A Polynomial-Time Algorithm for Finding a Minimal Conflicting Set Containing a Given Row

Guillaume Blin, Romeo Rizzi, Stéphane Vialette

A Polynomial-Time Algorithm for Finding a Minimal Conflicting Set Containing a Given Row

Guillaume Blin[1], Romeo Rizzi[2], and Stéphane Vialette[1]

[1] Université Paris-Est, LIGM - UMR CNRS 8049, France.
{gblin,vialette}@univ-mlv.fr
[2] DIMI, Università di Udine, Italy. rrizzi@dimi.uniud.it

Abstract. A binary matrix has the *Consecutive Ones Property* (C1P) if there exists a permutation of its columns (i.e. a sequence of column swappings) such that in the resulting matrix the 1s are consecutive in every row. A *Minimal Conflicting Set* (MCS) of rows is a set of rows $\mathcal{R}$ that does not have the C1P, but such that any proper subset of $\mathcal{R}$ has the C1P. In [5], Chauve *et al.* gave a $O(\Delta^2 m^{\max(4,\Delta+1)}(n+m+e))$ time algorithm to decide if a row of a $m \times n$ binary matrix with at most Δ 1s per row belongs to at least one MCS of rows. Answering a question raised in [2], [5] and [25], we present the first polynomial-time algorithm to decide if a row of a $m \times n$ binary matrix belongs to at least one MCS of rows.

1 Introduction

A binary matrix has the *Consecutive Ones Property* (C1P) if its columns can be ordered in such a way that all 1s on each rows are consecutive. Both deciding if a given binary matrix has the C1P and finding the corresponding columns permutation can be done in linear-time [4, 11, 12, 15–17, 19, 22]. A characterization of matrices having the C1P is given in [23]. The C1P of matrices has a long history and it plays an important role in combinatorial optimization, including application fields such as scheduling [1, 13, 14, 28], information retrieval [18], and railway optimization [20, 21, 24] (see [8] for a recent survey).

This paper is devoted to *Minimal Conflicting Sets* (MCS), *i.e.*, minimal sets of rows or columns that prevent the matrix from having the C1P. A *Minimal Conflicting Sets of Rows* (MCSR) (resp. *Minimal Conflicting Sets of Columns* (MCSC)) is a set of rows $\mathcal{R}$ (resp. columns $\mathcal{C}$) of a matrix that does not have the C1P but such that any proper subset of $\mathcal{R}$ (resp. $\mathcal{C}$) has the C1P. Dom [9] has given an algorithm to find a minimum conflicting set in a given matrix. Recent research in comparative genomics has proved MCS to be of particular interest. Indeed, Bergeron *et al.* [2] and Stoye *et al.* [25] have shown how to compute parsimonious evolution scenarios of gene clusters by ranking rows according to their *Conflicting Index* (CI), *i.e.*, the number of MCSR involving a row. In both papers, the problems of efficiently computing the CI of a row and of generating

all the MCS of a matrix problems were explicitly raised. Chauve *et al.* [5] gave the first results for those two problems by presenting a $O(\Delta^2 m^{\max(4,\Delta+1)}(n+m+e))$ time algorithm to decide if a row of $m \times n$ binary matrix with at most Δ 1s per row has a positive CI. Note that this algorithm is practical only for small Δ and Chauve *et al.* left as an open problem the question of whether there exists a polynomial-time algorithm to decide if a row has a positive CI. In this paper we give a positive answer to this open problem by combining characterization of matrices having the C1P with graph pruning techniques.

This paper is organized as follows. In Section 2, we recall basic definitions and formally introduce the problem we are interested in. We give in Section 3 a polynomial-time algorithm to decide if a row has a positive CI, and propose in Section 4 some natural extensions. Due to space constraint, most proofs are omitted.

2 Preliminaries

We assume readers have basic knowledge about graph theory [7] and we shall thus use most conventional terms of graph theory without defining them (we only recall basic definitions). Let $G = (V, E)$ be a graph. The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u : \{u, v\} \in E\}$. Two distinct vertices $u, v \in V$ are called *twins* if they have the same neighborhood, *i.e.*, $N(u) = N(v)$. For any $V' \subseteq V$, we denote by $G[V']$ the *subgraph of G induced by V'* with the additional property that all isolated vertices have been deleted (whereas the latter requirement is non-standard it will prove useful to simplify the presentation). A *path* from vertex u to vertex v is abbreviated to a uv-path. Finally, for any path p in G, we let $V(p) \subseteq V$ stand for the set of all vertices involved in p.

A matrix M is *simple* if it does not contain two identical columns or rows, and *simplifying* a matrix is the (polynomial-time) process of deleting identical rows and columns. In the sequel, we assume any matrix to be simplified. A $(0,1)$-matrix is a matrix in which each entry is either zero or one. Let M be a $m \times n$ $(0,1)$-matrix. Its corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$ is defined as follows: for every row (resp. column) of M there is a black (resp. white) vertex in $\mathcal{R}$ (resp. $\mathcal{C}$), and there is an edge between a black vertex v_i and a white vertex v_j, *i.e.*, an edge between the vertices that correspond to the i^{th} row and the $j^{\text{th}th}$ column of M, if and only if $M[i,j] = 1$. Equivalently, M is the reduced adjacency matrix of $G(M)$. We shall usually write $\mathcal{R} = \{r_i : 1 \leq i \leq m\}$ and $\mathcal{C} = \{c_j : 1 \leq j \leq n\}$. In the sequel, we shall speak indistinctly about binary matrices and their corresponding vertex-colored bipartite graphs. An *asteroidal triple* is an independent set of three vertices such that each pair is joined by a path that avoids the neighborhood of the third. Tucker [27] has proved that if a $(0,1)$-matrix contains an asteroidal triple then it does not have the C1P. Furthermore, Tucker has given a complete characterization of matrices containing asteroidal triples.

Theorem 1 ([27], Theorem 9). *A $(0,1)$-matrix has the C1P if and only if it contains none of the matrices M_{I_k}, M_{II_k}, M_{III_k} ($k \geq 1$), M_{IV} and M_V depicted in Figure 1.*

Write $\mathcal{T} = \{M_{I_k}, M_{II_k}, M_{III_k}, M_{IV}, M_V\}$. Let M be a $(0,1)$-matrix. According to Theorem 1, for any MCSR $\mathcal{R}_T$ of M, $G(M)[\mathcal{R}_T \cup \mathcal{C}]$ contains at least one Tucker configuration $T = (\mathcal{R}_T, \mathcal{C}_T, E') \in \mathcal{T}$, and for any $\mathcal{R}'_T \subsetneq \mathcal{R}_T$, $G(M)[\mathcal{R}'_T \cup \mathcal{C}]$ has the C1P, *i.e.*, it does not contain a Tucker configuration. A similar observation can be done for MCSC. For the sake of brevity, any Tucker configuration contained in an MCSR (or MCSC) will be said to be *responsible* for this MCSR (or MCSC).

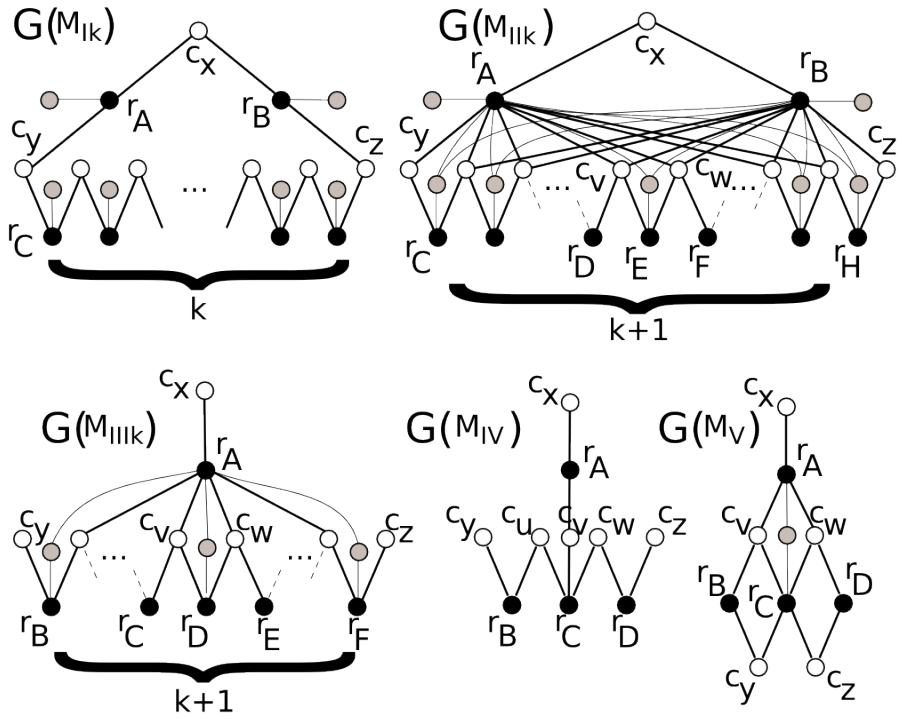

Fig. 1. Forbidden bipartite graphs [27]. Black (resp. white) vertices correspond to rows (resp. columns) of the corresponding matrices. Gray vertices and light edges are not part of the Tucker configurations but represent the extra columns that our algorithm will report. For $G(M_{I_k})$, any triple of white vertices forms an asteroidal triple. For all other forbidden structures, there are exactly one asteroidal triple (c_x, c_y, c_z).

Following our previous work on Tucker forbidden structures [3], our algorithm is based on shortest paths and two graph pruning techniques (graph pruning techniques were introduced by Conforti *et al.* [6]). Let us define the `clean`

and `anticlean` pruning operations. Let M be a binary matrix and $G(M) = (\mathcal{R}, \mathcal{C}, E)$ be the corresponding vertex-colored bipartite graph. For any vertex $v \in \mathcal{R}$ (resp. $v \in \mathcal{C}$), `clean`(v) results in the graph $G(M)[\mathcal{R} \cup (\mathcal{C} \setminus N(v))]$ (resp. $G(M)[(\mathcal{R} \setminus N(v)) \cup \mathcal{C}]$). In other words, `clean`$(v)$ results in a graph where any neighbor of v has been deleted. For any node $v \in \mathcal{R}$ (resp. $v \in \mathcal{C}$), `anticlean`(v) results in the graph $G(M)[\mathcal{R} \cup (\mathcal{C} \setminus \{u : u \notin N(v)\})]$ (resp. $G(M)[(\mathcal{R} \setminus \{u : u \notin N(v)\}) \cup \mathcal{C}]$). In other words, `anticlean`$(v)$ results in a graph where any node that does not belong to the same partition nor the neighborhood of v has been deleted. By abuse of notation, we shall write `clean`$(u_1, u_2, \ldots, u_k)$ for the sequence `clean`(u_1), `clean`(u_2), $\ldots$, `clean`(u_k) (a similar abuse will be used for `anticlean`).

Remark 1. It is of particular importance to note that we shall always consider that vertices given as inputs to our algorithms will never be affected (*i.e.*, deleted) by the `clean` and `anticlean` operations.

3 Finding an MCSR involving a given row

We present in this section a polynomial-time algorithm for reporting (if it exists) an MCSR involving a given row. Our main result can be stated as follows.

Proposition 1. *Let M be $m \times n$ $(0,1)$-matrix. For any row r of M, deciding whether there exists an MCSR involving row r is solvable in $O(m^6 n^5 (m+n)^2 \log(m+n))$ time.*

To prove Proposition 1, we provide a sequence of polynomial-time algorithms for finding a minimal Tucker configuration of a given type $T \in \mathcal{T} = \{M_{I_k}, M_{III_k}, M_{II_k}, M_{IV}, M_V\}$ (in this particular order) responsible for an MCSR involving a given row (if it exists). The following easy lemma will prove to be extremely useful in the sequel.

Lemma 1. *Let $T = (\mathcal{R}_T, \mathcal{C}_T, E_T)$ be a Tucker configuration responsible for an MCSR involving a given row r in $G(M) = (\mathcal{R}, \mathcal{C}, E)$. Then $\mathcal{R}_T$ is an MCSR involving r and there is no smaller Tucker configuration – in terms of number of rows (or black nodes) – in $G(M)[\mathcal{R}_T \cup \mathcal{C}]$.*

3.1 M_{I_k} Tucker configurations

Proposition 2. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{I_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^4 n^4)$ time procedure.*

Observe that M_{I_k} is a hole (a chordless cycle of length at least 6), and hence without loss of generality we associate r to r_A in $G(M_{I_k})$ (see Figure 1). We

Algorithm 1 Check-$M_{I_k}(c_x, c_y, r_A, r_B, r_C)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, three black vertices $r_A, r_B, r_C \in \mathcal{R}$ (r identified to r_A) and two white vertices $c_x, c_y \in \mathcal{C}$ such that $(r_C, c_y, r_A, c_x, r_B)$ is a path in $G(M)$. It is assumed that $G(M)$ does not contain any $G(M_{I_1})$ or $G(M_{I_2})$ involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{I_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is an MCSR involving row r, or return the failure message "NO" if such a configuration does not exist.

if $N(r_A) \cap N(r_B) \cap N(r_C) \neq \emptyset$ **then**
  return "NO"
end if
`clean`(c) for all $c \in N(r_A) \setminus N(r_B)$
`clean`(c) for all $c \in N(r_A) \setminus N(r_C)$
`clean`(r_A, c_x, c_y)
delete vertex r_A
if there exists a $r_B r_C$-path in the pruned graph **then**
  let P be a shortest $r_B r_C$-path in the pruned graph
  return return $\{r_A\} \cup \{r_i : r_i \in V(P) \cap \mathcal{R}\}$
else
  return "NO"
end if

need to consider three cases: $k = 1$, $k = 2$ and $k > 2$. We first try to find some $G(M_{I_1}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ involving row r using any brute-force algorithm. If we succeed, we are done since any proper subset of $\mathcal{R}_T$ – of size at most 2 – cannot contain any other Tucker configuration. Otherwise, using any brute-force algorithm, we try to find some $G(M_{I_2}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ involving row r with the additional property that there do not exist $\mathcal{R}'_T \subsetneq \mathcal{R}_T$ and $\mathcal{C}'_T \subseteq \mathcal{C}$ such that $G(M_{III_1}) = G(M)[\mathcal{R}'_T \cup \mathcal{C}'_T]$. This latter additional constraint is necessary and sufficient since $G(M_{III_1})$ is the only smaller Tucker configuration involving row r that could occur in $G(M)$. If both tries failed, we turn to $k > 2$ and apply Algorithm 1 for every tuple of parameters (c_x, c_y, r, r_B, r_C), where $c_x, c_y \in \mathcal{C}$, $r_B, r_C \in \mathcal{R}$, and (r_C, c_y, r, c_x, r_B) is a path in $G(M)$. Among the non-failure answers (if any), we return the smallest one.

Lemma 2. *If there exist an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ with $\{r_A = r, r_B, r_C\} \subseteq \mathcal{R}_T$ such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{I_k})$ for some $k > 2$ and some $\mathcal{C}_T \subseteq \mathcal{C}$ with $\{c_x, c_y\} \subseteq \mathcal{C}_T$, then Algorithm 1 for parameters $(c_x, c_y, r_A = r, r_B, r_C)$ finds it.*

We now turn to evaluating the time complexity of one call to Algorithm 1. Checking that $N(r_i) \cap N(r_B) \cap N(r_C)$ is empty is a $O(n)$ time procedure. Cleaning any white vertex can be done in $O(m)$ time and cleaning r_A can be done in $O(n)$ time. Using a BFS search, finding a shortest $r_B r_C$-path is $O(n + m + mn)$ time. Summing up, the total time complexity of Algorithm 1 is $O(mn)$.

Correctness of Proposition 2 follows from Lemma 2. What is left is to prove the total time complexity. According to Lemma 2, for any row r, we can call Algorithm 1 for parameters $(c_x, c_y, r_A = r, r_B, r_C)$ to find an MCSR (if it exists)

involving row r. There are $O(m^2n^2)$ such tuples, and hence we have a $O(m^3n^3)$ time procedure for $k > 2$. As for $k = 1$ and $k = 2$, a brute-force algorithm yields a $O(m^4n^4)$ time procedure, the dominant term in our approach for $G(M_{I_k})$ Tucker configurations.

3.2 M_{III_k} Tucker configurations

We assume in this subsection that there does not exist a $G(M_{I_k})$ Tucker configuration in $G(M)$ responsible for an MCSR involving a given row r.

Proposition 3. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Assuming that there does not exist a $G(M_{I_k})$ in $G(M)$ responsible for an MCSR involving row r, finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{III_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^5n^5(m+n)^2\log(m+n))$ time procedure.*

If such a $G(M_{III_k})$ Tucker configuration exists and is responsible for an MCSR involving row r, then r can be any of the black vertices of $G(M_{III_k})$. However, thanks to symmetries, it is enough to suppose that row r is identified to r_A, r_D or r_F in $G(M_{III_k})$.

Our algorithm is as follows. If we don't succeed in finding some $G(M_{I_k})$ Tucker configuration responsible for an MCSR involving row r (see Subsection 3.1), we look for some $T = G(M_{III_1}) = (\mathcal{R}_T, \mathcal{C}_T, E_T)$ Tucker configuration involving row r (brute-force algorithm). If such a $G(M_{III_1})$ Tucker configuration exists, $\mathcal{R}_T$ is certainly an MCSR (involving row r). If we fail, we call Algorithm 2 for every tuple of arguments $(c_x, c_y, c_z, r, r_B, r_F)$ with $r_B, r_F \in \mathcal{R}$ and $c_x, c_y, c_z \in \mathcal{C}$, and next call Algorithm 3 for every tuple of arguments $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r, r_E, r_F)$ with $r_A, r_B, r_C, r_E, r_F \in \mathcal{R}$ and $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$. Among the non-failure solutions, we return the smallest one.

Lemma 3. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (identified to r_A or r_B) such that $\{r_A, r_B, r_F\} \subseteq \mathcal{R}_T$ and $\{c_x, c_y, c_z\} \in \mathcal{C}_T$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{III_k})$ for some $k > 1$ and some $\mathcal{C}_T \subseteq \mathcal{C}$, then Algorithm 2 for arguments $(c_x, c_y, c_z, r_A, r_B, r_F)$ finds it.*

Lemma 4. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (identified to r_D) such that $\{r_A, r_B, r_C, r_D, r_E, r_F\} \subseteq \mathcal{R}_T$ and $\{c_v, c_w, c_x, c_y, c_z\} \in \mathcal{C}_T$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{III_k})$ for some $k > 1$ and some $\mathcal{C}_T \subseteq \mathcal{C}$, then Algorithm 3 for arguments $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r, r_E, r_F)$ finds it.*

We now turn to evaluating the time complexity of Algorithm 3 (the time complexity of Algorithm 2 is clearly negligible with that of Algorithm 3). There are $O(m^5n^5)$ calls to Algorithm 3, and hence the whole procedure (summing up all calls to Algorithm 3) is $O(m^5n^5(m+n)^2\log(m+n))$ time. As for the exhaustive search for $G(M_{III_1})$ Tucker configurations, it is $O(m^3n^4)$ time. Therefore, the algorithm, as a whole, is $O(m^5n^5(m+n)^2\log(m+n))$ time. Proposition 3 is proved.

Algorithm 2 Check-$M_{III_k}(c_x, c_y, c_z, r_A, r_B, r_F)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, three black vertices $r_A, r_B, r_F \in \mathcal{R}$ and three white vertices $c_x, c_y, c_z \in \mathcal{C}$ such that $r_A \subseteq N(c_x)$, $r_B \subseteq N(c_y)$, and $r_F \subseteq N(c_z)$. Row r is identified to r_A or r_B. It is assumed that $G(M)$ does not contain any $G(M_{I_k})$ or $G(M_{III_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that where $G(M_{III_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is a (row-) minimal MCSR involving row r if it exists, or the failure message "NO" if such a configuration does not exist.

`clean`(c_x, c_y, c_z)
`clean`(c) for all $c \notin N(r_A)$
`anticlean`(r_A)
remove vertex r_A
if there exists a $r_B r_F$-path in the pruned graph **then**
  let P be a shortest $r_B r_F$-path in the pruned graph
  return return $\{r_A\} \cup \{r : r \in V(P) \cap \mathcal{R}\}$
else
  return "NO"
end if

Algorithm 3 Check-$M_{III_k}(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r_E, r_F)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, six black vertices $r_A, r_B, r_C, r_D, r_E, r_F \in \mathcal{R}$ and five white vertices $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$ such that $r_A \subseteq N(c_x) \cap N(c_v) \cap N(c_w)$, $r_B \subseteq N(c_y)$, $r_F \subseteq N(c_z)$, $r_C \subseteq N(c_v)$, $r_D \subseteq N(c_v) \cap N(c_w)$, and $r_E \subseteq N(c_w)$. Row r is identified to r_D. It is assumed that $G(M)$ does not contain any $G(M_{I_k})$ or $G(M_{III_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{III_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is a (row-) minimal MCSR involving row r, or the failure message "NO" if such a configuration does not exist.

if $N(r_C) \cap N(r_D) \cap N(r_E) \neq \emptyset$ **or** $(N(r_C) \cup N(r_D) \cup N(r_E)) \setminus N(r_A) \neq \emptyset$ **then**
  return "NO"
end if
`clean`(c) for all $c \in N(r_D)$
`clean`$(c_v, c_w, c_x, c_y, c_z)$
`clean`(c) for all $c \notin N(r_A)$
`anticlean`(r_A)
remove the node r_A
if there exists a $r_B r_F$-path using r_D in the pruned graph **then**
  let P be a shortest such $r_B r_F$-path in the pruned graph
  return return $\{r_A\} \cup \{r | r \in V(P) \cap \mathcal{R}\}$
else
  return "NO"
end if

3.3 M_{II_k} Tucker configurations

We assume in this subsection that there does not exist a $G(M_{I_k})$ nor a $G(M_{III_k})$ Tucker configuration in $G(M)$ responsible for an MCSR involving a given row r.

Proposition 4. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Assuming that there does not exist a $G(M_{I_k})$ in $G(M)$ responsible for an MCSR involving row r, finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^6 n^5 (m+n)^2 \log(m+n))$ time procedure.*

Notice that if such a $G(M_{II_k})$ Tucker configuration does exist and is responsible for an MCSR involving row r then r can be any of the black vertices of $G(M_{II_k})$ (see Figure 1). However, thanks to symmetries, it is enough to suppose that row r is identified to r_A, r_C or r_E in $G(M_{II_k})$ (all other possibilities are equivalent up to a straightforward renaming). Although at first odd, it is also crucial for correctness to assume that no $G(M_{I_k})$ is responsible in $G(M)$ for an MCSR involving row r.

Our algorithm is as follows. If we don't succeed in finding some $G(M_{I_k})$ Tucker configuration responsible for an MCSR involving row r (see Subsection 3.1), we next look for some $G(M_{II_1}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ Tucker configuration involving row r using any brute-force algorithm. If we succeed, we are done since any proper subset of $\mathcal{R}_T$ – of size at most 3 – cannot contain any other Tucker configuration. Otherwise, we use a three-step procedure. We first call Algorithm 4 for every tuple $(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$ with $r_A = r, r_B, r_C, r_H \in \mathcal{R}$ and $c_x, c_y, c_z \in \mathcal{C}$, and next for every tuple $(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$ with $r_A, r_B, r_C = r, r_H \in \mathcal{R}$ and $c_x, c_y, c_z \in \mathcal{C}$. Finally, we call Algorithm 5 for every tuple $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_H, r_D, r, r_F)$, $r_A, r_B, r_C, r_D, r_F, r_H \in \mathcal{R}$ and $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$. Among the non-failure solutions, we return the smallest one.

Lemma 5. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (either identified to r_A or r_C) such that $\{r_A, r_B, r_C, r_H\} \subseteq \mathcal{R}_T$, $\{c_x, c_y, c_z\} \subseteq \mathcal{C}_T$ for some $\mathcal{C}_T \subseteq \mathcal{C}$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $k > 1$, then Algorithm 4 for arguments $(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$ finds it.*

Lemma 6. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (identified to r_E) such that $\{r_A, r_B, r_C, r_D, r_E, r_F, r_H\} \subseteq \mathcal{R}_T$, $\{c_v, c_w, c_x, c_y, c_z\} \subseteq \mathcal{C}_T$ for some $\mathcal{C}_T \subseteq \mathcal{C}$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $k > 1$, then Algorithm 5 for arguments $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r, r_F, r_H)$ finds it.*

We now turn to evaluating the time complexity of Algorithm 5 (the time complexity of Algorithm 4 is clearly negligible with that of Algorithm 5). We first observe that, in a graph of order n, one can find a shortest uv-path that goes through a given node w in $O(n^2 \log n)$ time [26]. Indeed, it is enough to add a new vertex x, $N(x) = \{u, v\}$, and use the algorithm of Suurballe to find two vertex-disjoint paths between a source (*i.e.*, w) and a sink (*i.e.*, x) with minimum sum length. Testing emptiness of $N(r_H) \cap N(r_A) \setminus N(r_B)$, $N(r_C) \cap N(r_B) \setminus N(r_A)$, $N(r_D) \cap N(r_E) \cap N(r_F)$, and $(N(r_D) \cup N(r_E) \cup N(r_F)) \setminus (N(r_A) \cap N(r_B)))$ is a simple $O(n)$ time procedure. Cleaning any white node can be done in $O(m)$ time, and cleaning r_A and r_B in $O(n)$ time. Moreover, according to the above,

Algorithm 4 Check-$M_{II_k}(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, four black vertices $r_A, r_B, r_C, r_H \in \mathcal{R}$ and three white vertices $c_x, c_y, c_z \in \mathcal{C}$ such that $(r_C, c_y, r_A, c_x, r_B, c_z, r_H)$ is a path in $G(M)$. Row r is identified either to r_A or r_C. Furthermore, it is assumed that $G(M)$ does not contain any $G(M_{I_k})$ or $G(M_{II_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{II_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is an MCSR involving row r, or the failure message "NO" if such a configuration does not exist.

if $N(r_H) \cap (N(r_A) \setminus N(r_B)) \neq \emptyset$ or $N(r_C) \cap (N(r_B) \setminus N(r_A)) \neq \emptyset$ **then**
  return "NO"
end if
`clean`(c) for all $c \notin N(A) \cap N(B)$
`clean`(c_x, c_y, c_z)
`anticlean`(r_A, r_B)
remove the vertices r_A and r_B
if there exists a $r_C r_H$-path in the pruned graph **then**
  let P be a shortest $r_C r_H$-path in the pruned graph
  return $\{r_A, r_B, r_C, r_H\} \cup \{r : r \in V(P) \cap \mathcal{R}\}$
else
  return "NO"
end if

finding a shortest $r_C r_H$-path that goes through r_E in the pruned graph (after having removed r_A and r_B) is a in $O((m+n)^2 \log(m+n))$ procedure. Therefore, the time complexity of one call to Algorithm 5 is $O((m+n)^2 \log(m+n))$ time.

According to Lemma 6, for a given row r, we have to call Algorithm 5 for any tuple $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r, r_F, r_H)$, $r_A, r_B, r_C, r_D, r, r_F, r_H \in \mathcal{R}$ and $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$, and return the smallest MCSR involving row r (if such a Tucker configuration exists). There are $O(m^6 n^5)$ such tuples for a given row r, and hence trying all tuples results in a $O(m^6 < n^5 (m+n)^2 \log(m+n))$ time procedure. The exhaustive search for $G(M_{II_1})$ is a simple $O(m^4 n^4)$ time procedure. Therefore, one can find the smallest $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ that is responsible for an MCSR involving row r in $O(m^6 n^5 (m+n)^2 \log(m+n))$ time (if it exists). Proposition 4 is proved.

3.4 M_{IV} and M_V Tucker configurations

Proposition 5. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{IV})$ (resp. $G(M_V)$) for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^3 n^6)$ (resp. $O(m^3 n^5)$) time procedure.*

Proof. The proof is by brute-force searching for a $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{IV})$ (resp. $G(M_V)$)) Tucker configuration involving row r (identified to r_A, see Fig. 1). The running time for both cases follows easily.

Algorithm 5 Check-$M_{II_k}(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r_E, r_F, r_H)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, seven black vertices $r_A, r_B, r_C, r_D, r_E, r_F, r_H \in \mathcal{R}$ and five white vertices $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$ such that both $(r_C, c_y, r_A, c_x, r_B, c_z, r_H)$ and $(r_D, c_v, r_E, c_w, r_F)$ are paths in $G(M)$ and $\{c_v, c_w\} \subseteq N(r_A) \cap N(r_B)$. Row r is identified to r_E. It is assumed that $G(M)$ contains neither a $G(M_{I_k})$ or a $G(M_{II_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{II_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is row-minimal MCSR involving row r if it exists, or the failure message "NO" is such a configuration does not exist.

1: **if** $N(r_H) \cap N(r_A) \setminus N(r_B) \neq \emptyset$ **or** $N(r_C) \cap N(r_B) \setminus N(r_A) \neq \emptyset$ **or** $N(r_D) \cap N(r_E) \cap N(r_F) \neq \emptyset$ **or** $(N(r_D) \cup N(r_E) \cup N(r_F)) \setminus (N(r_A) \cap N(r_B)$ **then**
2: **return** "NO"
3: **end if**
4: `clean`(c) for all $c \in N(r_E)$
5: `clean`(c) for all $c \notin N(A) \cap N(B)$
6: `clean`$(c_x, c_y, c_z, c_v, c_w)$
7: `anticlean`(r_A, r_B)
8: remove the black vertices r_A and r_B
9: **if** there exists a $r_C r_H$-path that goes though r_E in the pruned graph **then**
10: let P be a shortest $r_C r_H$-path that goes though r_E in the pruned graph
11: **return** return $\{r_A, r_B, r_C, r_D, r_E, r_F, r_H\} \cup \{r : r \in V(P) \cap \mathcal{R}\}$
12: **else**
13: **return** "NO"
14: **end if**

What is left is to prove that $G(M)[\mathcal{R}_T, \mathcal{C}]$ does not contain any smaller Tucker configuration. We first prove correctness for $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{IV})$. Indeed, focus on $G(M)[\mathcal{R}_T, \mathcal{C}]$ and suppose that there exists some white vertex $c_s \in \mathcal{C} \setminus \mathcal{C}_T$ that is not a clone of some white vertex in $\mathcal{C}_T$. Then it follows that $N(c_s) = \{r_A, r_B\}$, $N(c_s) = \{r_A, r_D\}$, $N(c_s) = \{r_B, r_D\}$, or $N(c_s) = \{r_c\}$. If $N(c_s) = \{r_c\}$ we are done. Otherwise, $G(M)[\mathcal{R}_T, \mathcal{C}]$ contains a (smaller) $G(M_{I_1})$ Tucker configuration. A contradiction since $G(M)$ is assumed not to contain a M_{I_k} Tucker configuration involving row r. We now turn to prove correctness for $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_V)$. Focus on $G(M)[\mathcal{R}_T, \mathcal{C}]$ and suppose that there exists some white vertex $c_s \in \mathcal{C} \setminus \mathcal{C}_T$ that is not a clone of some white vertex in $\mathcal{C}_T$. Then it follows that $N(c_s) = \{r_A, r_B\}$, $N(c_s) = \{r_A, r_C\}$, $N(c_s) = \{r_A, r_D\}$, or $N(c_s) = \{r_B, r_D\}$. If $N(c_s) = \{r_A, r_C\}$ we are done. Otherwise, $G(M)[\mathcal{R}_T, \mathcal{C}]$ contains a (smaller) $G(M_{I_1})$ Tucker configuration. A contradiction since $G(M)$ is assumed not to contain a M_{I_k} Tucker configuration involving row r. □

3.5 Summing up

Table 1 summarizes our results.

Tucker configuration	Running time
M_{I_k}	$O(m^3n^4)$
M_{II_k}	$O(m^6n^5(m+n)^2\log(m+n))$
M_{III_k}	$O(m^5n^5(m+n)^2\log(m+n))$
M_{IV}	$O(m^2n^6)$
M_V	$O(m^3n^5)$
Total	$O(m^6n^5(m+n)^2\log(m+n))$

Table 1.

4 Applying our framework to related problems

Our graph pruning techniques can be used for solving related combinatorial problems. We briefly discuss these related points.

First, the property we have considered was C1P, where a matrix has C1P when the columns can be sorted in such a way that on each row the 1s are consecutive. It is simple to check that our framework can also consider the case when the property is the transpose, *i.e.*, the rows can be sorted in such a way that on each column the 1s are consecutive.

More interestingly, let us point out that our framework also implies an polynomial-time algorithm for the *Circular Ones Property* (Circ1P) studied in [10]. A matrix has the Circ1P if its columns can be ordered in such a way that all 1s or all 0s (possibly both) on each row are consecutive (it may help to consider the matrix as being wrapped around a vertical cylinder). Indeed, according to [10], Corollary 2.2, given an $m \times n$ matrix M and an arbitrary integer $1 \leq j \leq n$, one can compute a matrix M' such that M has the Circ1P if and only if M' as the C1P. Therefore, we can check in polynomial-time if a given row is involved in an MCSR for both C1P and Circ1P.

References

1. J.J. Bartholdi, J.B. Orlin, and H.D. Ratliff. Cyclic scheduling via integer programs with circular ones. *Oper Res*, 28(5):1074–1085, 1980.
2. A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve. Reconstructing ancestral gene orders using conserved intervals. In Inge Jonassen and Junhyong Kim, editors, *4th International Workshop on Algorithms in Bioinformatics (WABI'04)*, volume 3240 of *Lecture Notes in Computer Science*, pages 14–25, Bergen, Norway, September 2004. Springer.
3. G. Blin, R. Rizzi, and S. Vialette. A faster algorithm for finding minimum Tucker submatrices. In *6th Computability in Europe (CiE'10)*, Lecture Notes in Computer Science, 2010. To appear.
4. K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335379, 1976.
5. C. Chauve, U.-U Haus, T. Stephen, and V.P. You. Minimal conflicting sets for the consecutive ones property in ancestral genome reconstruction. In Francesca

Ciccarelli and István Miklós, editors, *RECOMB-CG 09*, volume 5817 of *LNCS*, pages 48–58. Springer, 2009.
6. M. Conforti and M.R. Rao. Structural properties and decomposition of linear balanced matrices. *Mathematical Programming*, 55:129–168, 1992.
7. R. Diestel. *Graph Theory*. Number 173 in Graduate texts in Mathematics. Springer-Verlag, second edition, 2000.
8. M. Dom. Algorithmic aspects of the consecutive-ones property. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, 98:2759, 2009.
9. M. Dom. *Recognition, Generation, and Application of Binary Matrices with the Consecutive-Ones Property*. Dissertation. Cuvillier, 2009. Institut für Informatik, Friedrich-Schiller-Universität Jena.
10. M. Dom, J. Guo, and R. Niedermeier. Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. *Journal of Computer and System Sciences*, In Press, Corrected Proof, 2009.
11. D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15(3):835855, 1965.
12. M. Habib, R.M. McConnell, C. Paul, and L. Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoret. Comput. Sci.*, 234(12):5984, 2000.
13. R. Hassin and N. Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29 – 42, 1991.
14. D.S. Hochbaum and A. Levin. Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. *Discrete Optimization*, 3(4):327 – 340, 2006.
15. W.-L. Hsu. A simple test for the consecutive ones property. *J. Algorithms*, 43(1):116, 2002.
16. W.-L. Hsu and R.M. McConnell. Pc trees and circular-ones arrangements. *Theoret. Comput. Sci.*, 296(1):99116, 2003.
17. N. Korte and R.H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):6881, 1989.
18. L.T. Kou. Polynomial complete consecutive information retrieval problems. *SIAM J. Comput.*, 6(1):67–75, 1977.
19. R.M. McConnell. A certifying algorithm for the consecutive-ones property. In ACM Press, editor, *15th Annual ACMSIAM Symposium on Discrete Algorithms SODA '04*, page 768777, 2004.
20. S. Mecke, A. Schöbel, and D. Wagner. Station location complexity and approximation. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways ATMOS '05*, Dagstuhl, Germany, 2005.
21. S. Mecke and D. Wagner. Solving geometric covering problems by data reduction. In Springer, editor, *12th Annual European Symposium on Algorithms ESA '04*, volume 3221 of *Lecture Notes in Comput. Sci.*, page 760771, 2004.
22. J. Meidanis, O. Porto, and G.P. Telles. On the consecutive ones property. *Discrete Appl. Math.*, 88:325354, 1998.
23. N.S. Narayanaswamy and R. Subashini. A new characterization of matrices with the consecutive ones property. *Discrete Applied Mathematics*, 157(18):3721–3727, 2009.
24. N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. *Discrete Optimization*, 1(2):215 – 228, 2004.
25. J. Stoye and R. Wittler. A unified approach for reconstructing ancient gene clusters. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 6(3):387–400, 2009.

26. J.W. Suurballe. Disjoint paths in networks. *Networks*, 4:125–145, 1974.
27. A.C. Tucker. A structure theorem for the consecutive 1s property. *Journal of Combinatorial Theory. Series B*, 12:153162, 1972.
28. A.F. Veinott and H.M. Wagner. Optimal capacity scheduling. *Oper Res*, 10:518–547, 1962.