Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

Algorithmic Aspects of the Consecutive-Ones Property

Michael Dom*

Abstract

We survey the consecutive-ones property of binary matrices. Herein, a binary matrix has the *consecutive-ones property (C1P)* if there is a permutation of its columns that places the 1s consecutively in every row. We provide an overview over connections to graph theory, characterizations, recognition algorithms, and applications such as integer linear programming and solving Set Cover.

1 Introduction

We start with considering a short example for the occurrence of the C1P in practical applications. The example has its background in computational biology, where the construction of physical maps for the human DNA was a central issue in the past years [5, 6, 45, 72, 97]. A *physical map* is a map that describes the relative order of *markers* on a *chromosome*. A chromosome is basically a long sequence of DNA, and a marker is a short DNA sequence that appears only once on the chromosome and, therefore, is of special interest. To create a physical map, the chromosome is cut into shorter pieces, which are duplicated and called *clones*. Thereafter, one tests for each of the clones which of the markers appears on it. These tests find out whether a marker appears on a clone, but it is not possible to determine the order of the markers on the clone. The result is a binary matrix where every row corresponds to a clone and every column corresponds to a marker. If a marker appears on a clone, then the corresponding entry of the matrix is 1, otherwise it is 0. The crucial observation for finding the correct order of the markers is that if two markers A and B appear on a clone x, but another marker C does not appear on x, then C cannot lie between A and B on the chromosome. Therefore, to figure out the order of the markers on the chromosome, all one has

*Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany, michael.dom@uni-jena.de.

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

to do is to order the columns of the matrix in such a way that in every row the 1s appear consecutively. In concrete practical applications, however, the biochemical methods always produce errors such that it is often impossible to order the columns in the resulting matrices as described. One way to deal with these errors is to discard a smallest possible number of clones such that the remaining clones lead to a consistent order of the markers. On the level of binary matrices, this approach means that one has to delete a minimum number of rows such that the resulting matrix has the C1P.

The survey is structured as follows. In the remainder of this section, we provide the definitions used throughout the paper and some basic observations and results on the C1P. Section 2 deals with the relation between the C1P and graph classes. In Section 3, we survey recognition algorithms for the C1P. Section 4 describes some cases of problems that are NP-hard in general, but become polynomial-time solvable on instances with the C1P.

1.1 Preliminaries

An $m \times n$ *matrix* contains $m \cdot n$ entries, which are arranged in m rows and n columns. The entry in the ith row and jth column of a matrix M is denoted by $m_{i,j}$; moreover, we usually use r_i and c_j to denote the ith row and the jth column, respectively, of a matrix. One can also regard a matrix as a set of columns together with an order on this set; the order of the columns is called the *column ordering* of the matrix. Two matrices M and M' are called *isomorphic* if M' is a permutation of the rows and columns of M. A matrix M' is a *submatrix* of a matrix M if we can select a subset of the rows and columns of M in such a way that deleting all but the selected rows and columns results in a matrix that is isomorphic to M'. If one can find a submatrix M' of M in this way, we say that M *contains* M' as a submatrix and that M' is *induced* by the selected rows and columns. A matrix M is *M'-free* if M' is not a submatrix of M.

A matrix whose entries are all from $\{0, 1\}$ is called a *binary matrix* or 0/1-*matrix*; a matrix whose entries are all from $\{0, 1, -1\}$ is called a $0/\pm1$-matrix. *Complementing* a column or row of a matrix means that all 1-entries in this column or row, respectively, are replaced by 0s and all 0-entries are replaced by 1s.

Every 0/1-matrix M can be interpreted as a bipartite graph, which is called the *representing graph* G_M of M: For every row and every column of a matrix M, there is a vertex in its representing graph G_M, and for every 1-entry $m_{i,j}$ in M, there is an edge in G_M connecting the vertices corresponding to the ith row and the jth column of M.

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [30, 37, 79]. One dimension is the input size n and the other one a *parameter* d. A problem is called *fixed-parameter*

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

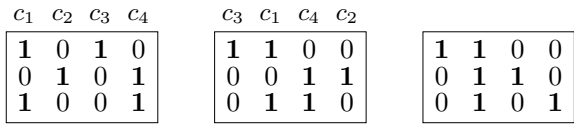


Figure 1: Example for the C1P: The matrix on the left has the C1P because by permuting its columns (labeled with c_1–c_4) one can obtain the matrix shown in the middle where the 1s in each row appear consecutively. The matrix on the right, in contrast, does not have the C1P [91].

tractable if it can be solved in $f(d) \cdot n^{O(1)}$ time, where f is a function only depending on d. The basic concept for parameterized intractability is W[1]-*hardness*; if a problem is W[i]-hard for any $i \geq 1$, it is presumably not fixed-parameter tractable.

1.2 The Consecutive-Ones Property

The consecutive-ones property of binary matrices appears in many practical applications, such as scheduling [8, 53, 54, 68, 93], information retrieval [67], railway optimization [75, 76, 86], and computational biology [3, 5, 6, 18, 45, 72, 97]. Moreover, the C1P has close connections to graph theory (see Section 2) and plays an important role in the area of solving (integer) linear programs [54, 55, 80, 81, 93] (see also Section 4.1). The formal definition of the C1P and some related concepts reads as follows.

Definition 1.1. A *block of 1s* (*block of 0s*) in a row of a binary matrix M is a maximal set of consecutive 1-entries (0-entries) in this row. A binary matrix has the *strong consecutive-ones property* (*strong C1P*) if in every row the 1s appear consecutively, that is, if every row contains at most one block of 1s. A binary matrix has the *consecutive-ones property* (*C1P*) if its columns can be permuted in such a way that the resulting matrix has the strong C1P. If an ordering for the columns of a binary matrix yields the strong C1P, it is called a *C1-ordering*.

See Figure 1 for examples of matrices with and without the C1P. The terms introduced in Definition 1.1 can be defined analogously for 1-entries appearing consecutively in the columns of a matrix instead of the rows: If the *rows* of a matrix M can be permuted in such a way that in every *column* the 1s appear consecutively, then M has the *C1P for columns*.

A property that is very similar to the C1P but less restrictive is called the *circular-ones property*: Here one imagines the matrix as wrapped around a vertical cylinder and demands that, possibly after some column permutations, in every row the 1s appear consecutively on the cylinder (which implies that the 0s also appear consecutively):

A:

c_1	c_2	c_3	c_4
1	0	**1**	0
0	**1**	**1**	**1**
1	0	0	**1**

B:

c_1	c_3	c_2	c_4
1	**1**	0	0
0	**1**	**1**	**1**
1	0	0	**1**

C:

c_1	c_2	c_3	c_4
1	**1**	0	0
0	**1**	**1**	0
0	**1**	0	**1**

D:

c_1	c_2	c_3	c_4
1	**1**	0	0
0	**1**	**1**	0
1	0	**1**	0

Figure 2: Example for the Circ1P: The matrix A has the Circ1P because by permuting its columns (labeled with c_1–c_4) one can obtain the matrix B where in each row the 1s or the 0s appear consecutively. The matrices C and D in contrast, do not have the Circ1P [90, 91].

Definition 1.2. A binary matrix has the *strong circular-ones property* (*strong Circ1P*) if in every row the 1s appear consecutively or the 0s appear consecutively (or both). A binary matrix has the *circular-ones property* (*Circ1P*) if its columns can be permuted in such a way that the resulting matrix has the strong Circ1P. If an ordering for the columns of a binary matrix yields the strong Circ1P, then it is called a *Circ1-ordering*.

See Figure 2 for an example. When imagining a matrix M as wrapped around a vertical cylinder, it makes no sense to declare one of its columns as the "leftmost" or "rightmost" column. In this setting, therefore, the term *circular column ordering* is used to describe the order of M's columns: The circular column ordering defines for every column c of M a predecessor and a successor, but it does not declare any of M's columns as the "leftmost" or "rightmost" column. If M has the strong Circ1P, then its circular column ordering is called a *Circ1-circular ordering*.

There exist several characterizations for matrices having the C1P (see Sections 2 and 3). Together with the following observation due to Tucker [90], these characterizations can also be used to recognize matrices with the Circ1P.

Corollary 1.1 (following from [90, Theorem 1])**.** *Let M be an $m \times n$ matrix and let j be an arbitrary integer with $1 \leq j \leq n$. Form the matrix M' from M by complementing all rows with a 1 in the jth column of M. Then, M has the Circ1P if and only if M' has the C1P.*

An example for Corollary 1.1 can be seen in Figure 2: Complementing in one of the matrices C and D all rows with a 1 in column c_4 yields the matrix D, which does not have the C1P (this can easily be seen by considering the columns c_1–c_3); hence, the matrices C and D do not have the Circ1P. Corollary 1.1 implies the following conclusion.

Corollary 1.2. *Let M be a 0/1-matrix and M' be the matrix obtained by inserting a column that contains only 0s to M. Then the following statements are equivalent. 1. M' has the Circ1P. 2. M' has the C1P. 3. M has the C1P.*

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

The relation between different C1-orderings and Circ1-orderings for the columns of a matrix is summarized by Hsu and McConnell [63]:

Theorem 1.1 ([63, Theorems 3.4 and 3.8])**.** *1. Let M be a matrix having the C1P. Then* every *C1-ordering for M's columns can be obtained by starting from an arbitrary C1-ordering and applying a sequence of reverse operations, each of them reversing a linear module in the respective column ordering.*

2. Let M be a matrix having the Circ1P. Then every *Circ1-circular ordering for M's columns can be obtained by starting from an arbitrary Circ1-circular ordering and applying a sequence of reverse operations, each of them reversing a circular module in the respective circular ordering.*

Thereby, a *linear module* of a matrix M is a set C of columns such that in every row of M the entries belonging to C have the same value or the entries *not* belonging to C are all 0 (or both). A *circular module* is a set C of columns such that in every row of M the entries belonging to C have the same value or the entries *not* belonging to C have the same value (or both). A *reverse operation* takes a set of consecutive columns from the column ordering or the circular column ordering, respectively, of a matrix and puts it into reverse order. Clearly, applying a reverse operation to a linear module (a circular module) of a matrix that has the strong C1P (the strong Circ1P) does not destroy this property. Theorem 1.1, however, strengthens this observation.

One consequence of statement 2 in Theorem 1.1 is the following finding, which states a relation between the Circ1-orderings and the *shifted C1-orderings* of the columns of a matrix having the C1P. Thereby, the column ordering of a matrix M is a shifted C1-ordering if the strong C1P can be obtained by repeatedly taking the column that is currently placed at the leftmost position and moving it from there to the rightmost position.

Lemma 1.1 ([28])**.** *Let M be an $m \times n$ matrix that has the C1P and contains at most $(n+1)/2$ 1s per row. Then every Circ1-ordering for M's columns is also a shifted C1-ordering.*

2 Graph Classes and the C1P/Circ1P

Matrices can be represented by graphs and vice versa, and, therefore, the C1P is related to certain properties of graphs.

Graph classes closely related to the C1P or the Circ1P. Given a graph G, there are several "natural" ways to map G to a matrix that represents all information about G. The following definition describes the most common types of such matrices that represent graphs.

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

Definition 2.1. Let $G = (V, E_G)$ be a graph with $V = \{v_1, \ldots, v_n\}$ and $E_G = \{e_1, \ldots, e_m\}$, and let $H = (V_1, V_2, E_H)$ be a bipartite graph with $V_1 = \{u_1, \ldots, u_{n_1}\}$ and $V_2 = \{w_1, \ldots, w_{n_2}\}$.

1. The *adjacency matrix* of G is the symmetric $n \times n$ binary matrix M with $m_{i,j} = 1$ if and only if $\{v_i, v_j\} \in E_G$.
2. The *augmented adjacency matrix* of G is the matrix obtained from G's adjacency matrix by setting the entries of the main diagonal to 1.
3. The *edge-vertex incidence matrix* of G is the $m \times n$ binary matrix M with $m_{i,j} = 1$ if and only if v_j is an endpoint of e_i. The transpose of the edge-vertex incidence matrix is called the *vertex-edge incidence matrix* of G.
4. Let $c_1, \ldots, c_k$ be the maximal cliques of G. The *maximal clique matrix* (also called *vertex-clique incidence matrix*) of G is the $n \times k$ binary matrix M with $m_{i,j} = 1$ if and only if v_i belongs to c_j.
5. The *half adjacency matrix* of H is the $n_1 \times n_2$ binary matrix M with $m_{i,j} = 1$ if and only if $\{u_i, w_j\} \in E_H$.

Figures 3 and 5 show the matrix types introduced in Definition 2.1. Clearly, a matrix M is the half adjacency matrix of a bipartite graph H iff H is the representing graph of M.

Some elementary graph classes that are directly related to the properties C1P and Circ1P are defined as follows.

Definition 2.2. 1. A graph is *convex-round* if its adjacency matrix has the Circ1P, and it is *concave-round* if its augmented adjacency matrix has the Circ1P [7].

2. A graph G is an *interval graph* if its vertices can be mapped to intervals on the real line such that two vertices are adjacent if and only if their corresponding intervals overlap [9, 52]. If all intervals have the same length, then G is a *unit interval graph*; if no interval properly contains another interval, then G is a *proper interval graph*.
3. A graph G is a *circular-arc graph* if its vertices can be mapped to a set A of arcs on a circle such that two vertices are adjacent if and only if their corresponding arcs overlap. A circular-arc graph G is a *Helly circular-arc graph* if for every subset $A' \subseteq A$ it holds that $(\forall a_1, a_2 \in A' : a_1 \cap a_2 \neq \emptyset) \Rightarrow \bigcap_{a \in A'} a \neq \emptyset$.
4. A bipartite graph is *convex* if its half adjacency matrix has the C1P, it is *biconvex* if its half adjacency matrix has the C1P both for rows and for columns, and it is *circular convex* if its half adjacency matrix has the Circ1P.

See Figures 3 and 4 for illustrations. Interval graphs and circular-arc graphs are known in graph theory for a long time; they are well-studied (alone for the recognition problem of these graphs there exists a number of results, see [13,

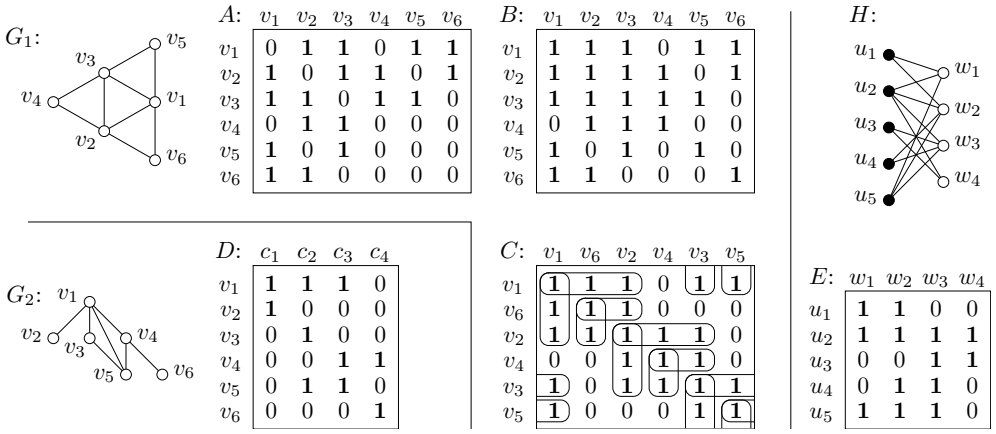


Figure 3: Matrices defined in Definition 2.1. Matrix A is the adjacency matrix of the graph G_1, and Matrix B is the augmented adjacency matrix of G_1. Matrix C is obtained from B by permuting the rows and columns; the shapes enclosing its 1-entries illustrate the quasi Circ1P (see Definition 2.3), which will be used in Table 1. (Actually, the matrix C shows not only that B has the quasi Circ1P, but also that B has the Circ1P.) Matrix D is the maximal clique matrix of G_2, and Matrix E is the half adjacency matrix of the bipartite graph H. Matrix C shows that G_1 is a concave-round graph (see Definition 2.2) as well as a circular-arc graph (see Table 1), Matrix D shows that G_2 is an interval graph (see Table 1), and Matrix E shows that H is a convex bipartite graph (see Definition 2.2).

23, 38, 47, 58, 62, 66, 69] for the recognition of interval graphs and [31, 59, 65, 73, 92] for the recognition of circular-arc graphs) and have applications in many fields [15, 46]. One reason for the attention that these two graph classes attract is that many problems that are NP-complete on general graphs (for example, Independent Set) are polynomial-time solvable on interval graphs and circular-arc graphs and also on the other graph classes mentioned in Definition 2.2 (see [1, 15, 46]). This important fact carries over to matrices with the C1P or the Circ1P, where many in general NP-hard matrix problems can be solved in polynomial time [22, 32, 70, 78, 93] (see also Section 4 and [75, 76, 86]). We summarize the relationships between the graph classes of Definition 2.2 on the one hand and the C1P or Circ1P occurring in the matrices of Definition 2.1 on the other hand in Table 1. Note that, since proper interval graphs coincide with unit interval graphs [85, 40], there is only one row for both classes. The property "quasi Circ1P" occurring in the table is defined as follows.

Definition 2.3 ([90])**.** A symmetric matrix has the *quasi Circ1P* if (possibly after permuting the rows and columns without destroying the symmetry) for every 1-

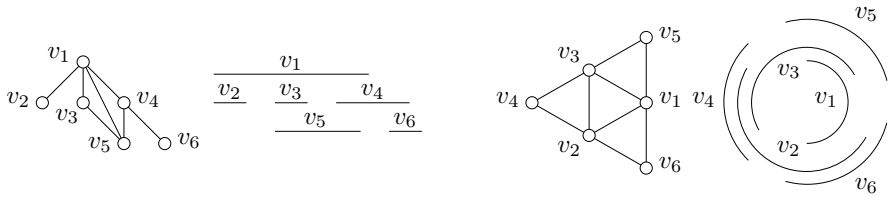


Figure 4: Left: An interval graph (which is not a proper or unit interval graph) and a set of intervals representing its vertices as described in Definition 2.2. Right: A circular-arc graph (which is not a Helly circular-arc graph) and a set of arcs representing its vertices as mentioned in Definition 2.2.

entry $m_{i,j}$ it holds that $m_{i,i} = m_{i,(i+1) \underline{\bmod} n} = m_{i,(i+2) \underline{\bmod} n} = \ldots = m_{i,(j-1) \underline{\bmod} n} = m_{i,j} = 1$ or that $m_{j,j} = m_{(j+1) \underline{\bmod} n,j} = m_{(j+2) \underline{\bmod} n,j} = \ldots = m_{(i-1) \underline{\bmod} n,j} = m_{i,j} = 1$.

Thereby, $i \underline{\bmod} j$ is defined as $i \underline{\bmod} j = \begin{cases} i \bmod j & \text{if } i \bmod j > 0 \\ j & \text{if } i \bmod j = 0 \end{cases}$ with $i \bmod j$ denoting the remainder of the division i by j. The quasi Circ1P is illustrated in Figure 3. The Circ1P always implies the quasi Circ1P [90].

Table 1 does not contain graphs whose vertex-edge incidence matrix or edge-vertex-edge incidence matrix has the C1P; however, these graphs also have a very special structure.

Definition 2.4. A *caterpillar* is a tree in which every non-leaf vertex has at most two non-leaf neighbors.

The two characterizations given in the following theorem follow directly from the results of Tucker [91] described in the following paragraph and from the considerations of Hajiaghayi and Ganjali [51] and Tan and Zhang [88].

Theorem 2.1. *A graph is a union of vertex-disjoint paths if and only if its edge-vertex incidence matrix has the C1P. A graph is a union of vertex-disjoint caterpillars if and only if its vertex-edge incidence matrix has the C1P.*

See Figure 5 for an illustration.

Tucker's characterization of matrices having the C1P. Matrices with the C1P can be characterized by a set of forbidden submatrices: A matrix has the C1P iff it does not contain a matrix from this set as a submatrix. Such a characterization is very helpful when regarding matrix modification problems where one has to modify a matrix to achieve the C1P (see Section 1).

The characterization by Tucker [91] is based on a characterization of convex bipartite graphs in terms of so-called asteroidal triples defined as follows.

Table 1: Relationship between graph classes and matrix properties. The symbol "⇒" expresses that the membership in a graph class implies the matrix property for the matrix associated with the corresponding graph; the symbols "⇐" and "⇔" denote implications in the back direction and in both directions, respectively. The abbreviation "C1P r+c" stands for "C1P for rows *and* for columns." Of course, due to its symmetry an (augmented) adjacency matrix has the C1P or the Circ1P for rows iff it has the C1P or the Circ1P, respectively, for columns.

graph class	adjacency matrix	augmented adjacency matrix	half adjacency matrix	maximal clique matrix
convex-round	⇔ Circ1P (per def.)			
concave-round		⇔ Circ1P (per def.)		
∩l				
circular-arc		⇔ quasi Circ1P [90] ⇐ Circ1P [90]		⇐ Circ1P
∪l				
Helly circular-arc		⇒ quasi Circ1P ⇐ C1P		⇔ Circ1P [43]
∪l				
interval		⇒ quasi Circ1P ⇐ C1P		⇔ C1P [38]
∪l				
proper / unit interval		⇔ C1P [85]		⇔ C1P r+c [35]
circular convex bipart.			⇔ Circ1P (per def.)	
∪l				
convex bipart.			⇔ C1P (per def.)	
∪l				
biconvex bipart.			⇔ C1P r+c (per def.)	

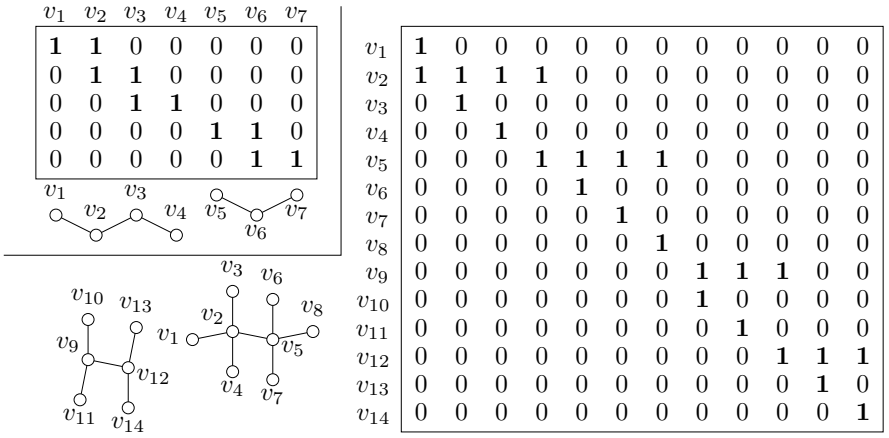


Figure 5: Top left: A graph that is a union of vertex-disjoint paths, and its edge-vertex incidence matrix. Bottom and right: A graph that is a union of vertex-disjoint caterpillars, and its vertex-edge incidence matrix. Both matrices have the C1P.

Definition 2.5. Let $G = (V, E)$ be a graph. Three vertices from V form an *asteroidal triple* if between any two of them there exists a path in G that does not contain any vertex from the closed neighborhood of the third vertex.

For example, every cycle of length at least six contains several asteroidal triples. More examples for graphs containing asteroidal triples are shown in Figure 6. In his characterization, Tucker does not use the term "convex bipartite graph"; however, convex bipartite graphs are identical to "*graphs with a V_2-consecutive*[1] *arrangement.*" The following two theorems, hence, characterize convex bipartite graphs.

Theorem 2.2 ([91, Theorem 6])**.** *A bipartite graph $G = (V_1, V_2, E)$ has a V_2-consecutive*[1] *arrangement if and only if V_2 contains no asteroidal triple of G.*

Theorem 2.3 ([91, Theorem 7])**.** *In a bipartite graph $G = (V_1, V_2, E)$ the vertex set V_2 contains no asteroidal triple*[1] *if and only if G contains none of the graphs G_{I_k}, G_{II_k}, G_{III_k} (with $k \geq 1$), G_{IV}, and G_{V} as shown in Figure 6.*

A characterization that is very similar to the one given in Theorem 2.2 is also known for interval graphs: A graph is an interval graph iff it is chordal and contains no asteroidal triple [71]. The following theorem, which finally characterizes matrices with the C1P, is a direct consequence of Theorems 2.2 and 2.3.

[1]Tucker considers the C1P for columns, whereas we describe the C1P for rows. Hence, the roles of V_1 and V_2 are interchanged here compared to Tucker's publication [91].

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

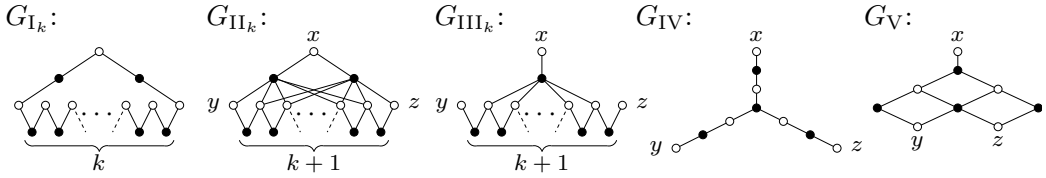


Figure 6: Forbidden induced subgraphs due to Tucker [91]: The vertex set V_2 of a bipartite graph $G = (V_1, V_2, E)$ contains an asteroidal triple iff G contains one of the displayed graphs as an induced subgraph, where white vertices correspond to vertices in V_2. The numbers k and $k + 1$ refer to the number of black vertices in the lower parts of the first three graphs. In the case of the graph $G_{\mathrm{I}_k} \in T$, every triple of white vertices is an asteroidal triple. In all other cases, there is exactly one asteroidal triple consisting of white vertices; this triple is denoted by x, y, z.

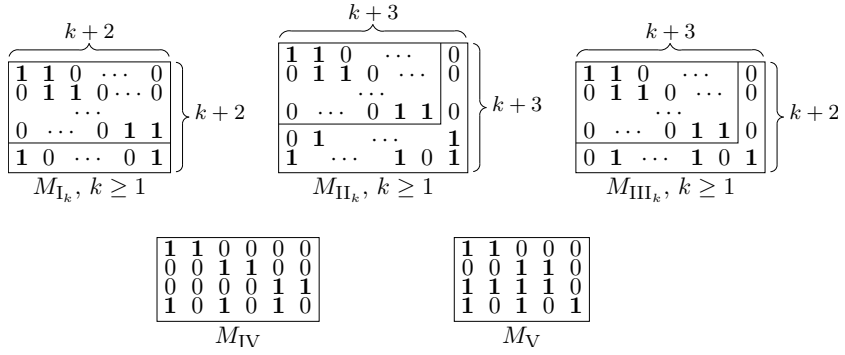


Figure 7: The forbidden submatrices for the C1P due to Tucker [91], given in Theorem 2.4.

Theorem 2.4 ([91, Theorem 9])**.** *A matrix M has the C1P if and only if it contains none of the matrices M_{I_k}, M_{II_k}, M_{III_k} (with $k \geq 1$), M_{IV}, and M_{V} as shown in Figure 7 as a submatrix.*[2]

Clearly, if a matrix M does not have the C1P, then it is possible to find in polynomial time one of the submatrices M_{I_k}, M_{II_k}, M_{III_k}, M_{IV}, and M_{V} in M: just check for every row and every column of M whether the matrix that results from the deleting the row or column, respectively, still does not have the C1P. It is also possible to find in polynomial time one of these submatrices that has a minimum number of rows, a minimum number of columns, a minimum number of rows and columns, or a minimum number of entries [24]; however, we are not aware of a linear-time algorithm for this task. To our knowledge, there is no characterization

[2]The roles of rows and columns are interchanged here compared to Tucker's publication [91].

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

similar to Theorem 2.4 for matrices with the Circ1P, although there is a kind of refinement of Theorem 2.4 with regard to the Circ1P [26, Theorem 4].

3 Recognizing the C1P

The C1P can be recognized in linear time. In this section, we survey several of the recognition algorithms. As we will see (Theorem 3.2), every algorithm that recognizes interval graphs can also be used to recognize matrices with the C1P. However, here we mention only those results that explicitly deal with matrices and the C1P, because first transforming a matrix into a graph and then testing whether this graph is an interval graph does not automatically yield an efficient (that is, linear-time) algorithm for recognizing the C1P. Some of the publications surveyed in this section consider the C1P for rows and some the C1P for columns; to give a consistent presentation, we formulate all results in terms of the C1P for rows. The following definition introduces some terms needed in this section.

Definition 3.1. Two rows r_1, r_2 of a 0/1-matrix *overlap* if there exist three columns c_1, c_2, c_3 such that column c_1 contains a 1 in both rows r_1 and r_2, column c_2 contains a 1 in r_1 but not in r_2, and column c_3 contains a 1 in r_2 but not in r_1.

A column c_1 *contains* a column c_2 if for every row r it holds that if c_2 has a 1 in row r then c_1 also has a 1 in row r. If a column is not contained in any other column, it is *maximal*.[3]

Using overlapping rows. The first polynomial-time algorithm to recognize matrices having the C1P[4] was presented by Fulkerson and Gross [38]. Their idea is to decompose the input matrix M into disjoint row sets in such a way that the whole matrix has the C1P iff each matrix induced by one of the row sets has the C1P. The partitioning of M's rows into different row sets is performed by defining an *overlap graph* $\mathcal{G}(M)$: Every vertex of this graph corresponds to a row of M, and two vertices are connected iff their corresponding rows overlap. Every connected component of this graph defines one row set of the partition of M needed by the algorithm. Now, for the columns of every submatrix of M that is induced by one of the row sets of the partition, a C1-ordering can easily be found, if existing, by considering one row after the other in a certain order and re-arranging the columns if necessary. In the last phase of the algorithm, the column orderings

[3]Note that this definition of a maximal column does not allow the existence of two identical maximal columns.

[4]Fulkerson and Gross [38] as well as Booth and Lueker [13] consider the C1P for columns, whereas we describe the C1P for rows. Hence, the roles of rows and columns are interchanged here compared to the original publications.

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

computed for the submatrices are combined, while possibly re-arranging some of the columns. The whole matrix M has the C1P iff all of the submatrices have the C1P [38, Theorem 4.1]. The whole procedure takes polynomial time. A more recent linear-time algorithm by Hsu [61] for recognizing the C1P is based on very similar ideas.

PQ-Trees. Booth and Lueker were the first to present a linear-time algorithm for recognizing matrices with the C1P[4] [13]. Linear time means a running time that is linear in the number of columns plus the number of rows plus the number of 1-entries of the given matrix. Booth and Lueker introduced so-called *PQ-Trees*, which are not only useful for recognizing matrices with the C1P, but also, for example, for recognizing matrices with the Circ1P and for recognizing interval graphs and planar graphs. In the context of recognizing matrices with the C1P, a PQ-tree is an ordered rooted tree that represents a C1-ordering for the columns of a matrix M. To this end, the inner nodes of the PQ-tree are labeled as P-nodes and Q-nodes, and the leaves one-to-one correspond to the columns of the underlying matrix M. Ordering the columns of M in the same way as their corresponding leaves in the PQ-tree yields the strong C1P. In addition, any PQ-tree for M implicitly represents *all* possible C1-orderings for M's columns because by applying a series of certain node reordering operations every possible PQ-tree for the set of M's columns can be transformed into any other possible PQ-tree for this column set. Therefore, PQ-trees have the following properties.

1. If T is a PQ-tree for a matrix M, then the sequence of T's leaves from left to right describes a C1-ordering for M's columns.
2. Each C1-ordering for M's columns one-to-one corresponds to a PQ-tree.
3. The set of PQ-trees for the columns of a matrix is closed under the following two operations:

   - Arbitrarily reordering the children of a P-node.
   - Putting the children of a Q-node in reverse order.

   In particular, none of these two operations destroys property 1.

See Figure 8 for an illustration. In order to either construct a PQ-tree for a given matrix M or decide that M does not have the C1P, the algorithm of Booth and Lueker starts with a tree (the so-called *universal PQ-tree*) consisting of one P-node forming the root and one leaf node for every column of M. The algorithm considers the rows of M one after the other, and in every step it either reports that M does not have the C1P, or it modifies the tree, by using a complicated case distinction, in such a way that the resulting tree is a PQ-tree for the matrix that is induced by the rows considered so far.

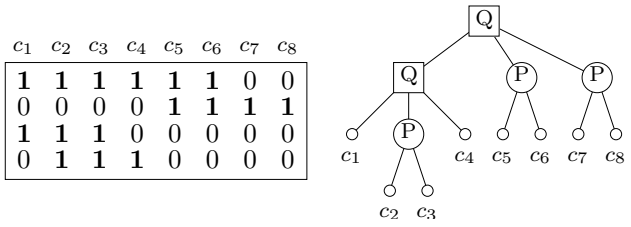


Figure 8: A matrix M and the PQ-tree representing the column ordering of M.

Variations of PQ-Trees. Several variations of PQ-trees have been proposed since their first appearance. Korte and Möhring [66] introduced *MPQ-trees* ("modified PQ-trees"), where the inner nodes contain some additional information, which results in a simpler construction of these trees. Meidanis et al. [77] defined *PQR-trees*, which are a generalization of PQ-trees in the following sense: For every matrix M that has the C1P for columns, the set of PQR-trees for M's rows is identical with the set of PQ-trees for M's rows. However, in contrast to PQ-trees, PQR-trees are also defined for matrices that do not have the C1P for columns; in this case they contain, in addition to P-nodes and Q-nodes, inner nodes labeled as R-nodes, which can be useful for identifying why the matrix does not have the C1P. A similar approach was used by McConnell [74]. He introduced *generalized PQ-trees* in order to determine if a matrix has the C1P for rows and, if it does not, to generate a "certificate" therefor: Such a certificate is a small (compared to the size of the input matrix) proof that can be verified by a "fast and uncomplicated" polynomial-time algorithm (for more details about such certificates see [69]). The certificate produced by the algorithm of McConnell [74] for an $m \times n$ input matrix M consists of an odd cycle of length at most $n+3$ in the so-called *incompatibility graph* of M. This graph is defined as the graph $G = (V, E)$ with

$$\begin{aligned} V =& \{(j_1, j_2) \mid 1 \le j_1 \le n \ \wedge \ 1 \le j_2 \le n \ \wedge \ j_1 \ne j_2\} \\ E =& \{\{(j_1, j_2), (j_2, j_1)\} \mid 1 \le j_1 < j_2 \le n\} \cup \\ & \{\{(j_1, j_2), (j_2, j_3)\} \mid j_1 \ne j_3 \wedge \\ & \exists i \in \{1, \ldots, m\} : (m_{i,j_1} = m_{i,j_3} = 1 \ \wedge \ m_{i,j_2} = 0)\}. \end{aligned}$$

Intuitively speaking, the incompability graph contains two vertices for every pair of columns of M: one vertex for every possible relative ordering of the two columns. If two vertices (j_1, j_2) and (j_3, j_4) in the incompatibility graph are connected by an edge, then the corresponding two orderings conflict in the sense that there is no C1-ordering for M's columns that places the column j_1 to the left of j_2 *and* the column j_3 to the left of j_4. See Figure 9 for an example. The connection

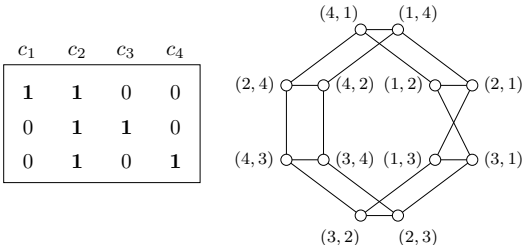


Figure 9: The matrix M_{III_1} and its incompability graph [74]. The graph contains several odd cycles of length 7.

between the incompability graph of a matrix and the C1P is specified in the following theorem. The original formulation of this theorem is due to McConnell [74] and contains a minor error concerning the cycle length.

Theorem 3.1 ([74, Theorem 6.1])**.** *An $m \times n$ matrix M has the C1P iff its incompatibility graph is bipartite. If M does not have the C1P, then its incompatibility graph has an odd cycle of length at most $n + 3$.*

PC-Trees. A remarkable simplification for building PQ-trees was exhibited by Hsu and McConnell [63], who introduced *PC-trees*. These trees can be seen as unrooted PQ-trees that represent Circ1-circular orderings for the columns of a matrix instead of C1-orderings. Instead of Q-nodes, PC-trees contain C-nodes; the order of the leaves of a PC-tree describes a Circ1-circular ordering for the columns of the underlying matrix. PC-trees have the following properties.

1. If T is a PC-tree for a matrix M, then any sequence obtained by considering T's leaves in clockwise or counter-clockwise order describes a Circ1-circular ordering for M's columns.
2. Each Circ1-circular ordering for M's columns one-to-one corresponds to a PC-tree.
3. The set of PC-trees for the columns of a matrix is closed under the following two operations:

   - Arbitrarily reordering the neighbors of a P-node.
   - First rooting T at a neighbor of a C-node v, then "flipping" the subtree whose root is v, and finally un-rooting the tree. Herein, "flipping" a subtree means putting the children of every node of the subtree in reverse order.

   In particular, none of these two operations destroys property 1.

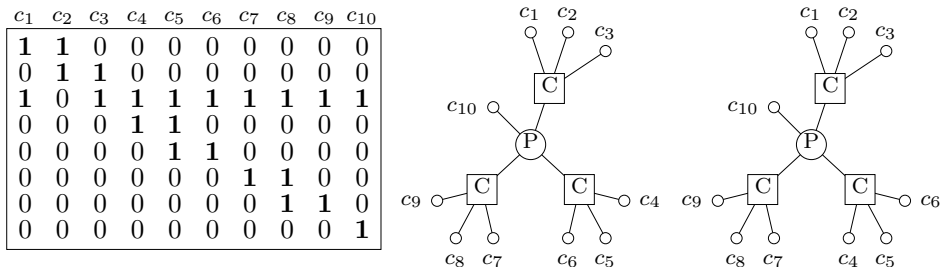


Figure 10: PC-trees for a matrix. Left: a matrix M. Middle: the PC-tree representing the circular column ordering of M. Right: The PC-tree obtained from the PC-tree in the middle by "flipping" the subtree rooted at the bottom-right C-node.

See Figure 10 for an illustration.

Like in the case of PQ-trees, there is a linear-time algorithm that either constructs a PC-tree for a given matrix M or decides that M does not have the Circ1P [63]. Similarly to the algorithm of Booth and Lueker [13], this algorithm starts with a tree consisting of one P-node which has one leaf neighbor for every column of M. The algorithm considers the rows of M one after the other, and in every step it either reports that M does not have the Circ1P, or it modifies the tree. These modifications, however, are much simpler than those proposed by Booth and Lueker for updating a PQ-tree.

Due to Corollary 1.2, the PC-tree algorithm can not only be used to decide whether a matrix has the Circ1P, but also to decide whether it has the C1P: Just add a column that contains only 0s to the given matrix. Rooting the PC-tree for the resulting matrix at the neighbor of the leaf node that corresponds to the newly inserted column and then deleting this leaf node yields a PQ-tree for the original matrix.

Further recognition algorithms. A simple linear-time algorithm without using any variant of PQ-trees was presented by Habib et al. [47]: They use a so-called "Lex-BFS ordering" of the vertices of a graph to decide in linear time whether the graph is an interval graph. Habib et al. also prove Theorem 3.2 below, which implies that any algorithm recognizing interval graphs can also be used for recognizing matrices with the C1P. Habib et al. show that the recognition of matrices with the C1P in this way is possible in linear time.

For describing how an algorithm recognizing interval graphs can be used to decide whether a given matrix M has the C1P, let $G_{\text{ra}}(M)$ denote the graph that has one vertex for every row of M and where two vertices are adjacent iff their corresponding rows in M have a 1 in a common column; we call this graph the

row adjacency graph of M. Note that several matrices can have identical row adjacency graphs; moreover, if a matrix M is the maximal clique matrix of a graph G, then G is the row adjacency graph of M. The latter observation leads to the following finding.

Theorem 3.2 ([47, Theorem 2])**.** *For a* 0/1*-matrix M the following statements are equivalent:*

1. *The row adjacency graph $G_{ra}(M)$ is an interval graph and M is its maximal clique matrix.*
2. *The columns of M are maximal and M has the C1P for rows.*

By appending a size-$n \times n$ unit matrix I to a matrix M, a matrix $\tilde{M} = \begin{pmatrix} M \\ I \end{pmatrix}$ can be constructed, which obviously has the following properties: $\tilde{M}$ has the C1P iff M has the C1P, and every column of $\tilde{M}$ is maximal.[5] Testing whether M has the C1P now obviously reduces to checking whether $G_{\mathrm{ra}}(\tilde{M})$ is an interval graph and $\tilde{M}$ is its maximal clique matrix. However, note that using Theorem 3.2 in combination with a linear-time algorithm for recognizing interval graphs does not automatically yield a *linear-time* algorithm for recognizing matrices with the C1P. Nevertheless, Habib et al. give an algorithm using Theorem 3.2 for recognizing matrices with the C1P in linear time.

Clearly, if the columns *and* rows of a matrix M shall be permuted such that the resulting matrix contains at most one block of 1s per row *and* column, this can be done in linear time by first permuting the columns and then permuting the rows. However, weighted variants of this problem can be NP-hard [83]. Finally, we like to point out that recognizing matrices that have "almost the C1P" is much more difficult than recognizing matrices with the C1P: While matrices with the C1P can be recognized in linear time, the problem of deciding whether the columns of a matrix M (not having the C1P) can be permuted in such a way that the overall number of blocks of 1s is at most k is NP-complete [41, 48] (see also [49]); deciding whether M's columns can be permuted such that the number of blocks of 1s in each row is at most c is NP-complete for every constant $c \geq 2$ [6, 36, 45, 96] (see also [12, 97]). However, there are algorithms for recognizing matrices that are "close" to having the C1P in some other sense [60, 72].

Deciding whether matrix can obtain the C1P by deleting a given number d of rows or columns is NP-complete [41, 50, 51, 88]—however, for special cases there are approximation and fixed-parameter algorithms [26] (see [24] for a detailed overview). In related problems, 0-entries have to be flipped (that is, replaced

[5] In the original paper, published in the Bulletin of the EATCS, 98:27–59, there is an error in the description of $\tilde{M}$'s properties.

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

by 1-entries) in order to obtain the C1P [41, 94], or arbitrary entries have to be flipped [82]. For all variants of these matrix modification problems as well as for the recognition of matrices being "close" to the C1P, (further) approximation and fixed-parameter results would be desirable.

4 Hard Problems on Instances with the C1P/Circ1P

Many in general NP-hard problems become polynomial-time solvable when restricted to inputs that have the C1P. In this section, we consider the in general NP-complete problems Integer Linear Programming and Set Cover.

4.1 Integer Linear Programming with Coefficient Matrices having the C1P/Circ1P

Here, we consider the connection between matrices with the C1P or Circ1P and the hardness of solving integer linear programs (ILPs). While solving ILPs in general is NP-hard (see Section 4.1.1), we are here interested in the solvability of ILPs in the special case where the matrix consisting of the coefficients of the inequations has the C1P—such ILPs occur, for example, in biological applications [3]—or the Circ1P.

The algorithms that we consider do not only work for ILPs with 0/1-coefficient matrices, but also for ILPs whose coefficient matrices consist of entries from $\{0, 1, -1\}$. Therefore, we extend the definition of the C1P to $0/\pm1$-matrices as follows: A $0/\pm1$-matrix has the C1P if every row contains only entries either from $\{0, 1\}$ or from $\{0, -1\}$ and the columns can be permuted such that in every row the non-zero entries appear consecutively. The Circ1P for $0/\pm1$-matrices is defined analogously. We refer to Schrijver [87] for more details on (integer) linear programming.

4.1.1 (Integer) Linear Programming Basics

A *linear program (LP)* is an instance of the following optimization problem.

Linear Programming
Input: An $m \times n$ matrix $A = (a_{i,j})$, an n-entry column vector $\vec{b}$ and an m-entry row vector $\vec{c}^{\,\mathrm{T}}$ with all entries in $A, \vec{b}, \vec{c}^{\,\mathrm{T}}$ from $\mathbb{Q}$.
Task: Find an m-entry column vector $\vec{x}$ that satisfies $A\vec{x} \leq \vec{b}$ and maximizes $\vec{c}^{\,\mathrm{T}}\vec{x}$.

In other words, the task is to assign values to a set of variables $x_1, \ldots, x_n$ (the entries of the vector $\vec{x}$) such that a set of m inequations (given by the matrix A and

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

the vector $\vec{b}$) are satisfied and that the value of a linear *objective function* (given by the vector $\vec{c}^{\,\mathrm{T}}$) is maximized. An assignment of values to the variables that satisfies the given inequations is called a *(feasible) solution* for the LP. A vector containing only integers is called *integral*. The variant of Linear Programming where only integral solutions are allowed is called Integer Linear Programming; its instances are called *integer linear programs (ILPs)*. There are also decision problems corresponding to the optimization problems Linear Programming and Integer Linear Programming. The instances of these decision problems contain no objective function (no vector c); the task is to decide if there is any feasible solution.

Linear Programming can be solved in polynomial time. In particular, there is an algorithm for solving LPs that needs $O((n^3/\ln n)L)$ arithmetic operations [4] (see also [84] for an overview over efficient algorithms for solving LPs), where L is the total bit number of the input. In contrast, it is easy to see that Integer Linear Programming is NP-hard. Actually, the decision version of Integer Linear Programming is NP-complete [14, 95, 64, 57].

An LP (ILP) is called *feasible* if it admits a feasible solution, and *unfeasible* otherwise. Given an LP on n variables, one can interpret its solution space as an n-dimensional Eukledian space; every inequation of the LP defines a half-space that contains all value-to-variable assignments satisfying this inequation. The intersection of all the half-spaces defined by the inequations of an LP is called the *polyhedron* defined by the LP. If the polyhedron defined by an LP is *integral*, which means that each of its corners corresponds to an integral solution, then the ILP defined by the matrix A and the vectors $\vec{b}, \vec{c}^{\,\mathrm{T}}$ of the LP can be solved in polynomial time by solving the LP.

Given an LP *Maximize* $\vec{c}^{\,T}\vec{x}$ *subject to* $A\vec{x} \leq \vec{b}$, then the problem *Minimize* $\vec{z}^{\,T}\vec{b}$ *subject to* $\vec{z}^{\,T}A = \vec{c}^{\,T}, \vec{z}^{\,T} \geq \vec{0}^{\,T}$ is called the *dual (problem)* for the given LP—here the task is to find an optimal row vector $\vec{z}^{\,\mathrm{T}}$. The dual problem has a well-defined optimal solution iff the original LP has a well-defined optimal solution; if this is the case then it holds that $\max\{\vec{c}^{\,\mathrm{T}}\vec{x} \mid A\vec{x} \leq \vec{b}\} = \min\{\vec{z}^{\,\mathrm{T}}\vec{b} \mid \vec{z}^{\,\mathrm{T}}A = \vec{c}^{\,\mathrm{T}} \ \wedge \ \vec{z}^{\,\mathrm{T}} \geq \vec{0}^{\,\mathrm{T}}\}$.

Note that if the coefficient matrix of an LP has the C1P (Circ1P) for columns, then the coefficient matrix of its dual has the C1P (Circ1P) for rows. Therefore, all algorithms described below can be used for solving ILPs whose coefficient matrices have the C1P (Circ1P) for rows *or* the C1P (Circ1P) for columns.

4.1.2 Totally Unimodular Matrices

The problem Integer Linear Programming is NP-hard. However, there are special cases that can be solved in polynomial time. Here, we consider the special case where the coefficient matrix A of the ILP is "totally unimodular"—in this case, the polyhedron defined by $A\vec{x} \leq \vec{b}$ is integral for every integral vector $\vec{b}$. We will

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

see that matrices with the C1P have this property. For a more detailed description of classes of matrices see [15, 21, 46, 87].

Definition 4.1. A $0/\pm 1$-matrix is *totally unimodular* if every square submatrix has determinant 0, 1, or -1.

There is a useful "bicolorability" characterization for totally unimodular matrices.

Theorem 4.1 ([44])**.** *An $m \times n$ matrix A with entries $0, 1, -1$ is totally unimodular if and only if each collection of columns from A can be partitioned into two column sets such that in each row the sum of the entries of the first set and the sum of the entries of the second set differ by at most 1.*

The following theorem shows the polynomial-time solvability of ILPs with totally unimodular coefficient matrices.

Theorem 4.2 ([56])**.** *Let A be an $m \times n$ integral matrix. Then the polyhedron defined by $A\vec{x} \leq \vec{b}, \vec{x} \geq \vec{0}$ is integral for every integral vector $\vec{b} \in \mathbb{Z}^m$ if and only if A is totally unimodular.*

From Theorem 4.2 it follows that for every totally unimodular matrix A, every integral vector $\vec{b}$ and every vector $\vec{c}^{\,\mathrm{T}}$ the ILP

$$\begin{array}{lrl} \text{Maximize} & \vec{c}^{\,\mathrm{T}}\vec{x} & \\ \text{subject to} & A\vec{x} & \leq \vec{b} \\ & \vec{x} & \geq \vec{0} \\ & \vec{x} & \text{is integral} \end{array} \tag{1}$$

can be solved with $O((n^3/\ln n)L)$ arithmetic operations [4], where L is the total bit number needed for encoding the ILP.

In the case of totally unimodular coefficient matrices A, the polyhedron defined by $A\vec{x} \leq \vec{b}, \vec{x} \geq \vec{0}$ is integral for *every* integral vector $\vec{b}$. There exist also matrices with the property that the polyhedron is integral only for certain vectors $\vec{b}$, these matrices are called *balanced* [10, 11, 20, 39, 89].

4.1.3 ILPs with Coefficient Matrices having the C1P

The first method to solve ILPs whose coefficient matrices have the C1P is to use the fact that any matrix A having the C1P clearly fulfills the conditions of Theorem 4.1 and, hence, is totally unimodular. To see this, consider an arbitrary collection of columns from A and order them according to the C1P. Partitioning the columns by putting every second column, starting with the first, into one column

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

set and every remaining column into the other column set leads to a partitioning as required in Theorem 4.1 (see also [78, page 544]). Therefore, if a matrix A has the C1P, then for every integral vector $\vec{b}$ and every vector $\vec{c}^{\,\mathrm{T}}$ the ILP shown in (1) can be solved in polynomial time due to Theorem 4.2.

Using Theorem 4.2 to solve ILPs with coefficient matrices having the C1P exploits only the fact that such coefficient matrices are totally unimodular. However, it is known that an ILP whose coefficient matrix has the C1P can be solved even faster by transforming it into an edge-weighted graph and solving a shortest-path problem or a minimum-cost flow problem on this graph, depending on whether the decision version or the optimization version of Integer Linear Programming is considered (see [93] and [2, pages 304–306] for the transformation of the optimization version into the flow problem and [2, pages 310–315] for the connection between minimum-cost flow problems and shortest-path problems; see also [78, pages 546–550]). The running time obtained in this way is $O(mn)$ for the decision version and $O(m^2 \log(n) + mn \log(n)^2)$ for the optimization version; this approach is, therefore, much faster than using Theorem 4.2, where $O((n^3/\ln n)L)$ operations [4] are needed—L is the size of the ILP and, hence, lower-bounded by mn.

We start with showing how to solve the decision version of Integer Linear Programming for coefficient matrices with the C1P by reducing it to a shortest path problem. We assume that the coefficient matrix A of the given ILP has m rows and that the rows of A and the entries of $\vec{b}$ are sorted in such a way that the first $m' \leq m$ of these rows contain only entries from $\{0, 1\}$ and the remaining $m - m'$ rows contain only entries from $\{0, -1\}$. Moreover, we assume that A has the strong C1P, which is not a restriction since a C1-ordering for A's columns can be found in linear time (see Section 3). With $\mathrm{lx}(i)$ and $\mathrm{rx}(i)$ we denote the column index of the first and the last, respectively, non-zero entry in the ith row of A. Hence, an instance of the problem to be solved consists of an inequation system as follows.

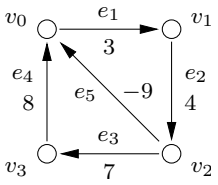
$$\begin{aligned} x_{\mathrm{lx}(i)} + x_{\mathrm{lx}(i)+1} + \ldots + x_{\mathrm{rx}(i)} &\leq b_i \quad \forall i \in \{1, \ldots, m'\} \\ -x_{\mathrm{lx}(i)} - x_{\mathrm{lx}(i)+1} - \ldots - x_{\mathrm{rx}(i)} &\leq b_i \quad \forall i \in \{m'+1, \ldots, m\} \\ x_j &\in \mathbb{Z} \quad \forall j \in \{1, \ldots, n\} \end{aligned} \tag{2}$$

To transform the inequation system into a graph, we first drop the constraint of integrality and replace the n variables $x_1, \ldots, x_n$ by $n + 1$ variables $y_0, \ldots, y_n$ such that $x_j = y_j - y_{j-1}$ for all $j \in \{1, \ldots, n\}$. This yields the following inequation system.

$$\begin{aligned} -y_{\mathrm{lx}(i)-1} + y_{\mathrm{rx}(i)} &\leq b_i \quad \forall i \in \{1, \ldots, m'\} \\ y_{\mathrm{lx}(i)-1} - y_{\mathrm{rx}(i)} &\leq b_i \quad \forall i \in \{m'+1, \ldots, m\} \end{aligned} \tag{3}$$

In the resulting coefficient matrix, each row contains exactly one 1 and one -1; hence, each row can be interpreted as a directed edge in a graph G whose vertices correspond to the variables $y_0, \ldots, y_n$. More precisely, let $G = (V, E)$ be the

$$
\begin{array}{rrrrll}
-y_0 & +y_1 & & & \leq 3 & \text{(interpreted as edge } e_1\text{)} \\
 & -y_1 & +y_2 & & \leq 4 & \text{(interpreted as edge } e_2\text{)} \\
 & & -y_2 & +y_3 & \leq 7 & \text{(interpreted as edge } e_3\text{)} \\
y_0 & & & -y_3 & \leq 8 & \text{(interpreted as edge } e_4\text{)} \\
y_0 & & -y_2 & & \leq -9 & \text{(interpreted as edge } e_5\text{)}
\end{array}
$$



$$
\begin{array}{rrrrrll}
-z_1 & & & +z_4 & +z_5 & = 0 & \text{(interpreted as flow conservation at } v_0\text{)} \\
z_1 & -z_2 & & & & = 0 & \text{(interpreted as flow conservation at } v_1\text{)} \\
 & z_2 & -z_3 & & -z_5 & = 0 & \text{(interpreted as flow conservation at } v_2\text{)} \\
 & & z_3 & -z_4 & & = 0 & \text{(interpreted as flow conservation at } v_3\text{)} \\
3z_1 & +4z_2 & +7z_3 & +8z_4 & -9z_5 & < 0 & \\
 & & & z_1, \ldots, z_5 & & \geq 0 &
\end{array}
$$

Figure 11: Solving an ILP whose coefficient matrix has the C1P. Top left: An example for the inequation system (3) obtained from an ILP with the C1P. Every row can be interpreted as an edge in a directed, edge-weighted graph G. Top right: The graph corresponding to the ILP. Bottom: This inequation system is *not* feasible iff the the inequation system displayed at the top of the figure is feasible (Farkas' Lemma). The inequation system at the bottom can be interpreted as a flow problem.

directed edge-weighted graph with

$$
\begin{aligned}
V = \{v_j \mid & \text{ the inequation system (3) contains a variable } y_j\}, \\
E = \{(v_{j_1}, v_{j_2}) \mid & \text{ the inequation system (3) contains an inequation} \\
& \text{whose left side is } -y_{j_1} + y_{j_2}\},
\end{aligned}
$$

where every edge $e \in E$ has a weight that is equal to the right side of the inequation corresponding to e in the inequation system (3), see Figure 11.

Now consider the following statement known as *Farkas' Lemma* (see [87]).

Lemma 4.1. *Let A be an $m \times n$ matrix with entries from $\mathbb{R}$, and let $\vec{b} \in \mathbb{R}^m$ be a vector. Then the inequation system $A\vec{y} \leq \vec{b}$ has a solution $\vec{y} \in \mathbb{R}^n$ if and only if the inequation system $\vec{z}^T A = (0^n)^T, \vec{z}^T \vec{b} < 0, \vec{z} \geq 0^m$ has* no *solution $\vec{z} \in \mathbb{R}^m$.*

Applying Farkas' Lemma to the inequation system (3), the lemma says that the inequation system is feasible iff G contains no *negative cycle*, that is, no directed cycle in which the sum of the edge weights is negative. To see this, observe that by interpreting the edge weights b_i as "per-flow-costs", the inequation system $\vec{z}^T A = (0^n)^T, \vec{z}^T \vec{b} < 0, \vec{z} \geq 0^m$ in Farkas' Lemma can be interpreted as the following "negative-cost" flow problem on the graph G: Find a flow function $f : E \to \mathbb{R}$ such that

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

- the flow $f(e_i)$ along every directed edge $e_i \in E$ is nonnegative (expressed by the constraint $\vec{z} \geq 0^m$),
- for every vertex the sum of the ingoing and the outgoing flow is 0 (expressed by the constraint $\vec{z}^{\mathrm{T}} A = (0^n)^{\mathrm{T}}$), and
- the sum $\sum_{i=1}^{m} b_i f(e_i)$ of all costs arising from sending flow along the edges is negative (expressed by the constraint $\vec{z}^{\mathrm{T}} \vec{b} < 0$).

See Figure 11. If a flow f has these three properties, then setting $z_i = f(e_i)$ clearly yields a feasible solution for the LP. Obviously, a flow with negative cost can only exist if the graph G contains a negative cycle. By using the Bellmann-Ford-Moore-Algorithm (see [22]), it can be decided in $O(|V| \cdot |E|)$ time whether G contains a negative cycle. Hence, the decision version of Integer Linear Programming with C1P can be decided in $O(n \cdot m)$ time.

If G contains no negative cycle and a solution for the inequation system (3) shall be constructed (that is, the values of the y_j shall be computed), then just select an arbitrary $k \in \{0, \ldots, n\}$ and set y_k to 0. For every $j \in \{0, \ldots, n\} \setminus \{k\}$ for which there exists no directed path from v_k to v_j in G, add an edge (v_k, v_j) of weight $|E| \cdot \max\{-b_i \mid i \in \{1, \ldots, m\} \ \wedge \ b_i < 0\}$. Note that this operation does not create any negative cycles; note also that in the resulting graph G' every vertex is reachable from v_k on a directed path. Now, for every $j \in \{0, \ldots, n\} \setminus \{k\}$, set y_j to the length of the shortest path in G' from v_k to v_j. Since G' contains no negative cycle, these shortest paths are all well-defined. It is easy to see that this solution satisfies all inequations of the inequation system (3). The shortest paths can be computed by the Bellmann-Ford-Moore-Algorithm in $O(|V| \cdot |E|) = O(n \cdot m)$ time. A solution for the original ILP (2) can be computed by setting $x_j = y_j - y_{j-1}$ for all $j \in \{1, \ldots, n\}$.

For solving the optimization version of the problem, one has to use a minimum-cost flow algorithm instead of a shortest path algorithm. Again, start with omitting the integrality constraint and replacing the variables $x_1, \ldots, x_n$ by variables $y_0, \ldots, y_n$. The dual of the resulting LP can now be interpreted as a minimum-cost flow problem on a directed, vertex-weighted and edge-weighted graph, which is constructed in analogy to the graph used for solving the decision problem, and in which every vertex has a (positive or negative) "flow demand". Computing a minimum-cost flow which respects the flow demands yields an optimal solution for the ILP. Such a minimum-cost flow can be found in $O(m^2 \log(n) + mn \log(n)^2)$ time [2].

4.1.4 ILPs with Coefficient Matrices having the Circ1P

Not all matrices that have the Circ1P are totally unimodular. For example, all matrices M_{I_k} (see Figure 7) with even k are totally unimodular, while all matrices M_{I_k}

with odd k are not (this can easily be seen by using the characterization of Theorem 4.1). Nevertheless, every ILP whose coefficient matrix has the Circ1P can be solved in polynomial time by solving a series of ILPs that all have the C1P [8] (see also [2, page 346–347] and [55]).

To solve a given ILP

$$\begin{array}{lrcll} \text{Maximize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n & & & \\ \text{subject to} & a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n & \leq & b_i & \forall i \in \{1,\ldots,m\} \\ & x_j & \in & \mathbb{Z} & \forall j \in \{1,\ldots,n\} \end{array} \tag{4}$$

whose $0/\pm1$-coefficient matrix $A = (a_{i,j})$ has the Circ1P, define L_k, $k \in \mathbb{Z}$, as the ILP that results from appending the constraint

$$x_1 + x_2 + \ldots + x_n = k \tag{5}$$

to the the ILP (4). It is obvious that the ILP (4) is feasible iff there is a value k such that L_k is feasible. Moreover, if the ILP (4) is feasible, then there is a k such that the optimal solution for L_k is an optimal solution for the ILP (4): just set k to the sum of the x_i in an optimal solution for the ILP (4). Now, any ILP L_k can be transformed into an ILP having the C1P: Add the equation (5) to every inequation of L_k whose coefficients are from $\{0,-1\}$ and in which the non-zero coefficients do not appear consecutively, and subtract the equation (5) from every inequation of L_k whose coefficients are from $\{0,1\}$ and in which the non-zero coefficients do not appear consecutively. The resulting ILP is equivalent to the ILP L_k (that is, every feasible solution of the latter ILP is a feasible solution for L_k and vice versa) and has the C1P—therefore, it can be solved with the approach described in Section 4.1.3. The optimum value of k can be determined by a binary search [2, 8], such that the number of ILPs that have to be solved is linear in the size of the given ILP (4).

4.2 Set Cover Problems and the C1P

The C1P has attracted interest not least because it often makes hard problems easy. Our first example substantiating this statement was Integer Linear Programming in Section 4.1. As a second example, we consider the problem Set Cover. Formulated as a matrix problem, Set Cover is defined as follows.

Set Cover
Input: A binary matrix M and a positive integer k.
Question: Is there a set C' of at most k columns of M such that C' contains at least one 1 from every row?[6]

Originally published in *Bulletin of the European Association for Theoretical Computer Science*, 98:27–59. EATCS, 2009.

Due to its generality, Set Cover has practical applications in almost all disciplines (see [22, 19, 16]); unfortunately, Set Cover is not only NP-hard, but it allows only for a logarithmic-factor polynomial-time approximation [34]. Moreover, Set Cover is W[2]-complete (that is, parameterized intractable) with respect to the parameter k = "solution size" [30]. In the *weighted version* of Set Cover, each column of the given matrix M has a positive integral weight and one asks for a column set C' of *weight* at most k.

4.2.1 Set Cover with the C1P/Circ1P

Whereas Set Cover in general is NP-complete, the problem becomes polynomial-time solvable when the input matrix M has the C1P or the Circ1P. To solve such a restricted instance of Set Cover, formulate the problem in a straightforward way as an ILP that has one variable for each column of M and whose coefficient matrix is M. As described in Section 4.1, this ILP is polynomial-time solvable. Moreover, there is a well-known greedy algorithm for Set Cover on input matrices with the C1P: First order the columns of M such that in each row the 1s appear consecutively (this takes linear time, see Section 3), and then proceed from left to right as follows: Repeatedly search for a row r with minimum $\mathrm{rx}(r)$, where $\mathrm{rx}(r)$ denotes the index of the rightmost column having a 1 in row r. Then take column $c_{\mathrm{rx}(r)}$ into the solution and delete all rows from M that have a 1 in column $c_{\mathrm{rx}(r)}$.

If the input matrix M has the C1P or the Circ1P, even the weighted version of Set Cover can be solved in polynomial time: one can either use the ILP approach or, in case of input matrices having the C1P, use a simple dynamic programming algorithm. The following theorem summarizes these results.

Theorem 4.3. Weighted Set Cover *can be solved in polynomial time if the input matrix has the C1P or the Circ1P (for rows or for columns).*

A further approach for tackling Set Cover is to use polynomial-time *data reduction rules*: Such a reduction rule takes as input an instance X of a problem and outputs in polynomial-time an instance X' with $|X'| \leq |X|$ of the same problem such that X' is a *yes*-instance iff X is a *yes*-instance.

The following data reduction rules for Set Cover are well-known; their correctness is obvious.

—If M contains two rows r_{i_1}, r_{i_2} such that for each column c_j it holds that $m_{i_1,j} = 1$ implies $m_{i_2,j} = 1$, then remove row r_{i_2} from M.

—If M contains two columns c_{j_1}, c_{j_2} such that for each row r_i it holds that $m_{i,j_1} = 1$ implies $m_{i,j_2} = 1$, then remove column c_{j_1} from M.

[6]The reader may be familiar with Set Cover as a subset selection problem; however, the equivalence of our definition and the more common definition of Set Cover as a subset problem can easily be seen by identifying columns with subsets and rows with elements to be covered.

—If M contains a column c_j without any 1-entry, then remove column c_j from M.
—If M contains a row r_i that contains exactly one 1-entry $m_{i,j}$, then remove r_i and all rows $r_{i'}$ with $m_{i',j} = 1$ from M, remove column c_j from M, and decrease k by one.
—If $k \geq 0$ and M has no rows, then answer "M is a *yes*-instance."
—If M contains a row without any 1-entry, or if $k < 0$, or if $k = 0$ and M contains at least one row, then answer "M is a *no*-instance."

For Set Cover without restrictions, these rules can be used in a preprocessing step in order to decrease the size of a problem instance before solving it. However, since there are Set Cover instances to which none of the rules applies, it is not possible to give any guarantee on the size of the problem instance resulting from the preprocessing step. If, however, the input matrix M has the C1P (for rows or for columns), then the instance can be solved by iteratively applying the rules, that is, eventually one of the rules will output "M is a *yes*-instance" or "M is a *no*-instance."

The C1P does not only help in the case of Set Cover, but even for more general problems: Red-Blue Set Cover [17] and Minimum-Degree Hypergraph [33] are generalizations of Set Cover; when restricted to instances that have the C1P, they become polynomial-time solvable [27].

4.2.2 Set Cover with almost C1P

In some applications, the arising Set Cover instances do not have the C1P, but are "close to the C1P". Motivated by problems arising from railway optimization, Mecke and Wagner [76], Ruf and Schöbel [86], and Mecke et al. [75] consider Weighted Set Cover on input matrices that have "almost C1P", which basically means that either the input matrices have been generated by starting with a matrix that has the C1P and replacing randomly a certain percentage of the 1's by 0's [76], that the average number of blocks of 1's per row is much smaller than the number of columns of the matrix [86], or that the maximum number of blocks of 1's per row is small [75]. The latter restriction was also considered by Dom and Sikdar [29] and Dom et al. [25]; here, the problem is interpreted as a a geometric covering problem called Rectangle Stabbing. Apart from heuristics performing well in practice [76, 86], the following results have been obtained.

Theorem 4.4 ([25, 42, 75, 76])**.** *1.* Set Cover *is* NP*-complete even if the input matrix M can be split into two submatrices M_1, M_2 such that $M = (M1 \mid M_2)$ and both M_1 and M_2 have the strong C1P [42, 75]. Moreover, this restricted variant of* Set Cover *is* W[1]*-complete with respect to the parameter k [25].*

2. Weighted Set Cover *restricted to input matrices with at most d blocks of 1s per row can be approximated in polynomial time with a factor d [75].*

3. Weighted Set Cover *can be solved in* $2^{\ell} \cdot \text{poly}(m, n)$ *time with* ℓ *denoting the maximum distance between the topmost and the bottommost 1 in any column of M [75, 76].*

5 Concluding Remarks

In summary, the C1P can be recognized in linear time and many in general NP-hard problems become polynomial-time solvable when restricted to inputs that have the C1P. In particular, Integer Linear Programming and several variants of Set Cover are polynomial-time solvable if the input has the C1P. In contrast, the recognition of matrices that are "close" to the C1P (for example, matrices whose columns can be permuted such that there are two blocks of 1s per row or matrices that can obtain the C1P by few column or row deletions) is typically NP-hard; moreover, NP-hard problems, such as Integer Linear Programming and Set Cover, tend to stay NP-hard on inputs being "close" to the C1P.

References

[1] Graphclasses in ISGCI. http://wwwteo.informatik.uni-rostock.de/isgci/classes.cgi.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[3] E. Althaus, S. Canzar, M. R. Emmett, A. Karrenbauer, A. G. Marshall, A. Meyer-Baese, and H. Zhang. Computing H/D-exchange speeds of single residues from data of peptic fragments. In *Proc. 23rd SAC*, pages 1273–1277. ACM Press, 2008.

[4] K. M. Anstreicher. Linear programming in $O(\frac{n^3}{\ln n}L)$ operations. *SIAM J. Optim.*, 9(4):803–812, 1999.

[5] J. E. Atkins, E. G. Boman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM J. Comput.*, 28(1):297–310, 1998.

[6] J. E. Atkins and M. Middendorf. On physical mapping and the consecutive ones property for sparse matrices. *Discrete Appl. Math.*, 71(1–3):23–40, 1996.

[7] J. Bang-Jensen, J. Huang, and A. Yeo. Convex-round and concave-round graphs. *SIAM J. Discrete Math.*, 13(2):179–193, 2000.

[8] J. J. Bartholdi, III, J. B. Orlin, and H. D. Ratliff. Cyclic scheduling via integer programs with circular ones. *Oper. Res.*, 28(5):1074–1085, 1980.

[9] S. Benzer. On the topology of the genetic fine structure. *Proc. Natl. Acad. Sci. USA*, 45:1607–1620, 1959.

[10] C. Berge. Sur certains hypergraphes généralisant les graphes bipartites. In P. Erdős, A. Rhényi, and V. T. Sós, editors, *Combinatorial Theory and its Applications I (Proceedings of the Colloquium on Combinatorial Theory and its Applications, 1969)*, pages 119–133. North-Holland, 1970.

[11] C. Berge. Balanced matrices. *Math. Program.*, 2:19–31, 1972.

[12] V. Bilò, V. Goyal, R. Ravi, and M. Singh. On the crossing spanning tree problem. In *Proc. 7th APPROX*, volume 3122 of *LNCS*, pages 51–60. Springer, 2004.

[13] 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.*, 13:335–379, 1976.

[14] I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Amer. Math. Soc.*, 55(2):299–304, 1976.

[15] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 1999.

[16] A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set covering problem. *Ann. Oper. Res.*, 98:353–371, 2000.

[17] R. D. Carr, S. Doddi, G. Konjevod, and M. V. Marathe. On the red-blue set cover problem. In *Proc. 11th SODA*, pages 345–353. ACM Press, 2000.

[18] T. Christof, M. Oswald, and G. Reinelt. Consecutive ones and a betweenness problem in computational biology. In *Proc. 6th IPCO*, volume 1412 of *LNCS*, pages 213–228. Springer, 1998.

[19] N. Christofides and J. M. P. Paixão. Algorithms for large scale set covering problems. *Ann. Oper. Res.*, 43(5):259–277, 1993.

[20] M. Conforti and G. Cornuéjols. Balanced $0, \pm 1$-matrices, bicoloring and total dual integrality. *Math. Program.*, 71:249–258, 1995.

[21] M. Conforti, G. Cornuéjols, and K. Vuskovic. Balanced matrices. *Discrete Math.*, 306(19–20):2411–2437, 2006.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[23] D. G. Corneil, S. Olariu, and L. Stewart. The ultimate interval graph recognition algorithm? (extended abstract). In *Proc. 9th SODA*, pages 175–180. ACM/SIAM, 1998.

[24] M. Dom. *Recognition, Generation, and Application of Binary Matrices with the Consecutive-Ones Property*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2008.

[25] M. Dom, M. R. Fellows, and F. A. Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. In *Proc. 3rd WALCOM*, volume 5431 of *LNCS*, pages 298–309. Springer, 2009.

[26] M. Dom, J. Guo, and R. Niedermeier. Approximability and parameterized complexity of consecutive ones submatrix problems. In *Proc. 4th TAMC*, volume 4484 of *LNCS*, pages 680–691. Springer, 2007.

[27] M. Dom, J. Guo, R. Niedermeier, and S. Wernicke. Red-blue covering problems and the consecutive ones property. *J. Discrete Algorithms*, 6(3):393–407, 2008.

[28] M. Dom and R. Niedermeier. The search for consecutive ones submatrices: Faster and more general. In *Proc. 3rd ACiD*, volume 9 of *Texts in Algorithmics*, pages 43–54. College Publications, 2007.

[29] M. Dom and S. Sikdar. The parameterized complexity of the rectangle stabbing problem and its variants. In *Proc. 2nd FAW*, volume 5059 of *LNCS*, pages 288–299. Springer, 2008.

[30] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[31] E. M. Eschen and J. Spinrad. An $O(n^2)$ algorithm for circular-arc graph recognition. In *Proc. 4rd SODA*, pages 128–137. ACM/SIAM, 1993.

[32] G. Even, R. Levi, D. Rawitz, B. Schieber, S. Shahar, and M. Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Trans. Algorithms*, 4(3), Article 34, 2008.

[33] T. Feder, R. Motwani, and A. Zhu. *k*-connected spanning subgraphs of low degree. Technical Report TR06-041, Electronic Colloquium on Computational Complexity (ECCC), 2006.

[34] U. Feige. A threshold of ln *n* for approximating Set Cover. *J. ACM*, 45(4):634–652, 1998.

[35] P. C. Fishburn. *Interval Orders and Interval Graphs*. Wiley, 1985.

[36] M. Flammini, G. Gambosi, and S. Salomone. Interval routing schemes. *Algorithmica*, 16(6):549–568, 1996.

[37] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[38] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15(3):835–855, 1965.

[39] D. R. Fulkerson, A. J. Hoffman, and R. Oppenheim. On balanced matrices. *Mathematical Programming Study*, 1:120–132, 1974.

[40] F. Gardi. The Roberts characterization of proper and unit interval graphs. *Discrete Math.*, 307(22):2906–2908, 2007.

[41] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[42] D. R. Gaur, T. Ibaraki, and R. Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *J. Algorithms*, 43(1):138–152, 2002.

[43] F. Gavril. Algorithms on circular-arc graphs. *Networks*, 4:357–369, 1974.

[44] A. Ghouila-Houri. Caractérisation des matrices totalement unimodulaires. *C. R. Acad. Sci. Paris*, 254:1192–1194, 1962.

[45] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. *J. Comput. Biol.*, 2(1):139–152, 1995.

[46] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier B. V., 2nd edition, 2004. First edition Academic Press, 1980.

[47] 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. *Theor. Comput. Sci.*, 234(1–2):59–84, 2000.

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

[49] S. Haddadi and Z. Layouni. Consecutive block minimization is 1.5-approximable. *Inf. Process. Lett.*, 108:132–135, 2008.

[50] M. Hajiaghayi. Consecutive ones property. Manuscript, School of Computer Science, University of Waterloo, Canada, 2000.

[51] M. Hajiaghayi and Y. Ganjali. A note on the consecutive ones submatrix problem. *Inf. Process. Lett.*, 83(3):163–166, 2002.

[52] G. Hajös. Über eine Art von Graphen. *Intern. Math. Nachr.*, 11, Problem 65, 1957. In German language.

[53] R. Hassin and N. Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Appl. Math.*, 30:29–42, 1991.

[54] D. S. Hochbaum and A. Levin. Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. *Discrete Optim.*, 3(4):327–340, 2006.

[55] D. S. Hochbaum and P. A. Tucker. Minimax problems with bitonic matrices. *Networks*, 40(3):113–124, 2002.

[56] A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 223–246. Princeton University Press, 1956.

[57] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[58] W.-L. Hsu. A simple test for interval graphs. In *Proc. 18th WG*, volume 657 of *LNCS*, pages 11–16. Springer, 1992.

[59] W.-L. Hsu. $O(M * N)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM J. Comput.*, 24(3):411–439, 1995.

[60] W.-L. Hsu. On physical mapping algorithms – an error-tolerant test for the consecutive ones property. In *Proc. 3rd COCOON*, volume 1276 of *LNCS*, pages 242–250. Springer, 1997.

[61] W.-L. Hsu. A simple test for the consecutive ones property. *J. Algorithms*, 43(1):1–16, 2002.

[62] W.-L. Hsu and T.-H. Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM J. Comput.*, 28(3):1004–1020, 1999.

[63] W.-L. Hsu and R. M. McConnell. PC trees and circular-ones arrangements. *Theor. Comput. Sci.*, 296(1):99–116, 2003.

[64] R. Kannan and C. L. Monma. On the computational complexity of integer programming problems. In R. Henn, B. Korte, and W. Oettli, editors, *Optimization and Operations Research*, volume 157 of *Lect. Notes Econ. Math. Syst.*, pages 161–172. Springer, 1978.

[65] H. Kaplan and Y. Nussbaum. A simpler linear-time recognition of circular-arc graphs. In *Proc. 10th SWAT*, volume 4059 of *LNCS*, pages 41–52. Springer, 2006.

[66] N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989.

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

[68] S. Kovaleva and F. C. R. Spieksma. Approximation of a geometric set covering problem. In *Proc. 12th ISAAC*, volume 2223 of *LNCS*, pages 493–501. Springer, 2001.

[69] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM J. Comput.*, 36(2):326–353, 2006.

[70] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz. SNPs problems, complexity, and algorithms. In *Proc. 9th ESA*, volume 2161 of *LNCS*, pages 182–193. Springer, 2001.

[71] C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962.

[72] W.-F. Lu and W.-L. Hsu. A test for the consecutive ones property on noisy data – application to physical mapping and sequence assembly. *J. Comput. Biol.*, 10(5):709–735, 2003.

[73] R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003.

[74] R. M. McConnell. A certifying algorithm for the consecutive-ones property. In *Proc. 15th SODA*, pages 768–777. ACM Press, 2004.

[75] S. Mecke, A. Schöbel, and D. Wagner. Station location – complexity and approximation. In *Proc. 5th ATMOS*. IBFI Dagstuhl, Germany, 2005.

[76] S. Mecke and D. Wagner. Solving geometric covering problems by data reduction. In *Proc. 12th ESA*, volume 3221 of *LNCS*, pages 760–771. Springer, 2004.

[77] J. Meidanis, O. Porto, and G. P. Telles. On the consecutive ones property. *Discrete Appl. Math.*, 88:325–354, 1998.

[78] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Discrete Mathematics and Optimization. Wiley, 1988.

[79] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[80] M. Oswald and G. Reinelt. Polyhedral aspects of the consecutive ones problem. In *Proc. 6th COCOON*, volume 1858 of *LNCS*, pages 373–382. Springer, 2000.

[81] M. Oswald and G. Reinelt. Constructing new facets of the consecutive ones polytope. In *Proc. 5th Intern. Workshop on Combinatorial Optimization—"Eureka, You Shrink!"*, volume 2570 of *LNCS*, pages 147–157, 2003.

[82] M. Oswald and G. Reinelt. The weighted consecutive ones problem for a fixed number of rows or columns. *Oper. Res. Lett.*, 31(3):350–356, 2003.

[83] M. Oswald and G. Reinelt. The simultaneous consecutive ones problem. *Theor. Comput. Sci.*, 410(21–23):1986–1992, 2009.

[84] F. A. Potra and S. J. Wright. Interior-point methods. *J. Comput. Appl. Math.*, 124:281–302, 2000.

[85] F. S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, 1969.

[86] N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. *Discrete Optim.*, 1(2):215–228, 2004.

[87] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

[88] J. Tan and L. Zhang. The consecutive ones submatrix problem for sparse matrices. *Algorithmica*, 48(3):287–299, 2007.

[89] K. Truemper. On balanced matrices and Tutte's characterization of regular matroids. Manuscript, 1978.

[90] A. C. Tucker. Matrix characterizations of circular-arc graphs. *Pacific J. Math.*, 2(39):535–545, 1971.

[91] A. C. Tucker. A structure theorem for the consecutive 1's property. *J. Combin. Theory Ser. B*, 12:153–162, 1972.

[92] A. C. Tucker. An efficient test for circular-arc graphs. *SIAM J. Comput.*, 9(1):1–24, 1980.

[93] A. F. Veinott and H. M. Wagner. Optimal capacity scheduling. *Oper. Res.*, 10:518–547, 1962.

[94] M. Veldhorst. Approximation of the consecutive ones matrix augmentation problem. *SIAM J. Comput.*, 14(3):709–729, 1985.

[95] J. von zur Gathen and M. Sieveking. A bound on solutions of linear integer equations and inequalities. *Proc. Amer. Math. Soc.*, 72:155–158, 1978.

[96] R. Wang, F. C. M. Lau, and Y. Zhao. Hamiltonicity of regular graphs and blocks of consecutive ones in symmetric matrices. *Discrete Appl. Math.*, 155(17):2312–2320, 2007.

[97] S. Weis and R. Reischuk. The complexity of physical mapping with strict chimerism. In *Proc. 6th COCOON*, volume 1858 of *LNCS*, pages 383–395. Springer, 2000.