Computers & Operations Research 131 (2021) 105273

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Iterated local search for consecutive block minimization

Salim Haddadi

LabSTIC, Université 8 Mai 1945, Guelma, Algeria

ARTICLE INFO

Keywords:
Consecutive block minimization
Iterated local search
Large-scale neighborhood
Traveling salesman problem solver
2008 MSC:
90C27
90C59

ABSTRACT

A 1-block in a binary matrix is any maximal sequence of 1s occurring consecutively in the same row. Consecutive block minimization (CBM) requires a permutation of the matrix columns that minimizes the number of 1-blocks. CBM is NP-hard for matrices with a maximum of two 1s per row and an arbitrary number of 1s per column. First, we prove that it is NP-hard even when it is restricted to binary matrices having a maximum of two 1s per row and a maximum of three 1s per column. In these circumstances, a reasonable and traditional approach to solve the problem is to design a heuristic capable of finding good configurations within acceptable computation times. This paper proposes an Iterated Local Search (ILS) metaheuristic. A binary matrix satisfies the strong consecutive ones property (is strongly C1P) if its non-zero entries occur consecutively in each row. It is C1P if there is a permutation of its columns such that the resulting matrix is strongly C1P. In this study, two necessary conditions for a binary matrix to be C1P are provided, each allowing to define a neighborhood for local search (LS). In addition, a third large-scale neighborhood is proposed where the local optimal solution can be directly obtained by solving a Traveling Salesman Problem (TSP). It is noteworthy that the three neighborhoods are being proposed for the first time. The ILS algorithm is tested on a dataset from the OR-library, composed of thirty large-scale set covering instances of varying sizes and densities. Comparison with two TSP solvers as well as with a recently published metaheuristic, from both accuracy and computing time points of view, allows to conclude that the proposed method competes favorably with the methods available in the literature.

1. Introduction

Binary (or 01) matrices are ubiquitous in Computer Science and in Discrete Mathematics. A binary $m \times n$-matrix can be used to encode a family of m subsets of a finite ground set of n elements. An example of using a binary matrix arises in the context of consecutive retrieval file organization where a set of n records must be stored on a linear storage media and where m queries must be satisfied. Assuming that we can unambiguously decide the pertinence of any record to any query, we construct a binary $m \times n$-matrix A as follows:

$$a_{ij} = \begin{cases} 1 & \text{if record } j \text{ is relevant to query } i \\ 0 & \text{otherwise} \end{cases}$$

Consecutive retrieval file organization exists if queries have the consecutive retrieval property for records (*i.e.*, for each query, the pertinent records can be stored in consecutive storage locations). Let A be a binary matrix and let π be a permutation of the columns of A. We denote by $A^{(\pi)}$ the matrix induced by the permutation π. We say that A is strongly C1P if the 1s occur consecutively in every row. It is C1P if there is a permutation π of its columns so that $A^{(\pi)}$ is strongly C1P. Consider the decision problem (referred to as C1P-decision) which, given the binary matrix A as instance, asks if it is C1P. Fulkerson and Gross (1965) were the first to prove that C1P-decision is polynomially solvable. Since then, many attempts have been made to design faster recognition algorithms. The first linear-time algorithm is due to Booth and Lueker (1976), while McConnell (2004) proposed an $O(n)$-size certificate for non C1P matrices.

In fact, binary matrices are very rarely C1P. So, which is the best solution in these circumstances? One way is to lessen the requirement that the 1s occur consecutively in every row, for example by seeking a column permutation that makes A as 'close' as possible to C1P. A 1-block in a binary matrix is any maximal sequence of consecutive 1s in the same row. For example, the following matrix has five 1-blocks.

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

Consider the combinatorial optimization problem CBM which, given the binary matrix A as instance, requires a permutation π of its columns so that the number of 1-blocks in $A^{(\pi)}$ is minimized. CBM is NP-hard

E-mail address: haddadi.salim@univ-guelma.dz.

https://doi.org/10.1016/j.cor.2021.105273
Received 3 May 2020; Received in revised form 20 February 2021; Accepted 1 March 2021
Available online 8 March 2021

(Kou, 1977). According to common usage, (a, b)-matrices are binary matrices with a maximum of a 1s per column and a maximum of b 1s per row. Haddadi (2002) showed that CBM is NP-hard on $(*, 2)$-matrices where * means an arbitrary number of 1s. However, CBM can be approximated within a fixed guarantee. Indeed, Haddadi and Layouni (2008) constructed a polynomial-time heuristic such that the number of 1-blocks in the solution does not differ from optimal by more than 50%. An important property of C1P matrices is that they are totally unimodular. Therefore, combinatorial optimization problems with C1P constraint matrices are solved in polynomial time. Haddadi (2015) exploited this idea to solve the Set Covering problem with an 'almost C1P' binary matrix by Benders decomposition.

CBM has great practical significance. As a model, it has been used in scheduling (Linhares and Yanasse, 2002; Paiva and Carvalho, 2017), in production (Becceneri et al., 2004; Carvalho and Soma, 2015; Lima and Carvalho, 2017), in the glass industry (Dyson and Gregory, 1974; Madsen, 1979; Madsen, 1988), and in large-scale data compression (Johnson et al., 2004; Lemire and Kaser, 2011). Despite the practical relevance of CBM, only two algorithms are available in the literature: a polynomial-time LS procedure based on interchanging or moving columns (Haddadi et al., 2015) and a recently published ILS metaheuristic (Soares et al., 2020). In practice, there can be huge real-world CBM instances. For example, Johnson et al. (2004) presented a lossless compression strategy for efficiently storing and accessing very large matrices on disk. The number of rows or columns in the binary matrix is of the order of a million. It goes without saying that such instances cannot be solved exactly, nor can they be approximated by Lin-Kernighan-type methods. Since CBM can be reduced to TSP in polynomial time, the sophisticated approach of Johnson et al. (2004) splits the columns into several subsets, solves TSP on each submatrix by using elementary heuristics and then merges the resulting solutions. In fact, most of the published methods solve CBM via a polynomial-time transformation to TSP. In doing so, we are far from understanding and exploiting the combinatorial properties of CBM. By designing a specialized method, this study aims to understand its combinatorial properties, especially with regard to the neighborhood structures. In fact, these aspects are more important for us than the focus only on the running time or on the value of the solution.

First, we demonstrate that CBM remains NP-hard when the domain of its instances is even more restricted. Let $(3,2)$-CBM be the restriction of CBM to $(3,2)$-matrices. In Section 2, we prove that $(3,2)$-CBM is NP-hard. Next, an ILS method is proposed for CBM. The theoretical results obtained suggest the exploration of two new neighborhoods. A large-scale interesting neighborhood is also proposed, whose exploration amounts to solving a TSP. The ILS is tested on a dataset composed of thirty set covering instances from the OR-library (Beasley, 1990). Section 6 presents the experiments carried out with the proposed method, and the ILS results are compared with those obtained by two TSP solvers and the metaheuristic proposed by Soares et al. (2020). It imposes no loss of generality to assume that binary matrices have no null columns and no null rows. Moreover, since in our context rows with a single 1 are irrelevant, we assume that binary matrices have at least two 1s per row.

2. $(3,2)$-CBM is NP-hard

A simple graph is a graph without loops and without parallel edges. Let 3-HP be the Hamiltonian Path Problem restricted to simple graphs with a node degree not greater than 3. It is well-known that 3-HP is NP-complete (Garey et al., 1974). Consider a simple graph G with n nodes and m edges and with a node degree at most equal to 3. Let binary matrix A be the transpose of the node-edge incidence matrix of G. It is clear that A is a binary $(3,2)$-matrix having m rows and n columns. Moreover, A does not have identical rows, otherwise there would be parallel edges in G, which is impossible since G is simple. Since each 1-block in A contains either a single 1 or two 1s, for future reference we will call 2-ones a 1-block containing two 1s.

Lemma 1. *For any permutation π of the columns of A, the number of 2-ones in $A^{(\pi)}$ is at most equal to $n-1$.*

Proof. Suppose that for a permutation $\pi, A^{(\pi)}$ contains n 2-ones, then there must be a row i_1 such that $a^{(\pi)}_{i_1 1} = 1$ and $a^{(\pi)}_{i_1 2} = 1$ (which corresponds to the first 2-ones). Similarly, there must be a row i_2 such that $a^{(\pi)}_{i_2 2} = 1$ and $a^{(\pi)}_{i_2 3} = 1$ (and this is the second 2-ones). We continue until we find row i_{n-1} such that $a^{(\pi)}_{i_{n-1} n-1} = 1$ and $a^{(\pi)}_{i_{n-1} n} = 1$ (and this is the 2-ones number $n-1$). Clearly, any additional 2-ones results in two identical rows, which is prohibited as already discussed. □

Lemma 2. *The graph G is Hamiltonian if and only if there is a permutation π of the columns of A such that the number of 2-ones in $A^{(\pi)}$ is $n-1$. Moreover, π is a Hamiltonian path in G.*

Proof. $\Rightarrow$) Suppose graph G is Hamiltonian and let π be a Hamiltonian path. Let A be the transpose of the node-edge incidence matrix of G and let $A^{(\pi)}$ be the matrix resulting from the permutation π of the columns of A defined by the Hamiltonian path π. The $n-1$ rows of $A^{(\pi)}$ associated to the edges of the Hamiltonian path contain a 2-ones each. Hence, the number of 2-ones in $A^{(\pi)}$ is at least $n-1$. From Lemma 1, it follows that the number of 2-ones in $A^{(\pi)}$ is $n-1$.

$\Leftarrow$) Suppose there is a permutation π such that the number of 2-ones in $A^{(\pi)}$ is $n-1$. Call H the subgraph of G induced by the edges associated to the $n-1$ 2-ones. We then show that H is a Hamiltonian path. First, we show that the subgraph H is acyclic. Suppose k of the $n-1$ edges of H form a cycle, $k \geqslant 3$. Let $j_1, \cdots, j_k$ be the sequence of nodes (corresponding to columns $a^{(\pi)}_{j_1}, \cdots, a^{(\pi)}_{j_k}$ of $A^{(\pi)}$) that form the cycle of cardinality k. Hence, columns $a^{(\pi)}_{j_1}$ and $a^{(\pi)}_{j_2}$ contain a 2-ones since nodes j_1 and j_2 are adjacent. In the same manner, columns $a^{(\pi)}_{j_2}$ and $a^{(\pi)}_{j_3}$ contain a second 2-ones, and so on, until columns $a^{(\pi)}_{j_{k-1}}$ and $a^{(\pi)}_{j_k}$ contain the 2-ones number $k-1$. Now nodes j_1 and j_k are adjacent but do not constitute a 2-ones; this is a contradiction since the edge $j_1 j_k$ belongs to H. Therefore, there cannot be any k edges corresponding to 2-ones that form a cycle. Since H contains $n-1$ edges and is acyclic it is a tree. To prove that H is a Hamiltonian path, we still have to show that each node in H has a degree of 2 at most. Suppose there is a node of degree 3 in H. This means that this node is associated to three 2-ones which is impossible unless there are parallel edges. But there can be no parallel edges in H since it is a subgraph of a simple graph and is therefore itself simple. □

Theorem 1. *3-HP $\propto (3,2)$-CBM.*

Proof. The transformation from 3-HP to $(3, 2)$-CBM is trivially polynomial-time since it takes only binary $(3,2)$-matrix A (which is the transpose of the node-edge incidence matrix of the graph G which has n nodes, m edges and a node degree not greater than 3) as an instance of 3-HP and consider it simply as an instance of $(3,2)$-CBM. Since the number of 1s in A is $2m$, the number of 1-blocks in A is $2m-k$ if and only if the number of 2-ones in A is $k, k \leqslant n-1$. Hence, there is a permutation π of the columns of A such that the number of 1-blocks in $A^{(\pi)}$ is $2m-n+1$ if and only if the number of 2-ones in $A^{(\pi)}$ is $n-1$. According to Lemma 2, the number of 2-ones in $A^{(\pi)}$ is $n-1$ if and only if π is a Hamiltonian path in G. It follows that there is a permutation π of the columns of A such that the number of 1-blocks in $A^{(\pi)}$ is $2m-n+1$ if and only if π is a Hamiltonian path in the graph G considered as instance of 3-HP whose edge-node incidence matrix is the transpose of A. Consequently $(3,2)$-CBM is at least as hard as 3-HP. □

3. Preliminary results

Given binary $m \times n$-matrix A, let

$$B = (0|A|0) \tag{1}$$

be the binary $m \times (n+2)$-matrix obtained by concatenating to A two null columns in position number 0 and number $n+1$. These two columns are artificial and their position never changes. We need them because we may choose to insert a column to the left of the leftmost column of A (*i.e.,* between the artificial column number 0 and column number 1), and we may also want to insert a column to the right of the rightmost column of A (between column number n and the artificial column number $n+1$). In fact, as explained in this study, these artificial columns are useful to evaluate the contribution of the inserted columns in terms of the number of 1-blocks. Thus, A is C1P if and only if B is. Furthermore, it can be observed that CBM on instance A is equivalent to CBM on instance B. In other words, permutation π^* is optimal for A if and only if $\Pi^* = (0, \pi^*, n+1)$ is optimal for B. Hence, as far as CBM is concerned, we can work with either A or B matrices. Let U be the $m \times (n+2)$-matrix whose entries are all 1s. Consider the symmetric square (Hamming distance) matrix of order $n+2$ as:

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

We recall that CBM can be solved exactly via TSP instance W. Let Π be an optimal tour. Remove nodes 0 and $n+1$ (which correspond to the columns numbered 0 and $n+1$ concatenated to A and which are adjacent in every optimal tour) to obtain a path π on n nodes (which can be considered as a permutation of the columns of A).

Theorem 2. (*Haddadi and Layouni, 2008*) *There is a permutation π of the columns of A such that $A^{(\pi)}$ contains a minimum number μ of 1-blocks if and only if TSP defined with the distance matrix W has an optimal tour Π of length 2μ.*

Lemma 3 is easy to prove.

Lemma 3. *Let a, b, c be binary numbers and let $\alpha = a(1-b) + (1-a)b + b(1-c) + (1-b)c - a(1-c) - (1-a)c$. The number $\alpha/2$ is binary and its value is 1 if and only if $a = 1, b = 0, c = 1$ or $a = 0, b = 1, c = 0$ (hence, the value α recognizes the strings 101 and 010).*

Lemma 4. *For every three adjacent columns b_{j-1}, b_j, b_{j+1} of $B, 1 \leqslant j \leqslant n$, the number*

$$\delta(j-1, j, j+1) = w_{j-1,j} + w_{jj+1} - w_{j-1,j+1} \tag{3}$$

is nonnegative, even, and $\delta(j-1, j, j+1)/2$ counts the number of occurrences of 101 and 010 in the three columns.

Proof. Recall from the definition of matrix W in (2) that $w_{j-1,j} = b_{j-1}^T(e - b_j) + (e - b_{j-1})^T b_j$ where T indicates the transposition operation and where e is a vector whose components are all 1s. The proof derives from Lemma 3. □

Consequently, column b_j contributes to $\delta(j-1, j, j+1)/2$ 1-blocks in the three adjacent columns. Now, instead of the global view of Theorem 2, let us consider matrix B locally by giving a necessary condition to be C1P.

Theorem 3. *If B is C1P then there is a permutation π of its columns such that for every three adjacent columns $b_{j-1}^{(\pi)}, b_j^{(\pi)}, b_{j+1}^{(\pi)}$ in $B^{(\pi)}$ we have*

$$\delta(j-1, j, j+1) = 0, \quad 2 \leqslant j \leqslant n-1 \tag{4}$$

(note that columns number 0 and number $n+1$ are not concerned by the permutation, and that Eq. (4) is trivially satisfied if $j = 1$ or $j = n$).

Proof. Call $B^{(j)}$ the submatrix formed by the three adjacent columns. The sequence 101 cannot occur in any row of $B^{(j)}$ since it contradicts C1P. Furthermore, according to our assumption that every row has at least two 1s, the sequence 010 cannot occur either. Based on Lemma 4, these cases are precisely forbidden by Eq. (4). □

The necessary condition of Theorem 3 is unfortunately not sufficient. The following result is the most important for our purposes as it suggests a LS heuristic (more details in Section 5).

Corollary 1. *By removing column b_j from between columns b_{j-1} and b_{j+1} and inserting it between columns b_{k-1} and b_k, the number of 1-blocks in matrix B decreases if and only if $\Delta = \delta(j-1, j, j+1) - \delta(k-1, j, k) > 0$. Furthermore, the number of 1-blocks decreases by $\Delta/2$.*

As an example, consider matrix A which contains seven 1-blocks. Next, form matrix B defined in Eq. (1) and compute the Hamming distance matrix W as in Eq. (2).

$$A = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \quad B = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array} \quad W = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 4 & 3 & 5 & 6 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 2 & 2 & 2 & 2 & 0 \\ 1 & 0 & 1 & 3 & 3 & 1 & 1 \\ 2 & 1 & 0 & 4 & 2 & 2 & 2 \\ 2 & 3 & 4 & 0 & 2 & 2 & 2 \\ 2 & 3 & 2 & 2 & 0 & 4 & 2 \\ 2 & 1 & 2 & 2 & 4 & 0 & 2 \\ 0 & 1 & 2 & 2 & 2 & 2 & 0 \end{pmatrix} \end{array} \tag{5}$$

Compute $\delta(2, 3, 4) = w_{23} + w_{34} - w_{24} = 4 + 2 - 2 = 4$ and $\delta(4, 3, 5) = w_{34} + w_{35} - w_{45} = 2 + 2 - 4 = 0$. Hence, if we remove column 3 from between columns 2 and 4 and insert it between columns 4 and 5, the number of 1-blocks decreases by $\Delta/2 = 2$. Indeed, the new matrix A' has only five 1-blocks.

Table 1
Computing the value $\alpha/2$ of Lemma 5.

abcd	$+$ $a(1-b) + (1-a)b$	$+$ $b(1-c) + (1-b)c$	$+$ $c(1-d) + (1-c)d$	$-$ $a(1-d) + (1-a)d$	$\alpha/2$
0000	0	0	0	0	0
0001	0	0	1	1	0
0010	0	1	1	0	**1**
0011	0	1	0	1	0
0100	1	1	0	0	**1**
0101	1	1	1	1	**1**
0110	1	0	1	0	**1**
0111	1	0	0	1	0
1000	1	0	0	1	0
1001	1	0	1	0	**1**
1010	1	1	1	1	**1**
1011	1	1	0	0	**1**
1100	0	1	0	1	0
1101	0	1	1	0	**1**
1110	0	0	1	1	0
1111	0	0	0	0	0

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

The previous theorem, lemmas and corollary related to the value δ are generalizable to an arbitrary number k of columns provided that the binary matrix A has at least $k-1$ 1s per row. Let us consider the useful case with $k = 4$ assuming that the binary matrices have at least three 1s per row.

Lemma 5. *Let a, b, c, d be four binary numbers and let*

$$\alpha = a(1-b) + (1-a)b + b(1-c) + (1-b)c + c(1-d) + (1-c)d - a(1-d) - (1-a)d.$$

The number $\alpha/2$ is binary and its value is 1 if and only if the string bc contributes to a new 1-block when inserted between binary numbers a and d in the string abcd.

Proof. The proof can be easily provided based on Table 1. Strings in bold contain a new 1-block created by the insertion of string *bc* between binary numbers *a* and *d*. They are identified by the value $\alpha/2 = 1$. Recall that, according to our assumption, binary matrices have at least three 1s per row, so that the sequences $0100, 0010, 0110$ contain a 1-block each. More precisely, by inserting the string *bc* between binary numbers *a* and *d* either no 1-blocks are created or only one is created (case in bold). □

The proof of Lemma 6 is analogous to that of Lemma 4.

Lemma 6. *For every four adjacent columns $b_{j-1}, b_j, b_{j+1}, b_{j+2}$ of B, the number*

$$\delta'(j-1, j, j+1, j+2) = w_{j-1,j} + w_{j,j+1} + w_{j+1,j+2} - w_{j-1,j+2} \tag{6}$$

is nonnegative, even, and $\delta'(j-1, j, j+1, j+2)/2$ counts the number of 1-blocks contributed by the two internal columns b_j and b_{j+1}. Thus, we have another necessary condition for a binary matrix to be C1P whose proof comes from Lemmas 5 and 6.

Theorem 4. *If B is C1P then there is a permutation π of its columns such that for every four adjacent columns $b_{j-1}^{(\pi)}, b_j^{(\pi)}, b_{j+1}^{(\pi)}, b_{j+2}^{(\pi)}$ in $B^{(\pi)}$ we have*

$$\delta'(j-1, j, j+1, j+2) = 0, \quad 2 \leqslant j \leqslant n-2$$

Theorem 4 has two interesting corollaries depending on whether the two columns we want to move are adjacent or not.

Corollary 2. *Suppose that we remove (adjacent) columns b_j, b_{j+1} from between columns b_{j-1} and b_{j+2} and insert them between columns b_{r-1} and b_r. Then the number of 1-blocks in matrix A decreases if and only if $\Delta = \delta'(j-1, j, j+1, j+2) - \delta'(r-1, j, j+1, r) > 0$. Moreover, the number of 1-blocks decreases by $\Delta/2$.*

Corollary 3. *Suppose that columns b_j and b_k are not adjacent (i.e. $k \neq j-1$ and $k \neq j+1$). Suppose that we remove column b_j from between columns b_{j-1} and b_{j+1}, and we remove column b_k from between columns b_{k-1} and b_{k+1}. Next, we insert the two columns b_j and b_k in this order between columns b_{r-1} and b_r. Then, the number of 1-blocks in matrix A decreases if and only if $\Delta = \delta(j-1, j, j+1) + \delta(k-1, k, k+1) - \delta'(r-1, j, k, r) > 0$. Moreover, the number of 1-blocks decreases by $\Delta/2$.*

We illustrate Corollary 2 by considering the matrix A in Example (5) which has seven 1-blocks. We want to remove columns 3 and 4 and insert them between columns 5 and 6. Hence, we compute $\delta'(2, 3, 4, 5) = 8$ and $\delta'(5, 3, 4, 6) = 4$ so that $\Delta/2 = 2$. Indeed, the new configuration A'' contains $7 - 2 = 5$ 1-blocks.

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

4. LS and ILS: An overview

This outline is inspired by the comprehensive tutorials (Lourenço et al., 2010; Stützle et al., 2018) presenting the implementation issues of ILS and emphasizing its efficiency, confirmed by numerous papers published in the last decade or so.

LS is a natural and basic tool to solve combinatorial optimization problems of the form: find $s^* \in S$ such that

$$f(s^*) = \min_{s \in S} f(s)$$

where S is a finite set whose elements are called solutions and where f is a function defined over S. LS starts with an initial solution s_0. A neighborhood $N(s_0)$ of s_0 is defined in which the search for a better solution s_1 is conducted. Generally, the neighborhood $N(s_0)$ of s_0 is the set of all solutions that can be obtained from s_0 by a small 'perturbation'. The search is iterated by looking for $s_2, s_3, \cdots$ until s_k is found so that there is no better solution in its vicinity. Therefore $\sigma = s_k$ is a local minimum of the function f. The most important issue in LS is the neighborhood structure, which is problem-dependent. In contrast with more elaborated search techniques, LS is parameter-free, but it has a serious shortcoming. Due to its myopic nature, it has no way of escaping the attraction basin of the local optimum. One way to overcome this drawback is to repeat the search from another starting solution. General as well as empirical arguments (Lourenço et al., 2010) argue against random restart with independent samplings. Better results are achieved if a bias is introduced. This is precisely the aim of ILS.

ILS is a high-level heuristic designed to drive a low-level LS heuristic to find good solutions to an optimization problem by sampling the solution set that is too large to be completely searched. The basic ILS scheme, given as pseudo-code in Algorithm 1, can be summarized as follows: the current local optimum σ is perturbed to reach an intermediate solution s'; LS is then applied to s' and obtains a new local optimum σ'. If the latter passes the acceptance test, it becomes the next solution to perturb; otherwise, we return to the previous local optimum σ. These steps are repeated until a termination criterion is met.

Algorithm 1. ILS skeleton

```
1: s0 ← initial_solution()
2: σ ← local_search(s0)
3: repeat
4:    s' ← perturbation(σ).
5:    σ' ← local_search(s').
6:    σ ← acceptance_criterion(σ, σ')
7: until termination criterion met.
```

What makes ILS attractive is its modularity and conceptual simplicity. The high-level mechanism of ILS is the 'perturbation-iteration' and the most important building blocks are the initial solution, the embedded LS and the perturbation phase. Lourenço et al. (2010) gave important recommendations:

(i) As in any metaheuristic, we must balance between diversification and intensification; in ILS, this balance is controlled by the perturbation and acceptance mechanisms.
(ii) We also need to find a balance between small and large perturbations. If the modification of the local optimum σ is too small, we run the risk that the perturbed solution s' remains in the basin of attraction of σ. On the contrary, if it is too large, the incumbent will be almost random and we will recover a random restart. Ideally, the perturbation should be designed in such a way that LS cannot recover the local optimum σ from the intermediate solution σ'.
(iii) The acceptance criterion must find a balance between two extreme possibilities: always accept the best solution (strong intensification) or always accept the last solution (strong diversification).
(iv) The design of a successful ILS metaheuristic is based on the following steps: (1) design each module separately; (2) bring together the modules as in Algorithm 1; (3) try to optimize each component empirically; (4) try to optimize the whole algorithm.

5. ILS for CBM

This study addresses the matrix B defined in Eq. (1). The very first and most important thing we do is computing (in $O(mn^2)$) the distance matrix W defined in Eq. (2). Later, it will be easy to evaluate δ in Eq. (3) and δ' in Eq. (6) since this takes time $O(1)$ once W is known.

5.1. Starting solution

Constructing an initial solution to start with is an important ingredient in ILS. CBM has a singular characteristic: any permutation of the binary matrix columns constitutes a solution. In particular, the identity permutation (*i.e.*, the configuration of columns obtained by taking the matrix as it is given) is a solution we can start with. However, Lourenço et al. (2010) indicated that starting from a good solution has two main advantages over a random starting solution. First, it often leads to better quality solutions; second, LS requires less computing time. Taking these observations into account, we construct a starting solution in the following way: since CBM can be solved via TSP distance matrix W defined in Eq. (2), it can also be approximated by using any TSP heuristic. In our case, we choose to start our ILS method with the Lin-Kernighan tour (or permutation) obtained by applying the `Linkern` solver to instance W. Since we know that Lin-Kernighan heuristic is one of the most efficient methods for TSP, we expect our starting solution to be of good quality.

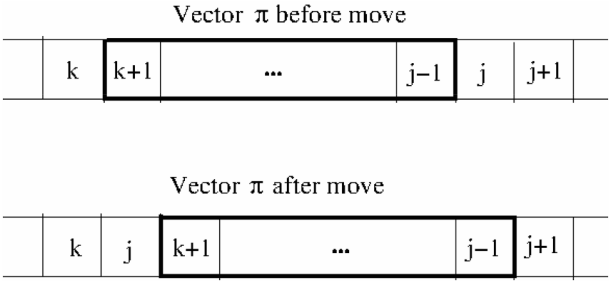

Fig. 1. Case $k < j$: Components in the bold frame will be shifted one position to the right.

5.2. Three proposed neighborhoods

Every ILS metaheuristic has an embedded LS procedure and LS requires neighborhood structures. The latter, if carefully chosen, are probably the most important ingredient of ILS. In this paper, three neighborhoods are proposed for the first time.

5.2.1. An $O(n^2)$ sized neighborhood

The structure of the first neighborhood, called `Best_Insertion`, emanates from Corollary 1. A move in this neighborhood considers a column and shifts it to a profitable place if it exists. Specifically, it consists in: (i) choosing a column b_j for some $j \in \{1,\cdots,n\}$, (ii) evaluating its contribution $\delta(j-1,j,j+1)$ in its current position, (iii) evaluating its contribution $\delta(k,j,k+1)$ if inserted between columns b_k and b_{k+1} for some $k \in \{0,\cdots,n\}, k \neq j$, (iv) moving it to position k if there is a decrease in the number of 1-blocks (*i.e.*, if Δ defined in Corollary 1 is positive). Since there are n ways for choosing column b_j and since there are $n+1$ positions (we remove its current position from the count) where to insert it, the size of `Best_Insertion` is $O(n^2)$. Furthermore, since computing Δ takes time $O(1)$, exploring `Best_Insertion` takes time $O(n^2)$. When exploring `Best_Insertion`, we have to be careful which columns we want to move. We have to distinguish between two cases, $k < j$ or $j < k$, because the intermediate columns are either shifted one position to the left or to the right (Fig. 1). In fact, in Fig. 1 we should have $k \leqslant j-2$ because otherwise we cannot correctly evaluate the contributions of columns b_k and b_j.

Algorithm 2. Exploring `Best_Insertion`

Let π be the perturbed solution and $nb1$ be the resulting number of 1-blocks
repeat
  $improved \rightarrow false$ 4: $j \rightarrow 2$ ▷ Case $k < j$ starts here
  while $j \leq n$ **do**
    $k \rightarrow 1$
    while $k \leq j-2$ **do**
      $a \rightarrow \pi_{j-1}, b \rightarrow \pi_j, c \rightarrow \pi_{j+1}, d \rightarrow \pi_{k-1}, e \rightarrow \pi_k$
      $\delta \rightarrow w_{ab} + w_{bc} - w_{ac} - w_{bd} - w_{be} + w_{de}$
      if $\delta > 0$ **then**
        $improved \rightarrow true$
        go to 26
      $k \rightarrow k+1$
    $j \rightarrow j+1$
  $j \rightarrow 1$ ▷ Case $k > j$ starts here
  while $j \leq n-1$ **do**
    $k \rightarrow j+2$
    while $k \leq n$ **do**
      $a \rightarrow \pi_{j-1}, b \rightarrow \pi_j, c \rightarrow \pi_{j+1}, d \rightarrow \pi_{k-1}, e \rightarrow \pi_k$
      $\delta \rightarrow w_{ab} + w_{bc} - w_{ac} - w_{bd} - w_{be} + w_{de}$
      if $\delta > 0$ **then**
        $improved \rightarrow true$
        go to 31
      $k \rightarrow k+1$
    $j \rightarrow j+1$
  $nb1 \rightarrow nb1 - \delta/2$ ▷ the number of 1-blocks decreases
  for $i = j, \cdots, k+1$ **do** ▷ i is decreasing from j to $k+1$
    $\pi_i \rightarrow \pi_{i-1}$ ▷ Cols. are moved one position to the right
  $\pi_k \rightarrow b$
  go to 3
  $nb1 \rightarrow nb1 - \delta/2$
  for $i = j, \cdots, k-2$ **do** ▷ Cols. are moved one position to the left
    $\pi_i \rightarrow \pi_{i+1}$
  $\pi_{k-1} \rightarrow b$
until $improved = false$

Using the first-improvement strategy, Algorithm 2 shows how to explore the `Best_Insertion` neighborhood. As the loop 'repeat' is usually executed a small number of times, a simple count shows that it takes time $O(n^2)$ to find a local optimum. The procedure is repeated while an improvement is found (*improved* = *true*), *i.e.*, a better neighbor is found. In this case the search is conducted in the vicinity of the new solution. Algorithm 2 ends when *improved* remains unchanged with value *false*, which means that a local optimum is found.

5.2.2. An $O(n^3)$ sized neighborhood

The definition of the second neighborhood, called `2_Best_Insertion`, originates from Corollaries 2 and 3. A move selects two distinct columns and shifts them together in a profitable position. It consists precisely in choosing two distinct columns, b_j for some $j \in \{1, \cdots, n\}$ and b_k for some $k \in \{1, \cdots, n\}, k \neq j$ and considering two cases:

(a) If the two columns are adjacent (without loss of generality we can assume $k = j+1$), we compute $u = \delta'(j-1, j, j+1, j+2)$, then we compute $v = \delta'(r-1, j, j+1, r)$, and if $u - v > 0$ then we perform the move by shifting columns b_j and b_{j+1} between columns b_{r-1} and b_r. The move allows to decrease the number of 1-blocks by $(u-v)/2$.

(b) If the two columns are non-adjacent (we can assume that $j < k$), we compute $u = \delta(j-1, j, j+1)$ and $v = \delta(k-1, k, k+1)$, then we compute $w = \delta'(r-1, j, k, r)$, and if $w - u - v > 0$ then we perform the move by shifting columns b_j and b_k in this order between columns b_{r-1} and b_r. The move allows to decrease the number of 1-blocks by $(w-u-v)/2$.

As mentioned above, regarding `Best_Insertion`, we had two cases to consider. In exploring `2_Best_Insertion`, however, we need to deal with five distinct cases. As there are $n(n-1)/2$ ways to choose two distinct columns and as there are $n+1$ possible positions to place them, exploring `2_Best_Insertion` takes time $O(n^3)$. Unfortunately, this time complexity seems to be deterrent, at least for large values of n. Indeed, using `2_Best_Insertion` neighborhood and running a single LS iteration on instance SCPA1 takes more than 7000 s of CPU time. Needless to say, this neighborhood is not chosen.

5.2.3. A large-scale neighborhood

Consider matrix B which has eleven 1-blocks, as an example. It is important to note that we do not need an artificial null column in the last position because we will not move the columns but the submatrices (we will explain why we need the first null column). First, we start by computing the Hamming distance matrix defined in Eq. (2) as follows:

$$B = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \\ \begin{matrix} 3 & 3 & 2 & 4 & 1 & 2 & 3 & 2 \end{matrix} \end{array} \qquad W = \begin{pmatrix} 0 & 3 & 2 & 2 & 2 & 3 & 3 & 2 & 2 \\ & 0 & 3 & 1 & 3 & 2 & 2 & 1 & 1 \\ & & 0 & 2 & 2 & 3 & 1 & 4 & 2 \\ & & & 0 & 4 & 3 & 1 & 2 & 2 \\ & & & & 0 & 1 & 3 & 2 & 2 \\ & & & & & 0 & 2 & 1 & 3 \\ & & & & & & 0 & 3 & 3 \\ & & & & & & & 0 & 2 \\ & & & & & & & & 0 \end{pmatrix} \tag{7}$$

The row below matrix B gives the distance $w_{j,j+1}$ between adjacent columns. For example, the distance between column number 3 and column number 4 is $w_{34} = 4$. Hence, placing these two columns side by side is a bad strategy. However, a better one is to separate them by moving the two following submatrices:

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

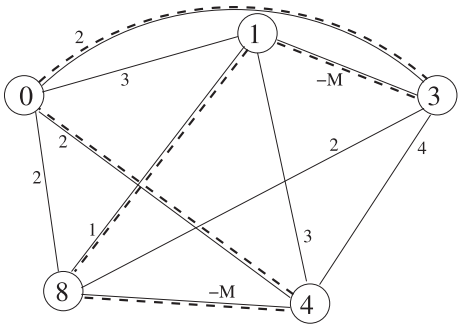

Fig. 2. TSP model for the separation of columns 3 and 4.

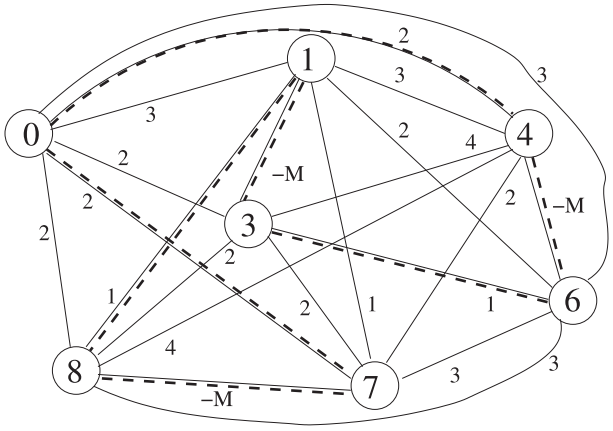

Fig. 3. Illustrating example with $p = 3$.

In addition, the question we can ask is: in how many ways can we separate columns 3 and 4 by moving the two submatrices? There are $2! = 2$ ways to permute the two submatrices. Furthermore, every submatrix can be either maintained in its current configuration or flipped.

Hence, the total number of possible moves is $2^2 \times 2! - 1 = 7$. In the following, we give the exhaustive list of possible configurations which constitutes a neighborhood of the current configuration.

1 2 3 8 7 6 5 4
3 2 1 4 5 6 7 8
3 2 1 8 7 6 5 4
4 5 6 7 8 1 2 3
4 5 6 7 8 3 2 1
8 7 6 5 4 1 2 3
8 7 6 5 4 3 2 1

Now, the aim is to obtain the best of the seven moves. The problem can be modeled as a TSP. Since the internal columns of the submatrices (column 2 in the first submatrix and columns 5, 6, 7 in the second) are irrelevant, they can be temporarily omitted. Consider a complete graph on five nodes labeled (by their column labels for convenience) 0,1,3,4,8. We need the artificial column number 0 for reasoning in terms of tours instead of Hamiltonian paths. The distance between nodes i and j is w_{ij}, but to enforce nodes 1 and 3 (respectively, nodes 4 and 8) to be adjacent in every optimal tour, we set $w_{13} = -M$ (respectively, $w_{48} = -M$) where M is a positive sufficiently large number (Fig. 2).

By Solving TSP, we obtain the optimal tour 03184 in dotted lines. By removing the artificial node 0 and recovering the internal columns, we obtain the locally optimal move (third in the exhaustive list above) giving nine 1-blocks.

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

This is the case with 2 submatrices. In fact, the procedure can be generalized to any number p of submatrices. The neighborhood construction depends on the choice of the p pairs of columns to separate. However, there is an obvious technical requirement: each submatrix must have at least two columns. Hence, we choose the p largest values among the distances $w_{12}, w_{23}, \cdots, w_{n-1,n}$ without creating a submatrix with a single column. The rationale behind this choice is that these pairs are those which contribute the most to the number of 1-blocks. The neighborhood thus constructed is called `Submatrices_Moving`. Since there are p submatrices, there are $p!$ ways of permuting them, and each submatrix can be flipped, the neighborhood size is $2^p \times p! - 1$ (the current configuration is dropped from the count). The corresponding TSP instance has $2p+1$ nodes (node 0 and the two external columns of each of the p submatrices). The local optimal solution in `Submatrices_Moving` can be directly computed by solving the TSP.

Let us illustrate the case with $p = 3$ by reconsidering matrix B in Example (7). We cannot, for example, separate columns 1 and 2, as this would create a submatrix with only one column. Because of the largest distances $w_{34} = 4$ and $w_{67} = 3$, we choose to separate column 3 from column 4 and column 6 from column 7. Fig. 3 illustrates the TSP model.

An optimal tour (dotted lines) is found giving a local optimal configuration with only eight 1-blocks, as follows:

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

Algorithm 3 gives a pseudo-code for exploring the large-scale neighborhood `Submatrices_Moving`. It is clear that the computational burden is in Line 4 which consists in solving a TSP on $2p+1$ cities.

Algorithm 3. Exploring `Submatrices_Moving`

1: Let π be the local optimal solution obtained by Algorithm 2 and let $nb1$ be the resulting number of 1-blocks
2: Among the $n-1$ pairs of adjacent columns π_j, π_{j+1} $(j = 1, \cdots, n-1)$ select p pairs having the largest value $w_{\pi_j \pi_{j+1}}$ without creating a submatrix with a single column
3: Define TSP instance on $2p+1$ cities as explained above.
4: Solve TSP to find the local optimal configuration
5: Update the best solution and the corresponding number $nb1$ of 1-blocks

5.3. LS

LS explores two neighborhoods: `Best_Insertion` and `Submatrices_Moving`. While the neighborhood `Best_Insertion` is systematically scanned, finding a local optimal solution in the large-scale neighborhood `Submatrices_Moving` by solving TSP on $2p+1$ cities may take a prohibitive amount of time when p is large enough. Exchanging columns is costly and we have to avoid it. At the very beginning, we start with the vector π corresponding to the tour obtained by the `Linkern` solver from which we remove node number 0. Any subsequent changes in the column configuration order are registered in this vector.

5.4. Perturbation, acceptance and termination criteria

Perturbation is what caused the ILS to escape from the local optima by introducing a bias in the random walk. It is therefore the third most important ingredient. The perturbation mechanism is required to construct a new solution to restart the search with. A rather complex procedure is described in the work of Lourenço et al., 2010 and introduced by Baxter, 1981 to solve a location problem. It has again proved its efficiency in the TSP context (Codenotti et al., 1996). It consists in modifying the costs and running an LS heuristic on this modified instance so that the output will be the perturbed solution.

In our case, we construct a new distance matrix W' simply by altering some of the coefficients of the distance matrix W, then we apply a simple 2-opt heuristic with the perturbed distance matrix W', either to the current local optimum or to the best solution found so far. The aim is to obtain an intermediate solution which will be improved by LS with the original distance matrix W. Two parameters govern the perturbation procedure (Algorithm 4): V, the value with which the coefficients of matrix W are altered; and ϕ $(0 < \phi < 0.5)$, the frequency of the coefficients affected (about $(200 \times \phi)$% of the coefficients). The choice of these two parameters depends on the matrix density.

Algorithm 4. Perturbation

1: Let π be either the best or the current local optimum
2: **for** $i = 0, \cdots, n$ **do**
3: **for** $j = i+1, \cdots, n+1$ **do**
4: $e \leftarrow w_{ij}$
5: generate a random number x uniformly distributed in $]0,1[$
6: **if** $x \leqslant \phi$ and $e \geqslant V$ **then**
7: $e \leftarrow e - V$
8: **else**
9: **if** $x > 1 - \phi$ **then**
10: $e \leftarrow e + V$
11: $W'_{ij} \leftarrow e$
12: $W'_{ji} \leftarrow e$
13: Use 2-opt to improve π for TSP with distance matrix W'

As the 2-opt heuristic takes a small number of iterations, the time

bound of Algorithm 4 is dominated by the $O(n^2)$ operations for computing the matrix W'. There is a rationale behind the choice of the 2-opt improvement heuristic in Line 13. Define distance $d(\pi, \pi')$ between two solutions or permutations π and π' as the number of distinct components. For example, if π' is obtained from π by swapping two columns then the distance is 2. Now suppose that columns a_k and a_j are concerned by a 2-opt move ($k < j$). Before the move, permutation π is:

$$\pi = (a_1, \cdots, a_{k-1}, a_k, a_{k+1}, \cdots, a_{j-1}, a_j, a_{j+1}, \cdots, a_n)$$

but after the move it becomes:

$$\pi' = (a_1, \cdots, a_{k-1}, a_j, a_{j-1}, \cdots, a_{k+1}, a_k, a_{j+1}, \cdots, a_n)$$

with distance $d(\pi,\pi') = j - k + 1$. If column a_k is far away from column a_j then the distance between the two solutions is large. In this case, a single move of 2-opt consists of a considerable perturbation which can hardly be undone by LS.

In the implementation of the proposed ILS method, we find a balance between intensification and diversification (*i.e.* the perturbation mechanism is applied half of the time to the last configuration of columns, and half of the time to the best). Finally, termination is controlled by an a priori fixed number *NBITER* of iterations.

5.5. Proposed algorithm

Algorithm 5 provides the pseudo-code of the proposed method. The computation in Line 1 of the symmetric matrix W takes time $O(m \times n^2)$. Let us consider a single iteration of the loop "repeat". Line 8 clearly runs in time $O(n^2)$, as already discussed, and Line 11 costs $O(n^2)$ since it mainly consists in scanning matrix W. The computation of the number of 1-blocks in Line 12 costs $O(m \times n)$ since it needs scanning binary matrix A. Hence, if there is a heavy burden, it must happen in Line 9. In fact, in the proposed implementation, instead of searching a local optimum in `Submatrices_Moving` by exactly solving the induced TSP, we settle for a local near optimal solution using the `Linkern` solver. Therefore, the complexity of the proposed ILS metaheuristic is dominated by solving at each iteration a TSP instance on $2p+1$ cities by using the `Linkern` solver.

Algorithm 5. `ILS for CBM`

Input: $m \times n$-binary matrix A and parameters $NBITER, \phi, V, p$
Output: Best permutation π^* and the number b^* of 1-blocks in $A^{(\pi^*)}$
1: Compute symmetric matrix W defined in (2)
2: Let π^* be the configuration obtained from Lin-Kernighan tour
3: Let b^* be the number of 1-blocks in $A^{(\pi^*)}$
4: $\pi \leftarrow \pi^*$
5: $nb1 \leftarrow b^*$
6: $i \leftarrow 0$
7: **repeat**
8: Use Algorithm 2 to improve π
9: Use Algorithm 3 to further improve the actual local optimal solution
10: Update π^*, b^* whenever an improvement is found
11: Use Algorithm 4 to obtain the intermediate perturbed permutation π
12: Compute the true number $nb1$ of 1-blocks in $A^{(\pi)}$ by using matrix W
13: $i \leftarrow i + 1$
14: **until** $i = NBITER$

6. Experiments

This section aims to demonstrate the applicability of our methodology through empirical testing. In this respect, two factors must be studied: accuracy and computation speed.

6.1. Experimental design

Our method was coded in C on Linux OS (Opensuse 13.2) and ran on Dell Optiplex 390 (Intel Core i3-2120, 3.3 GHz, 2 Mb RAM). It was tested on a dataset that contains thirty large instances from the OR-Library. These data are actually set covering instances and we used them by omitting the costs. There are six types and each type comprises five instances (Table 2). All computing times will be reported in seconds.

Table 2
Characteristics of the instances

Instance type	SCPA	SCPB	SCPC	SCPD	SCPG	SCPH
Density (%)	2	5	2	5	2	5
$m \times n$	300×3000		400×4000		1000×10000	

6.2. Solving CBM instances by two TSP solvers

As explained at the beginning of Section 3, CBM can be polynomially transformed to TSP with symmetric distance matrix W defined in Eq. (2). Hence, it can be solved or approximated using methods designed for TSP. Two of these methods are freely available on the website managed by Applegate et al. (2006): an exact method called `Concorde` and an approximate one called `Linkern`. The second is an implementation of the chained Lin-Kernighan heuristic. We used the two executable codes by calling them inside a system command and reading their results from an output file.

While using `Concorde`, a time limit of 18000 (respectively, 50000) seconds was imposed for the execution of type A, B, C and D (respectively, G and H) instances. Since `Linkern` has a certain level of randomization, it was run five times. Table 4 lists the results of the two solvers. The first column gives the instance name, while the second column provides the number of 1-blocks in the given matrix configuration. The results of `Concorde` are shown in the next two columns; the first giving the optimal solution value (or upper bound), and the second, the computing time, where the term 'Lim' or 'LIM' indicates that the imposed time limit is reached and the solver is stopped before it completes the computation. Six of the instances consumed all the allocated time limit without finishing. In this case, only upper bounds are reported. Thus, it can be said that solving CBM exactly is not a good option due to the high CPU times. The results of `Linkern` are given in the three following columns, where the first gives the best solution found over five runs, the average solution value and the average computing time. Unlike `Concorde`, computing times of `Linkern` are relatively smaller. Due to its time complexity $O(m \times n^2)$, computing the distance matrix W in (2) can take a long time. Indeed, it takes on average more than 500 s for instances of type G or H, which represents a significant amount of time.

6.3. Parameter setting

The proposed method requires the customization of a set of four parameters: the fixed number *NBITER* of iterations, the number p of submatrices chosen for defining the large-scale neighborhood `Submatrices_Moving`, and the numbers ϕ and V that govern the perturbation mechanism. A generic decision must be made for *NBITER* since its value does not impact heavily the ILS results. It is obvious that more iterations give better results, but computing times increase. The three other parameters need problem-specific decisions. Indeed, the two parameters ϕ and V directly affect the perturbation strength and hence the whole method. They prove to be dependent on the matrix density. Finally, the number p of submatrices for defining `Submatrices_Moving` is crucial. The larger the value of p, the greater the improvement of

Table 3
Parameter setting.

Density	Instance type	V	ϕ
2%	SCPA, SCPC, SCPG	1	0.2
5%	SCPB, SCPD, SCPH	2	0.2

Table 4
Results of the competing methods (Lim = 18.000 and LIM = 50.000 seconds)).

		Solving CBM by using Concorde		Approximating CBM by using Linkern (5 runs)			Results of the method of Soares et al., 2020 (5 runs)				Results of the proposed method (5 runs)		
Instance	Ini. config.	Opt. sol.	Comp. time	Best sol.	Aver. sol.	Aver. time	Ini. sol.	Best sol.	Aver. sol.	Aver. time	Best sol.	Aver. sol.	Aver. time
SCPA1	17710	11459	7066.71	11633	11643.6	114.98	14991	13515	13537.2	1500.0	11542	11545.2	336.40
SCPA2	17664	11425	4509.70	11600	11604.2	121.59	14968	13470	13522.8		11513	11521.4	316.44
SCPA3	17697	11467	8689.03	11631	11638.0	113.31	15014	13504	13522.0		11544	11554.6	327.60
SCPA4	17718	11464	Lim	11640	11650.2	112.19	14992	13494	13547.8		11549	11555.0	334.41
SCPA5	17683	11428	451.83	11611	11620.8	111.09	14973	13507	13525.2		11515	11522.4	328.66
SCPB1	42591	31462	Lim	31887	31909.4	193.90	39860	35406	35476.8	1500.0	31705	31714.6	393.24
SCPB2	42623	31438	579.48	31843	31854.4	179.54	39850	35371	35411.8		31675	31685.2	388.82
SCPB3	42546	31449	120.74	31859	31873.0	196.99	39890	35433	35458.0		31674	31684.2	405.34
SCPB4	42598	31439	589.34	31826	31847.8	221.22	39877	35403	35442.8		31679	31687.6	435.49
SCPB5	42524	31423	604.85	31825	31836.4	191.64	39883	35402	35433.0		31647	31657.0	395.32
SCPC1	31346	21953	2028.20	22280	22295.8	386.98	27702	26381	26397.6	2000.0	22089	22100.2	852.79
SCPC2	31256	21810	2836.20	22213	22218.4	376.14	27627	26297	26316.8		22000	22009.2	796.14
SCPC3	31227	21852	846.83	22244	22255.4	362.92	27659	26339	26350.6		22022	22032.4	780.46
SCPC4	31265	21875	1856.68	22251	22265.2	370.60	27642	26327	26357.4		22025	22034.2	845.82
SCPC5	31260	21847	Lim	22206	22217.8	369.62	27598	26318	26334.6		21984	22016.4	832.69
SCPD1	76019	58795	5625.22	59563	59597.8	396.24	72428	67607	67637.2	2000.0	59181	59214.4	823.93
SCPD2	75805	58801	949.12	59529	59552.4	355.10	72395	67528	67583.8		59188	59212.4	784.80
SCPD3	75971	58863	3054.71	59637	59660.2	448.08	72448	67602	67623.2		59291	59305.6	838.17
SCPD4	75855	58701	1608.59	59402	59423.2	383.53	72295	67492	67513.2		59064	59092.0	771.86
SCPD5	75891	58741	4081.85	59443	59474.6	384.95	72410	67558	67563.8		59133	59146.6	799.55
SCPG1	195364	159095	29184.37	161795	161846.4	5658.61					160642	160703.6	10804.73
SCPG2	195279	159078	34809.06	161810	161892.8	5573.38					160577	160619.8	10273.65
SCPG3	195379	159098	34987.83	161965	162009.2	7541.37					160650	160759.0	12468.61
SCPG4	195355	159069	LIM	161878	161990.4	7467.45					160682	160809.2	11824.62
SCPG5	195316	159059	37602.36	161934	162010.6	7277.65					160633	160702.6	11890.72
SCPH1	473738	405757	LIM	409969	410095.0	4358.02					408120	408222.0	8874.18
SCPH2	473775	405761	LIM	409892	410085.8	4311.26					408160	408255.8	8786.19
SCPH3	473682	405724	15574.66	410012	410082.4	4397.32					408158	408260.8	8580.02
SCPH4	473861	405744	5028.62	409927	410045.8	4358.13					408071	408207.0	8468.13
SCPH5	473758	405775	19016.41	410028	410152.4	4346.86					408136	408322.2	8298.88

Table 5
Comparison of the competing methods (average values are given for each instance type).

	Concorde	Linkern		Soares et al., 2020		Proposed method	
Instance type	Computing time	Aver. dev. from opt.	Computing time	Aver. dev. from opt.	Computing time	Aver. dev. from opt.	Computing time
SCPA	> 7743.45	1.60%	114.63	18.19%	1500.00	0.80%	328.70
SCPB	> 3978.88	1.35%	196.66	12.73%	1500.00	0.77%	403.64
SCPC	> 5113.58	1.75%	373.25	20.51%	2000.00	0.78%	821.58
SCPD	3063.90	1.30%	393.58	14.98%	2000.00	0.70%	803.66
SCPG	> 37602.36	1.80%	6703.69			1.03%	11452.47
SCPH	> 27923.94	1.07%	4346.86			0.62%	8601.48
Global aver.	> 14238.19	1.48%	2021.45	16.60%	1750.00	0.78%	3735.26

the current best solution, but also more computing time needed to scan the neighborhood.

For setting the parameters ϕ and V, the following methodology is adopted. Two groups of instances are randomly selected, one for each density value. Each group contains three instances, one from each type. The parameter V can take values $1, 2$ because with $V = 3$ the initial solution is almost never improved. The possible values for parameter ϕ are $0.05, 0.1, 0.2, 0.3, 0.4$. For example, if $\phi = 0.1$ then 20% of the coefficients of matrix W' are altered. Each instance is run 10 times, each time with a different combination. Table 3 summarizes the values of the parameters giving the best results.

It can be observed that when $p = n/5$ and $n = 10000$, the TSP instance that must be solved to obtain a local optimal solution in Submatrices_Moving includes $2p+1 = 4001$ cities. Solving such an instance is rather time-consuming even using Linkern. So, in the implementation, we take $p = n/5$ during $NBITER1 = 6$ iterations, then $p = n/10$ during $NBITER2 = 6$ iterations, and then $p = n/20$ during $NBITER3 = 12$ iterations. Thus, the total number of iterations of the ILS method is $NBITER = 24$.

6.4. Results and comparison

Applying our method to the dataset, we obtained the results detailed in the last three columns of Table 4. Since the code is run five times on each instance, the first of the columns gives the best solution, the second gives the average solution value, and the third gives the average computing time. The solutions provided by Linkern and used in our method as initial configurations are those that are reported in Columns 5 to 7. The total computation time of the proposed ILS method (last column of Table 4) is the sum of the Linkern computing time and the time

spent by ILS. The code of the method proposed by Soares et al. (2020) is available on the website indicated in Carvalho (2020). We run the code five times on each instance of type A, B, C and D, but not on types G and H. Columns 8 to 11 of Table 4 provide the results obtained, the first column giving the value of the initial solution.

To compare the four competing methods, the mean values for each instance type are calculated and listed in Table 5 where the last row provides the overall mean values. The comparison indicates the following:

1. `Concorde`, as expected, solves CBM exactly but with excessive CPU times. However, if the number n of columns of the binary matrices increases, then computation times will be prohibitive.
2. `Linkern` produces good solutions faster.
3. The method proposed by Soares et al. behaves badly. This fact has a rational explanation. The time complexity of this method is $O(m \times n^2)$. Whereas we use Eqs. (3) and (6) for the evaluation which need time $O(1)$, Soares et al., 2020 used the formula presented in Haddadi et al. (2015) which needs time $O(m)$. But in the last reference, LS is applied once and we had to find a compromise between computing matrix W in $O(m \times n^2)$ or applying LS with the same time complexity. What differs in ILS is that LS is iterated many times. So, the previous compromise disappears. In ILS, we must compute matrix W in $O(m \times n^2)$ only once and leave LS run in $O(n^2)$. LS with time complexity $O(m \times n^2)$ is costly. In these circumstances, given a fixed amount of time, it cannot perform a large number of iterations. This is the reason why, despite the large amount of computing time allowed, Soares et al. (2020) reported only an average of 8 iterations on the small instances tested.
4. The method proposed in this study spends about half of the time finding an initial solution using `Linkern` and the other half on improving it.
5. This method appears to be a good alternative to the three competing algorithms. It is about twice as accurate as `Linkern` and more than 20 times more accurate than the method presented by Soares et al. (2020). On the other hand, the proposed ILS metaheuristic is about four times faster than `Concorde` and about three times faster than the method presented by Soares et al. (2020).

Based on the previous observations, we can provide these recommendations: 1) If an optimal solution is mandatory, there is no other solution than to use `Concorde` or any other exact method. 2) If time is more important than the solution quality, `Linkern` is the best choice. 3) If we want a better solution than the one provided by `Linkern`, our method is recommended although the computing time is doubled.

6.5. What is the effect of the matrix density on problem-solving?

In general, regardless of the method used, the density is a sign that the problem is more difficult to solve and requires more computing time. This property has been observed in practice for most combinatorial optimization problems with binary constraint matrices. But the question that arises is: what is the effect of the matrix density on the efficiency of the proposed method? From a computing time point of view, since `Linkern` is the main ingredient in our method and seems insensitive to the matrix density, the ILS method is practically unaffected, and we can verify this fact based on the results in Table 5. Similarly, the solution quality does not seem to be affected since the average deviation from the optimum is almost the same regardless of the matrix density.

7. Conclusion

This study focused on the CBM, an important combinatorial optimization problem that models permutation problems arising in many contexts. We started by proving that it is NP-hard on $(3, 2)$-matrices, narrowing the gap between easy and hard instances. Next, we proposed an ILS metaheuristic to solve the CBM. The neighborhoods explored are proposed for the first time in this paper. Encouraging computational results were obtained making our method competitive with three other methods.

Based on these findings, three directions for future research can be highlighted. First, since the case where binary matrices have at most two 1s per row and at most two 1s per column is trivially solved, only a single case remains open: $(2, 3)$-CBM where binary matrices have a maximum of two 1s per column and a maximum of three 1s per row. Next, an in-depth study of the neighborhood k`_Best_Insertion`, $k \geqslant 2$, is worth conducting. Although its exhaustive exploration takes a prohibitive $O(n^{k+1})$ time bound, it is interesting to find an intelligent way to explore it, for $k = 2, 3$ or 4, by restricting the search to candidate sets and by using the don'tlook-bits technique. Furthermore, the search for a local optimal solution in the large-scale neighborhood `Submatrices_Moving` is time-consuming even using `Linkern`. A more sophisticated implementation is needed to make the whole method faster and more accurate.

Acknowledgment

The author would like to acknowledge D. Applegate (AT&T Labs-Research), R. Bixby (ILOG and Rice University), V. Chvátal (Concordia University), and W. Cook (University of Waterloo) for their kind permission to use their `Concorde` and `Linkern` TSP solvers. We would also like to thank M.A.M. Carvalho for providing the link (which can be found below) to the computer code of the ILS metaheuristic proposed by Soares et al. (2020).

References

Applegate, D., Bixby, R., Chvatal, V., Cook, W., 2006. Concorde TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html, last accessed June 2020.

Baxter, J., 1981. Local optima avoidance in depot location. J. Oper. Res. Soc. 32 (9), 815–819.

Beasley, J.E., 1990. Or-library. http://people.brunel.ac.uk/ mastjjb/jeb/orlib/scpinfo.html, last accessed June 2020.

Becceneri, J.C., Yanasse, H.H., Soma, N.Y., 2004. A method for solving the minimization of open stacks. Comput. Oper. Res. 31 (14), 2315–2332.

Booth, K.S., Lueker, G.S., 1976. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci 13 (3), 335–379.

Carvalho, M.A.M., 2020. Source code. https://github.com/MarcoCarvalhoUFOP/ILS-CBM, last accessed June 2020, available under GPL-3.0 license.

Carvalho, M.A.M., Soma, N.Y., 2015. A breadth-first search applied to the minimization of the open stacks. J. Oper. Res. Soc. 66 (6), 936–946.

Codenotti, B., Manzini, G., margara, L., Resta, G., 1996. Perturbation: an efficient technique for the solution of very large instances of the euclidean tsp. INFORMS J. Comput. 8 (2), 125–133.

Dyson, R., Gregory, A., 1974. The cutting stock problem in the flat glass industry. J. Oper. Res. Soc. 25 (1), 41–53.

Fulkerson, D., Gross, O., 1965. Incidence matrices and interval graphs. Pac. J. Math. 15 (3), 835–855.

Garey, M., Johnson, D., Stockmeyer, L., 1974. Some simplified NP-complete problems. In: 6th ACM Symposium on Theory of Computing (STOC'74), pp. 47–63.

Haddadi, S., 2002. A note on the NP-hardness of the consecutive block minimization problem. Int. Trans. Oper. Res. 9 (6), 775–777.

Haddadi, S., 2015. Benders decomposition for set covering problems almost satisfying the consecutive ones property. J. Comb. Optim. 33 (1), 60–80.

Haddadi, S., Chenche, S., Guessoum, F., Cheraitia, M., 2015. Polynomial-time local-improvement algorithm for consecutive block minimization. Inf. Process. Lett. 115, 612–617.

Haddadi, S., Layouni, Z., 2008. Consecutive block minimization is 1.5-approximable. Inf. Process. Lett. 108 (3), 132–135.

Johnson, D., Krishnan, S., Chhugani, J., Kumar, S., Venkatasubramanian, S., 2004. Compressing large boolean matrices using reordering techniques. In: Proceedings of the 30th International Conference on Very Large Data Bases. VLDB '04. VLDB Endowment, pp. 13–23.

Kou, L.T., 1977. Polynomial complete consecutive information retrieval problems. SIAM J. Comput. 6 (1), 67–75.

Lemire, D., Kaser, O., 2011. Reordering columns for small indexes. Inform. Sci. 181, 2550–2570.

Lima, J.R., Carvalho, M.A.M., 2017. Descent search approaches applied to the minimization of open stacks. Comput. Ind. Eng. 112, 175–186.

Linhares, A., Yanasse, H.H., 2002. Connections between cutting-pattern sequencing, VLSI design, and flexible machines. Comput. Oper. Res. 29 (12), 1759–1772.
Lourenço, H.R., Martin, O.C., Stützle, T., 2010. Iterated local search: Framework and applications. In: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer, US, Boston, MA, pp. 363–397.
Madsen, O.B., 1979. Glass cutting in a small firm. Math. Prog. 17 (1), 85–90.
Madsen, O.B., 1988. An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry. J. Oper. Res. Soc. 39 (3), 249–256.
McConnell, R.M., 2004. A certifying algorithm for the consecutive-ones property. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms. SODA'04. Philadelphia, USA, pp. 768–777.
Paiva, G.S., Carvalho, M.A.M., 2017. Improved heuristic algorithms for the job sequencing and tool switching problems. Comput. Oper. Res. 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. Comput. Oper. Res. 120, 1–11, published online doi: 10.1016/j.cor.2020.104948.
Stützle, T., Ruis, R., 2018. Iterated local search. In: Martí, R.e. (Ed.), Handbook of Heuristics. Springer Nature, pp. 579–605.