011	3014.55	3.07	3.14	4118	3051	3063.10	
F4	3009	22.49	3021	3024.65	4.12	4.42	4145	3061	3072.50	
F5	3021	5.63	3037	3038.80	3.29	3.50	4140	3078	3092.80	
G1	893	65.84	893	895.35	10.08	10.43	1224	904	909.95	500
G2	884	20.20	885	885.65	10.25	10.40	1200	896	900.30	
G3	912	44.65	913	913.15	10.77	11.01	1234	923	926.80	
G4	932	40.72	934	935.15	10.54	10.74	1260	945	949.10	
G5	863	20.83	864	865.70	9.99	10.30	1186	874	880.10	
H1	2708	117.92	2732	2736.65	7.47	7.70	3985	2820	2834.60	
H2	2597	Lim	2622	2627.15	7.62	7.82	3829	2715	2725.35	
H3	2688	Lim	2720	2721.80	7.10	7.27	3944	2807	2815.10	
H4	2516	29.23	2550	2554.30	7.21	7.46	3748	2636	2648.20	
H5	2780	43.68	2808	2809.95	7.85	8.28	4076	2898	2907.50	
I1	5714	39.37	5763	5766.00	9.56	9.79	8136	5967	5978.55	
I2	5767	45.58	5807	5811.75	9.57	10.29	8233	6001	6018.95	
I3	5653	Lim	5694	5699.05	10.21	10.75	8023	5888	5903.05	
I4	5674	97.64	5713	5720.25	9.88	10.37	8167	5920	5934.70	
I5	5676	Lim	5721	5725.20	9.23	10.27	8108	5920	5930.50	

Because the iterations of ENS_CBM seem to spend approximately the same amount of time, it is reasonable to control the algorithm termination by fixing a maximum number $NBITER$ of iterations. In our implementation, it is fixed to 500.

We tested two versions of our method, one starting with the binary matrix as it is given and the second starting from the configuration induced by the tour provided by Linkern. Table 2 displays the results. Each version is run 20 times on each instance.

The second column of the table provides the number of 1-blocks of the initial configuration (i.e., the number of 1-blocks in the binary matrix as it is given). This value deviates from optimal by about 67%. The other columns are self-explanatory. By the way, the starting solutions in the second version of algorithm ENS_CBM are those that are displayed in Table 1 under the label "Using Linkern for approximating CBM (20 runs)."

From the results in Table 2, it can be seen that the two versions consume about the same computing time. However, the second version, which starts from Lin–Kernighan tour, provides slightly better solutions since it deviates only by about 0.34% from the optimum while the first version deviates by about 0.43%. In fact, this slight difference between the two versions in terms of average deviation from optimal tends to demonstrate the robustness of the algorithm ENS_CBM. Another fact worth noting is that, in anyone of the two versions, the best solution does not deviates much from average.

3.4. Comparison of ENS-CBM *with the competition*

Since the second version of algorithm ENS_CBM provides better quality solutions, it will be used for the comparison with the competing methods: Concorde, Linkern, and Soares et al.'s ILS method. The comparison concerns two issues, the average deviation from optimal and the average computing time. For the ease of comparison, mean values are computed for each type of instances. The results are summarized in Table 3, where the best values are in bold. The last row shows that

- Concorde, while proving optimal solution (except for the four instances for which the solver attains the time limit), spends long if not prohibitive computing times in average. We can also see that is in average more than 12 times slower than ENS_CBM.
- Soares et al.'s method finds less quality solutions than Linkern and is more than 60 times slower.
- Not surprisingly, algorithm ENS_CBM finds solutions that deviate from the best two times less than Linkern but is about 36 times slower.
- Linkern is the best choice as far as the binary matrices are very sparse (2%) or if we want to obtain good solutions very quickly.
- Our method is recommended if we need better quality solutions albeit at much greater, but reasonable, computing times.

3.5. How does the matrix density affect problem-solving ?

It depends on the perspective from which we look at the question.

Table 2
Detailed results of our method

	Starting from initial configuration (20 runs)					Starting from Lin–Kernighan tour (20 runs)			
Instance	Initial solution	Best solution	Average solution	Best time	Average time	Best solution	Average solution	Best time	Average time
A1	402	210	211.15	14.31	14.73	210	210.00	13.64	14.18
A2	418	216	217.25	13.72	14.28	216	216.00	13.00	13.69
A3	407	204	205.20	12.09	13.06	204	204.00	11.54	12.45
A4	430	233	234.20	13.83	14.48	233	233.00	13.59	14.05
A5	423	227	227.35	14.04	14.73	226	226.00	13.27	13.93
			0.00%				0.00%		
B1	952	634	635.05	30.98	32.13	633	633.75	31.20	32.99
B2	960	632	633.15	30.58	31.35	631	631.70	29.58	30.86
B3	941	622	622.65	27.73	28.94	620	621.40	27.04	28.84
B4	968	640	641.25	27.24	28.68	638	639.55	27.99	29.36
B5	957	614	615.70	27.90	28.95	613	614.00	26.60	27.94
			0.54%				0.30%		
C1	1761	1250	1252.00	26.73	27.89	1248	1249.60	26.89	27.70
C2	1743	1248	1248.30	26.72	27.63	1245	1246.25	26.37	27.86
C3	1797	1252	1253.55	26.69	27.51	1249	1250.80	25.56	26.99
C4	1790	1258	1259.90	24.41	25.70	1256	1256.45	23.61	24.47
C5	1856	1314	1315.25	21.90	23.20	1312	1312.55	21.25	22.86
			0.49%				0.28%		
D1	1067	491	491.20	88.99	94.87	491	491.00	93.27	97.83
D2	1065	502	502.00	90.14	93.88	502	502.00	85.70	90.21
D3	972	445	445.05	88.37	91.79	445	445.00	92.69	96.02
D4	1034	472	472.10	79.84	83.00	472	472.00	82.49	84.75
D5	998	454	454.05	85.23	87.36	454	454.00	87.31	89.45
			0.02%				0.00%		
E1	2357	1422	1424.15	123.23	125.94	1420	1422.15	120.81	123.93
E2	2354	1408	1410.45	125.83	128.19	1406	1408.55	122.03	126.11
E3	2370	1402	1404.80	122.12	124.59	1399	1402.60	118.31	122.37
E4	2345	1393	1396.20	116.65	120.63	1391	1393.60	115.35	119.61
E5	2445	1441	1443.45	117.26	119.31	1440	1441.80	114.89	116.88
			0.85%				0.71%		
F1	4504	2961	2965.40	107.08	109.64	2959	2961.20	106.42	108.53
F2	4510	3011	3016.10	107.24	109.69	3010	3012.25	103.37	107.45
F3	4554	3008	3012.35	103.02	110.41	3006	3009.80	101.18	104.56
F4	4546	3017	3021.20	141.72	142.89	3017	3020.05	140.68	144.82
F5	4532	3034	3036.85	110.20	116.59	3032	3032.95	109.27	115.32
			0.44%				0.33%		
G1	2099	893	893.40	379.12	384.59	893	893.40	384.01	390.36
G2	2080	884	884.65	384.76	389.41	884	884.35	387.64	397.54
G3	2103	912	912.35	395.11	402.24	912	912.25	404.44	409.67
G4	2138	932	932.30	380.75	392.63	932	933.15	376.05	392.50
G5	2064	863	863.75	369.46	376.83	863	863.20	378.08	384.40
			0.05%				0.05%		

Continued

Table 2
Continued

Instance	Starting from initial configuration (20 runs)					Starting from Lin–Kernighan tour (20 runs)			
	Initial solution	Best solution	Average solution	Best time	Average time	Best solution	Average solution	Best time	Average time
H1	4856	2722	2726.60	350.06	356.47	2721	2723.60	344.35	348.89
H2	4695	2616	2619.00	346.12	352.22	2614	2616.75	339.16	346.12
H3	4818	2709	2711.45	356.02	364.03	2707	2709.85	345.75	347.05
H4	4614	2541	2543.40	333.90	343.60	2538	2541.10	335.12	340.40
H5	4909	2794	2798.55	354.68	358.75	2793	2796.75	350.89	355.59
			0.83%				0.75%		
I1	9061	5753	5756.95	347.89	356.08	5751	5753.65	342.68	355.63
I2	9082	5797	5800.55	341.62	347.84	5795	5800.45	340.53	346.11
I3	8931	5684	5688.50	340.99	345.64	5681	5685.10	339.29	345.43
I4	8966	5709	5711.40	339.31	347.72	5704	5708.25	333.20	342.25
I5	8954	5711	5714.10	355.45	360.03	5708	5714.70	351.81	357.60
			0.66%				0.63%		
Global deviation from optimal			0.43%				0.34%		

Table 3
Comparison of the competing methods (average values are given for each instance type)

Instance type	Concorde	Linkern		Soares et al.'s ILS		Our method	
	Computing time	Average deviation from optimal (%)	Average computing time	Aver. deviation from optimal (%)	Average computing time	Aver. deviation from optimal (%)	Average computing time
A	526.61	**0.00**	**0.51**	2.67	100.00	**0.00**	13.66
B	72.12	0.43	**0.79**	1.75	100.00	**0.30**	30.00
C	274.85	0.34	**0.76**	1.66	100.00	**0.28**	25.98
D	6.07	**0.00**	**3.31**	1.49	250.00	**0.00**	91.65
E	15.97	0.92	**3.21**	2.89	250.00	**0.71**	121.78
F	9.87	0.48	**3.55**	2.21	250.00	**0.33**	116.14
G	38.45	0.24	**10.58**	1.83	500.00	**0.05**	394.89
H	>2918.17	1.22	**7.71**	4.84	500.00	**0.75**	347.61
I	>2916.52	0.84	**10.29**	4.50	500.00	**0.63**	349.40
Global average	>2052.19	0.50	**4.52**	2.65	283.33	**0.34**	165.68

1. How does the matrix density affect the problem difficulty regardless of the method used? This question is rather general and it is not easy to answer it. I think it falls within the purview of parameterized complexity theory. What we can say is that, in general, denser instances are harder to solve and necessitate larger computing times. This property has been observed in many combinatorial optimization with binary constraint matrices, such as set covering instances.
2. How does the matrix density affect the efficiency of the proposed method?

This is a specific question and its answer is in the careful examination of the computational results.

- Let us first consider computing time.
  The first important observation is from Table 1, which indicates that `Linkern` is insensitive to the matrix density. Indeed, instances with fixed number of columns and with varied densities need about the same amount of computing time in average. Since `Linkern` is the most important component of our method, and the most time-consuming, we can conclude that `ENS_CBM` is itself almost insensitive to the matrix density as it can be verified in the last column of Table 3.
- Now, let us consider solution quality.
  Here, the proposed method seems to be heavily affected. Indeed we note in the penultimate column of Table 3 that the larger the density, the larger is the deviation of the proposed method from best.

4. Conclusion

We studied CBM, a hard combinatorial optimization problem related to binary matrices. Despite its practical significance, we do not know of computational methods for CBM other than an ILS metaheuristic (Soares et al., 2020). The work realized in this paper aims to fill this particular gap by proposing an ENS metaheuristic. Searching the exponential neighborhood amounts to solving a small TSP using an implementation of Lin–Kernighan heuristic (Applegate et al., 2006a).

The experimental results carried on a benchmark dataset and the comparison of our results with three alternative approaches show that the proposed method provides the best quality solutions (if we except `Concorde`, which is an exact solver), although it is dominated by `Linkern` in terms of computation time. The proposed method added to the benchmark dataset will facilitate the evaluation of future research techniques.

TSP instances defined in (2) from general binary matrices are hard. As a direction for future research, however, we identified an interesting open question: Are TSP instances defined in (2) from C1P binary matrices easy? A greedy acceptance strategy is used in our method and hence diversification was neglected. The second issue worth exploring is to balance between intensification and diversification by considering accepting worsening solution as in simulated annealing. Another promising possibility is to introduce a bias in the search, like in ILS, by perturbing the current best solution and defining the exponential neighborhood in its vicinity.

Acknowledgments

We wish to thank D. Applegate (AT&T Labs-Research), R. Bixby (ILOG and Rice University), V. Chvatal (Concordia University), and W. Cook (University of Waterloo) for their kind permission to use their TSP solvers. We would also like to thank M.A.M. Carvalho for sending us the link (which can be found below) to the computer code of Soares et al.'s ILS meta-heuristic.

References

Applegate, D., Bixby, R., Chvatal, V., Cook, W., 2006a. Concorde TSP solver. Available at http://www.math.uwaterloo.ca/tsp/concorde.html (accessed February 2017).

Applegate, D., Bixby, R., Chvatal, V., Cook, W., 2006b. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ.

Applegate, D., Cook, W., Rohe, A., 2003. Chained Lin–Kernighan for large traveling salesman problems. *INFORMS Journal on Computing* 15, 1, 82–92.

Becceneri, J.C., Yanasse, H.H., Soma, N.Y., 2004. A method for solving the minimization of open stacks. *Computers & Operations Research* 31, 14, 2315–2332.

Bernardino, R., Paias, A., 2021. Heuristic approaches for the family traveling salesman problem. *International Transactions in Operations Research* 28, 262–295.

Carvalho, M.A.M., 2020. Source code. Available at https://github.com/MarcoCarvalhoUFOP/ILS-CBM (accessed June 2020).

Carvalho, M.A.M., Soma, N.Y., 2015. A breadth-first search applied to the minimization of the open stacks. *Journal of the Operational Research Society* 66, 6, 936–946.

Dyson, R., Gregory, A., 1974. The cutting stock problem in the flat glass industry. *Journal of the Operational Research Society* 25, 1, 41–53.

Fulkerson, D., Gross, O., 1965. Incidence matrices and interval graphs. *Pacific Journal of Mathematics* 15, 3, 835–855.

Haddadi, S., 2002. A note on the NP-hardness of the consecutive block minimization problem. *International Transactions in Operations Research* 9, 6, 775–777.

Haddadi, S., Chenche, S., Guessoum, F., Cheraitia, M., 2015. Polynomial-time local-improvement algorithm for consecutive block minimization. *Information Processing Letters* 115, 6–8, 612–617.

Haddadi, S., Layouni, Z., 2008. Consecutive block minimization is 1.5-approximable. *Information Processing Letters* 108, 3, 132–135.

Johnson, D., Krishnan, S., Chhugani, J., Kumar, S., Venkatasubramanian, S., 2004. Compressing large boolean matrices using reordering techniques. Proceedings of the 30th International Conference on Very Large Data Bases, VLDB Endowment, Toronto, Canada, pp. 13–23.

Kou, L.T., 1977. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing* 6, 1, 67–75.

Lemire, D., Kaser, O., 2011. Reordering columns for small indexes. *Information Sciences* 181, 2550–2570.

Lima, J.R., Carvalho, M.A.M., 2017. Descent search approaches applied to the minimization of open stacks. *Computers & Industrial Engineering* 112, 175–186.

Linhares, A., Yanasse, H.H., 2002. Connections between cutting-pattern sequencing, VLSI design, and flexible machines. *Computers & Operations Research* 29, 12, 1759–1772.

Madsen, O.B., 1979. Glass cutting in a small firm. *Mathematical Programming* 17, 1, 85–90.

Madsen, O.B., 1988. An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry. *Journal of the Operational Research Society* 39, 3, 249–256.

Paiva, G.S., Carvalho, M.A.M., 2017. Improved heuristic algorithms for the job sequencing and tool switching problems. *Computers & Operations Research* 88, 208–219.

Soares, L.C.R., Reinsma, J.A., Nascimento, L.H.L., Carvalho, M.A.M., 2020. Heuristic methods to consecutive block minimization. *Computers & Operations Research* 120. https://doi-org.ez28.periodicos.capes.gov.br/10.1016/j.cor.2020.104948.