
Acta Univ. Sapientiae Informatica **15,** 2 (2023) 267–293

DOI: 10.2478/ausi-2023-0017

Methods for the graph realization problem

Zoltán KÁSA
Sapientia Hungarian University of Transylvania, Cluj-Napoca, Romania
Dept. of Mathematics and Informatics
email: kasa@ms.sapientia.ro
ORCID:0000-0002-2697-1599

Pál A. KUPÁN
Sapientia Hungarian University of Transylvania, Cluj-Napoca, Romania
Dept. of Mathematics and Informatics
email: kupanp@ms.sapientia.ro
ORCID:0000-0002-9290-3121

Csaba György PĂTCAȘ
Babeș–Bolyai University, Cluj-Napoca, Romania
Faculty of Mathematics and Informatics
email: csaba.patcas@ubbcluj.ro
ORCID:0009-0001-6485-8765

Abstract. The graph realization problem seeks an answer to how and under what conditions a graph can be constructed if we know the degrees of its vertices. The problem was widely studied by many authors and in many ways, but there are still new ideas and solutions. In this sense, the paper presents the known necessary and sufficient conditions for realization with the description in pseudocode of the corresponding algorithms. Two cases to solve the realization problem are treated: finding one solution, and finding all solutions. In this latter case a parallel approach is presented too, and how to exclude isomorphic graphs from solutions. We are also discussing algorithms using binary integer programming and flow networks.

In the case of a bigraphical list with equal out- and in-degree sequences a modified Edmonds–Karp algorithm is presented such that the resulting graph will be always symmetric without containing loops. This algorithm solves the problem of graph realization in the case of undirected graphs using flow networks.

Key words and phrases: degree sequences, graph realization algorithms, flow networks, graphical lists, bigraphical lists, modified Edmonds–Karp algorithm

1 Introduction

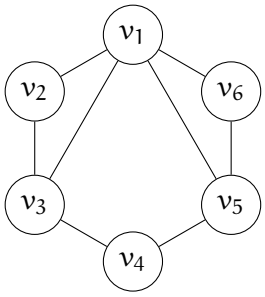
A graph realization or graph construction problem asks if for a given finite sequence $(d_1, d_2, \ldots, d_n)$ of natural numbers there exists a finite simple graph such that $d_1, d_2, \ldots, d_n$ represent the degrees of vertices of this graph. The problem has been widely studied mostly from a theoretical point of view, giving necessary and sufficient conditions for the existence of the solution [8, 7, 5, 9]. The problem can of course also be formulated for directed graphs, if we give two sets of natural numbers for the in-degrees and out-degrees [13, 11, 12].

Two cases to solve the realization problem can arise:

- finding a graph which satisfies the conditions,
- finding all graphs which satisfy the conditions.

A sequence of non-negative integers is called *graphical* if it is the degree sequence of some graph. A list $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ of pair of non-negative integers is *bigraphical*, if a_i are the out-degrees, b_i the in-degrees of the vertices of some directed graph.

For example: 4, 3, 3, 2, 2, 2 is a graphical sequence. For the corresponding graph and adjacency matrix see Fig. 1.

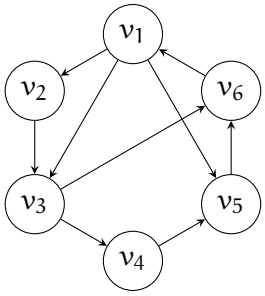


$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 1: Example of a graph with the degrees $4, 3, 3, 2, 2, 2$ and its adjacency matrix

In this article, we will use the following algorithms:

- Finding a solution
  - algorithm based on the Havel–Hakimi theorem for undirected graphs,
  - algorithm based on the Kleitman–Wang theorem for directed graphs,
  - algorithm using flow network for directed graph,
  - using binary integer programming algorithm.
- Finding all solutions
  - by parallel testing the solutions.



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

Figure 2: Example of a digraph with the out-degrees $3,1,2,1,1,1$ and in-degrees $1,1,2,1,2,2$, so with the bigraphical list $(3,1),(1,1),(2,2),(1,1),(1,2),(1,2)$, and its adjacency matrix

2 Necessary and sufficient conditions

The first characterization of graphic sequences, an algorithmic one, was published by Havel [8] in 1955, completed by Hakami in 1962 [7]. Erdős and Gallai gave a completely different type of characterization in 1960 [5]. In 2008 Tripathi and Tyagi presented two new characterizations [19].

The running time of these algorithms is $\Omega(n^2)$ in worst case. Iványi et al. [9] have proposed a faster algorithm called EGL (Erdős-Gallai Linear algorithm), whose worst running time is $\Theta(n)$. Other characterizations can be found in [18] (1994), [1] (1997), [14] (2004), [3], and [20] (2010). In [2] and [6] related problems are discussed.

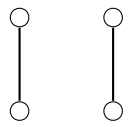
We will present here the first three characterizations which provide a necessary and sufficient condition for a sequence of natural numbers to be graphical.

Theorem 1 (Havel–Hakimi) [8, 7] *A sequence* $d_1, d_2, \ldots, d_n$ *of non-negative integers, with* $d_1 \geq d_2 \geq \cdots \geq d_n$*, where* $n \geq 2$*,* $d_1 \geq 1$*, is graphical if and only if the sequence*

$$d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3}, \ldots, d_n$$

is graphical too.

Example:
4 3 3 2 2 2
4 $\boxed{3\ 3\ 2\ 2}$ 2
2 2 1 1 2
2 $\boxed{2\ 2}$ 1 1
1 1 1 1 which is graphical representing the graph:



Because the last sequence is graphical, as is illustrated in the attached figure, the first one is graphical too.

For the next theorem let us consider a sequence of non-negative integers $d_1 \geq \cdots \geq d_n$, and let us denote:

$$H_i = \sum_{k=1}^{i} d_k, \quad K_i = \sum_{k=i+1}^{n} \min(d_k, i).$$

Theorem 2 (Erdős–Gallai) [5] *A sequence of non-negative integers* $d_1 \geq \cdots \geq d_n$ *is graphical if and only if*

- H_n *is even and*
- $H_i \leq i(i-1) + K_i$ *holds for every* i, $1 \leq i \leq n-1$.

Example. The sequence 3, 3, 3, 1 is not graphical, because for $i = 2$ we have $3 + 3 > 2(2-1) + 2 + 1$.

Theorem 4 (to be discussed later) is a better applicable variant of the presented Erdős–Gallai theorem.

The following theorem applies to directed graphs.

Theorem 3 (Kleitman–Wang) [13] *Let*

$$(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$$

be a list of pairs of non-negative integers in non-increasing lexicographic order and a pair (a_i, b_i) *with* $b_i > 0$. *The above list is bigraphical if and only if the list obtained by the following rules is bigraphical too.*

1. *Change* (a_i, b_i) *to* $(a_i, 0)$.
2. *Let's note by* (a_k, b_k) *each of the first* b_i *pairs from the beginning of the sorted list such that* $i \neq k$. *Change all to* $(a_k - 1, b_k)$.
3. *Leave the remaining pairs as they were.*

Example. $(3,1)$, $(2,2)$, $(2,2)$, $(1,3)$ is bigraphical because:

$(3,1)$, $(2,2)$, $(2,2)$, $\boxed{(1,3)}$
$(2,1)$, $(1,2)$, $(1,2)$, $(1,0)$
$(2,1)$, $(1,2)$, $\boxed{(1,2)}$, $(1,0)$
$(1,1)$, $(0,2)$, $(1,0)$, $(1,0)$

$(1,1)$, $(1,0)$, $(1,0)$, $\boxed{(0,2)}$
$(0,1)$, $(0,0)$, $(1,0)$, $(0,0)$
$\boxed{(1,0)}$, $(0,1)$, $(0,0)$, $(0,0)$,
$(0,0)$, $(0,0)$, $(0,0)$, $(0,0)$, which is obviously bigraphical.

Number of degree sequences Antal Iványi et al. [9] have counted the n-element graphical sequences for $n \leq 32$ with a parallel approach (server and client programs) using 350 university laboratory computers operated continuously for two summer months.

3 Algorithms

To see how the question of the graph realization problem arises, let

$$d = (4,3,3,2,2,2)$$

be a graphical sequence whose length is $n = 6$, the number of vertices of the graph. A solution graph and the corresponding adjacency matrix appear in Fig. 1.

3.1 Testing possible solutions

To solve the problem of obtaining a graph with the given degree sequence we start from the symmetric adjacency matrix

$$A = \begin{pmatrix} 0 & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\ x_{12} & 0 & x_{23} & x_{24} & x_{25} & x_{26} \\ x_{13} & x_{23} & 0 & x_{34} & x_{35} & x_{36} \\ x_{14} & x_{24} & x_{34} & 0 & x_{45} & x_{46} \\ x_{15} & x_{25} & x_{35} & x_{45} & 0 & x_{56} \\ x_{16} & x_{26} & x_{36} & x_{46} & x_{56} & 0 \end{pmatrix},$$

where x_{ij}, $i = 1, \ldots, n-1, i < j$ denotes the edge between the nodes i and j, i.e. $x_{ij} = 1$ if there exists an edge, and $x_{ij} = 0$ otherwise. The $x_{ij} \in \{0,1\}$ meet

the following conditions:

$$\begin{cases} x_{12}+x_{13}+x_{14}+x_{15}+x_{16} &= 4\\ x_{12}+x_{23}+x_{24}+x_{25}+x_{26} &= 3\\ x_{13}+x_{23}+x_{34}+x_{35}+x_{36} &= 3\\ x_{14}+x_{24}+x_{34}+x_{45}+x_{46} &= 2\\ x_{15}+x_{25}+x_{35}+x_{45}+x_{56} &= 2\\ x_{16}+x_{26}+x_{36}+x_{46}+x_{56} &= 2 \end{cases} \tag{1}$$

Each solution of this system of equations is a solution of the graph realization problem.
Reordering the equations in (1) we obtain the following system of linear equations:

$$\begin{cases} x_{12}+x_{13}+x_{14}+x_{15}+x_{16} & & & & & & & = 4\\ x_{12}+ & & x_{23}+x_{24}+x_{25}+x_{26} & & & & & = 3\\ & & & & & & & = 3\\ & & & & & \ddots & & = 2\\ & & & & & & & = 2\\ & & x_{16}+ & & x_{26}+ & & x_{36}+x_{46}+x_{56} & = 2 \end{cases} \tag{2}$$

with $x_{ij} \in \{0,1\}$, $i<j$,
or

$$Bx = d, \tag{3}$$

where $x=(x_{ij}) \in \{0,1\}^N$, $i<j$, and $B \in \{0,1\}^{n\times N}$, $N=\frac{n(n-1)}{2}, n>2$,

$$B = \begin{pmatrix} 1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\ 0&1&0&0&0&1&0&0&0&1&1&1&0&0&0\\ 0&0&1&0&0&0&1&0&0&1&0&0&1&1&0\\ 0&0&0&1&0&0&0&1&0&0&1&0&1&0&1\\ 0&0&0&0&1&0&0&0&1&0&0&1&0&1&1 \end{pmatrix}.$$

As we see in each column there are two 1's, and in each row there are $(n-1)$ 1's. The matrix B can be decomposed as

$$B = \begin{pmatrix} E_5 & 0 & & 0 & 0 \\ & E_4 & 0 & \ddots & 0 \\ I_5 & & E_3 & 0 & \\ & I_4 & & E_2 & 0 \\ & & I_3 & I_2 & E_1 \\ & & & & I_1 \end{pmatrix}, \tag{4}$$

or in the general case

$$B = \begin{pmatrix} E_{n-1} & 0 & & 0 & 0 \\ & E_{n-2} & & \ddots & 0 \\ I_{n-1} & & & 0 & \\ & I_{n-2} & \ddots & E_2 & 0 \\ & & & I_2 & E_1 \\ & & & & I_1 \end{pmatrix}, \tag{5}$$

where $E_k = \underbrace{(\ 1\ \ 1\ \ \dots\ \ 1\ )}_{k}$, and I_k is the identity matrix of order k.

Because $\triangle$, the minor formed with the first $(n-1)$ and the last column, differs from zero:

$$\det \triangle = \det \begin{pmatrix} E_{n-1} & 0 \\ I_{n-1} & \vdots \\ & E_1 \\ & I_1 \end{pmatrix} = 2 \cdot (-1)^{n-1},$$

it follows that the matrix B is of full rank: $\operatorname{rank}(B) = n$. For $n > 3$ the system is underdetermined, so the existence of a solution in $\{0,1\}^N$ depends on the fulfillment of the Havel-Hakimi theorem.

From (2) it follows that

$$\sum_{i<j} x_{ij} = \frac{1}{2} \sum_{k=1}^{n} d_k = m,$$

where m is the number of the edges. So, the number of 1's in each solution is m.

A solution of the system (3) can be obtained by testing binary sequences. From all 2^N, N length binary vectors, we need to test "only" $\binom{N}{m}$ vectors. In our example there are $\binom{15}{8} = 6435$ possibilities from which 27 are solutions (see Table 1). But from these solutions only 4 are not isomorphic. The isomorphic classes each containing respectively 12, 6, 6, and 3 isomorphic graphs.

One way to test a sequences is to calculate the scalar product of the binary sequence and each row of the matrix B. If all this equals the components of the $d = \begin{pmatrix} d_1 & d_2 & \dots & d_n \end{pmatrix}$ vector, then the binary sequence is a solution. But we can avoid the numerous zero operations in the dot product if we use the special shape of the sparse matrix B. We need only to add the components of the binary sequence corresponding to the 1's from the rows of the matrix B.

x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{23}	x_{24}	x_{25}	x_{26}	x_{34}	x_{35}	x_{36}	x_{45}	x_{46}	x_{56}
1	1	1	1	0	1	1	0	0	0	0	1	0	0	1
1	1	1	1	0	1	0	1	0	0	0	1	0	1	0
1	1	1	1	0	1	0	0	1	1	0	0	0	0	1
1	1	1	1	0	1	0	0	1	0	1	0	0	1	0
1	1	1	1	0	1	0	0	1	0	0	1	1	0	0
1	1	1	1	0	0	1	0	1	0	1	1	0	0	0
1	1	1	1	0	0	0	1	1	1	0	1	0	0	0
1	1	1	0	1	1	1	0	0	0	1	0	0	0	1
1	1	1	0	1	1	0	1	0	1	0	0	0	0	1
1	1	1	0	1	1	0	1	0	0	1	0	0	1	0
1	1	1	0	1	1	0	1	0	0	0	1	1	0	0
1	1	1	0	1	1	0	0	1	0	1	0	1	0	0
1	1	1	0	1	0	1	1	0	0	1	1	0	0	0
1	1	1	0	1	0	0	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	0	0	1	0	0	0	0	1
1	1	0	1	1	1	1	0	0	0	1	0	0	1	0
1	1	0	1	1	1	1	0	0	0	0	1	1	0	0
1	1	0	1	1	1	0	1	0	1	0	0	0	1	0
1	1	0	1	1	1	0	0	1	1	0	0	1	0	0
1	1	0	1	1	0	1	1	0	1	0	1	0	0	0
1	1	0	1	1	0	1	0	1	1	1	0	0	0	0
1	0	1	1	1	1	1	0	0	0	1	1	0	0	0
1	0	1	1	1	1	0	1	0	1	0	1	0	0	0
1	0	1	1	1	1	0	0	1	1	1	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	1	0	0	0
0	1	1	1	1	1	1	0	1	0	1	0	0	0	0
0	1	1	1	1	1	0	1	1	1	0	0	0	0	0

Table 1: 27 solutions from 6435 possible sequences

Parallel approach. When we generate the $\binom{N}{m}$ possibilities in fact the positions of the elements equal to 1 are generated. Grouping into a set the possible solutions with the same starting position, these sets can be tested in parallel, which greatly reduces the testing time.

Isomorphism. If A and B are adjacency matrices of two isomorphic graphs, then there is a permutation matrix P (each row and each column contains exactly one 1, the other elements are 0) such that $B = PAP^{-1}$.

Verifying isomorphism. We generate all permutations of $12\cdots n$. Each permutation yields a permutation matrix $P = (p_{ij})$ as follows: if $s_1 s_2 \cdots s_n$ is a permutation of $12\cdots n$, then $p_{is_i} = 1$ for $i = 1, 2, \ldots, n$. For example, the following matrix corresponds to permutation 13425:

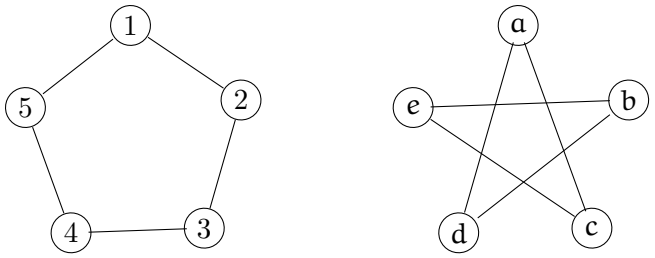
$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrix PAP^{-1} can be obtained without multiplying the matrices, only by simply swapping the corresponding rows and columns as follows:

```
for i =1,2,...,n
  for j =1,2,...,n
    if p_ij = 1 then
      swap row i and row j in A
      swap column i and column j in A
```

and the obtained matrix will be B, if the graphs represented by the adjacency matrices A and B are isomorphic.

For example the following graphs are isomorphic:



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \quad P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Based on matrix P the correspondence of the vertices are: $1 \to a$, $2 \to d$, $3 \to b$, $4 \to e$, $5 \to c$.

Parallel approach. As the permutations can be generated in parallel based on the first position, there will be n groups each of $(n-1)!$ permutations. This allows n computers to work in parallel. Thereby the execution time can be reduced from $cn!$ to $c(n-1)!$ This can be continued depending on the number of computers.

3.2 Determining a solution with binary integer programming algorithm

Integer linear programming ([17], [21]) techniques can be also used to solve the graph realization problem.

Problem (3) can be regarded as an integer linear programming problem of the form:

$$\max_x c^t x$$
$$\begin{cases} Bx \leq d \\ x_{ij} \in \{0,1\} \quad i = 1,\ldots,n-1, i < j \end{cases}.$$

Although, the original problem does not contain an objective function, this can be used to obtain different solutions. The components of the sequence c will be set to $0-1$, so they work as weights. It is worth noting that the 1's in the solutions are concentrated at the beginning (see Table (1)). This is due to the fact that the sequence d is non-increasing. In this light, let us set all components of c to zero. In this case we obtain the first solution from the Table 1:

$$sol_1 = (1\,1\,1\,1\,0\,1\,1\,0\,0\,0\,0\,1\,0\,0\,1).$$

If we set the last m components of the vector c to 1:

$$c = (0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1),$$

we obtain a solution with the most possible 1's in the last m positions (in our example there are only two such solutions):

$$sol_7 = (1\,1\,1\,1\,0\,0\,0\,1\,1\,1\,0\,1\,0\,0\,0).$$

Naturally, different sequences c does not necessarily mean different solutions.

The algorithm uses a branch-and-bound method to divide the problem into a few smaller ones, and a relaxation technique to obtain an optimal integer (binary) solution of the problem. The complexity of the algorithm is polynomial.

Algorithm 1: Linear Erdős–Gallai algorithm

Input: Sequence $d_1 \geq d_2 \geq \cdots \geq d_n > 0$, $n \geq 2$
Output: TRUE if the input sequence is graphical, and FALSE otherwise.

```
H_0 = 0
for i = 1 to n do H_i = H_{i-1} + d_i
if H_n is odd then return FALSE
d_0 = n - 1
for i = 1 to n do
    if d_i < d_{i-1} then
        for j = d_{i-1} downto d_i + 1 do w_j = i - 1
        w_{d_i} = i
    end
end
for j = d_n downto 1 do w_j = n
for i = 1 to n do
    if i ≤ w_i then
        if H_i > i(i-1) + i(w_i - i) + H_n - H_{w_i} then
            return FALSE
        end
    end
    if i > w_i then
        if H_i > i(i-1) + H_n - H_i then return FALSE
    end
end
return TRUE
```

3.3 Testing the graphical sequence property based on Erdős–Gallai theorem

Based on the Erdős–Gallai theorem in [9] a linear time algorithm is presented to check if a sequence is graphical or not. This algorithm follows directly from the next theorem proved in [9].

We need the following: for given sequence $d_1 \geq d_2 \geq \cdots \geq d_n > 0$ let $w = (w_1, \ldots, w_{n-1})$, where w_i gives the index of d_k having the maximal index among such elements of the sequence which are greater or equal to i.

Recall that $H_i = \sum_{k=1}^{i} d_k$.

Theorem 4 [9] *If* $n \geq 1$, *then the sequence* $d_1 \geq d_2 \geq \cdots \geq d_n > 0$, *is graphical if and only if*

$$H_n \quad \textit{is even}$$

and if $i > w_i$, *then*

$$H_i \leq i(i-1) + H_n - H_i,$$

further if $i \leq w_i$, *then*

$$H_i \leq i(i-1) + i(w_i - i) + H_n - H_{w_i}.$$

Algorithm 1 is based on the one described in [9], which will be prerequisite for the Havel–Hakimi graph realization algorithms. It is easy to see that it has a linear time complexity.

3.4 Graph realization problem using the Havel–Hakimi theorem

It is possible to check whether Theorem 1 is satisfied using an $O(n^2 \log n)$ time complexity algorithm by sorting the list after every step. To avoid this we can observe that after decreasing some d_1 values from the list at each step, the values can be sorted by swapping two contiguous subsequences of the list. This can be done in linear time, thus our algorithm has $O(n^2)$ time complexity. A less efficient variant with the same time complexity would be to merge the subsequence containing the decreased elements with the remainder of the list.

The sequence $z_1, z_2, \ldots, z_n$ keeps the original positions of vertices during the algorithm.

Algorithm 2 solves the problem using the Havel–Hakimi theorem.

Example. Using Algorithm 2 for the graphical sequence 4, 3, 3, 2, 2 the solution is given in Fig. 3.

The input of this algorithm must be a graphical degree sequence. A degree sequence can be checked out for this by Algorithm 1. But Algorithm 2 easily can be modified to check also that the input is graphical or not.

3.5 Graph realization problem using the Kleitman–Wang theorem

Let us denote for a vertex v_i in a digraph by a_i its out-degree, and by b_i its in-degree. The list $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ of pairs of non-negative integers is a degree sequence of some digraph. In an unusual way, here the

Algorithm 2: Graph realization using the Havel–Hakimi theorem

Input: Graphical sequence $d_1 \geq d_2 \geq \cdots \geq d_n > 0$, $n \geq 2$
Output: Adjacency matrix $A = (a_{ij})$ $(i, j = 1, 2, \ldots n)$ of the solution graph

for $i := 1$ **to** n **do**
  $z_i := i$
  for $j := 1$ **to** n **do** $a_{ij} := 0$
end
$k := 1$; $m := n$
while $(m > k)$ **do**
  $c := d_k$
  $s := -1$
  for $i := k + 1$ **to** $k + c$ **do**
    $d_i := d_i - 1$
    $a_{z_k, z_i} := 1$
    $a_{z_i, z_k} := 1$
    if $s = -1$ ***and*** $k + c < n$ ***and*** $d_i < d_{k+c+1}$ **then** $s := i$
  end
  if $s > 0$ **then**
    $i := s$
    $j := k + c + 1$
    while $j \leq n$ ***and*** $d_i < d_j$ **do** $j := j + 1$
    $l := k + c + 1 - s$
    $r := j - k - c - 1$
    if $l > r$ **then** $j := k + c + 1$
    else $j := j - l$
    while $i \leq k + c$ ***and*** $j \leq n$ ***and*** $d_i < d_j$ **do**
      Swap d_i and d_j, respectively z_i and z_j
      $i := i + 1$
      $j := j + 1$
    end
  end
  while $(d_m = 0)$ ***and*** $(m > k)$ **do** $m := m - 1$
  $k := k + 1$
end

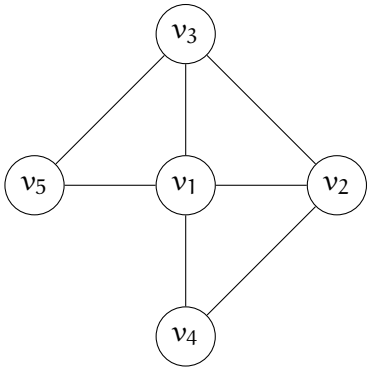


Figure 3: A solution to the graphical sequence 4, 3, 3, 2, 2

first number means the out-degree, the second the in-degree. We will see the benefits of this later.

We recall that a list $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ of pairs of nonnegative integers is called *bigraphical* if it is the degree sequence of some digraph.

We denote by $(a_1, b_1) \succeq (a_2, b_2) \succeq, \ldots, \succeq (a_n, b_n)$ a list $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ which is in non-increasing lexicographic order. Here, we use a similar idea as in Algorithm 2, sequence $z_1, z_2, \ldots, z_n$ keeps the original positions of vertices during the algorithm, and $y_1, y_2, \ldots, y_n$ denotes the inverse permutation of z, that is y_i stores the current position where the pair originally at position i can be found in the list. Because the sorting order now depends on two parameters, both values of each pair, it is not possible anymore to simply exchange two subsequences, we need the classical merging algorithm this time. We also use a queue (first in first out) called q.

This algorithm works not only for bigraphical sequences, the opposite is indicated by an error message.

Example. Using Algorithm 3 for the bigraphical sequence $(2,0)$, $(1,1)$, $(1,0)$, $(0,3)$ the solution is given in Fig. 4. In the first step of the **while** cycle from
$h = 2$ $(2,0)$, $(1,1)$, $(1,0)$, $(0,3)$, we obtain $(1,0)$, $(1,0)$, $(1,0)$, $(0,3)$,
In the next step from
$h = 4$ $(1,0)$, $(1,0)$, $(1,0)$, $(0,3)$, we obtain $(0,0)$, $(0,0)$, $(0,0)$, $(0,0)$.

3.6 Graph realization problem using flow networks

This method can be applied for directed graphs (see [16]), and for undirected graphs with some modifications. A bigraphical list (a_1, b_1), (a_2, b_2),

Algorithm 3: Digraph realization by the Kleitman–Wang theorem

Input: Bigraphical sequence $(a_1, b_1) \succeq (a_2, b_2) \succeq, \ldots, \succeq (a_n, b_n)$
Output: Adjacency matrix $X = (x_{ij})$ $(i, j = 1, 2, \ldots n)$ of the solution graph

```
for i := 1 to n do
    z_i := i
    y_i := i
    for j := 1 to n do x_ij := 0
    if b_i > 0 then  Push i to the end of q
end
while q is not empty do
    Pop the first element of q into h
    c := b_{y_h}
    if c > n − 1 then
        return Error: The sequence is not bigraphical.
    end
    s := −1
    i := 1
    while i ≤ c do
        if z_i ≠ h then
            if a_i ≤ 0 then
                return Error: The sequence is not bigraphical.
            else
                a_i := a_i − 1
                x_{z_i,h} := 1
            end
            if s = −1 and pair c + 1 ≻ pair i in the list then s := i
        else c := c + 1
        i := i + 1
    end
    if s > 0 then
        Merge a_{s...c} with a_{c+1...n} updating z and y accordingly
    end
    k := y_h
    b_k := 0
    Move pair k to the right in the list if necessary updating z and y
      accordingly
end
if a_1 > 0 then
    return Error: The sequence is not bigraphical.
end
```

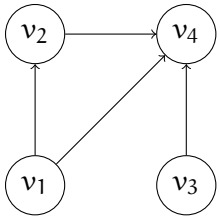


Figure 4: A solution to the bigraphical sequence $(2,0)$, $(1,1)$, $(1,0)$, $(0,3)$

$\ldots$, (a_n, b_n) is given, where as we have already seen $a_1, a_2, \ldots, a_n$ represent the out-degree, and $b_1, b_2, \ldots, b_n$ the in-degree sequences. To find a graph with these out- and in-degrees we will use a flow network of $2n+2$ nodes. Denote the source node by v, the sink node by w, and the internal nodes by $v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n$. The arcs are the following:

$$\begin{array}{lll} (v, v_i) & \text{for} & i = 1, 2, \ldots, n, \\ (v_i, w_j) & \text{for} & i = 1, 2, \ldots, n, j = 1, 2, \ldots, n, \\ & & \text{and} \quad i \neq j, \\ (v_j, w) & \text{for} & j = 1, 2, \ldots, n. \end{array} \tag{6}$$

The capacities are:

$$\begin{array}{lll} c(v, v_i) = a_i & \text{for} & i = 1, 2, \ldots, n, \\ c(v_i, v_j) = 1 & \text{for} & i = 1, 2, \ldots, n, j = 1, 2, \ldots, n, \\ & & \text{and } i \neq j, \\ c(v_j, w) = b_j & \text{for} & j = 1, 2, \ldots, n. \end{array} \tag{7}$$

For an example see Fig. 5.

A maximum flow in the above defined flow network can be obtained for example by the Edmonds–Karp algorithm [4]. A maximal flow in this network which saturates all arcs from v, and all arcs to w, will give us a solution to the realization problem. The arcs between the interior vertices with flow equal to 1 yield the edges of the resulting graph. A saturated arc (v_i, w_j) where $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$, and $i \neq j$, gives an arc (v_i, w_j) of the solution. Obviously, the solution is not unique, from any maximum flow with the above conditions, a new graph results.

Algorithm 4 constructs a graph from a bigraphical list using flow networks.

Algorithm 4: Graph realization using flow network

Input: Bigraphical list (a_1, b_1), (a_2, b_2), $\ldots$, (a_n, b_n)

Output: Adjacency matrix $A = (a_{ij})$ $(i, j = 1, 2, \ldots, n)$ of the solution graph

Construct the graph: $G = (V, E)$, where $V = \{v, w, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n, \}$ and the arcs in E are given by (6) with capacities by (7)

Apply to G the Edmonds–Karp algorithm obtaining a maximum flow f

```
for i := 1 to n do
    for j := 1 to n do
        if i = j then a_ii = 0
        else a_i,j = f(v_i, w_j)
    end
end
```

The complexity of the Algorithm 4 is $O(n^5)$ because of the complexity of the Edmonds-Karp algorithm, but it can be improved to $O(n^3)$ by using the algorithm from [15].

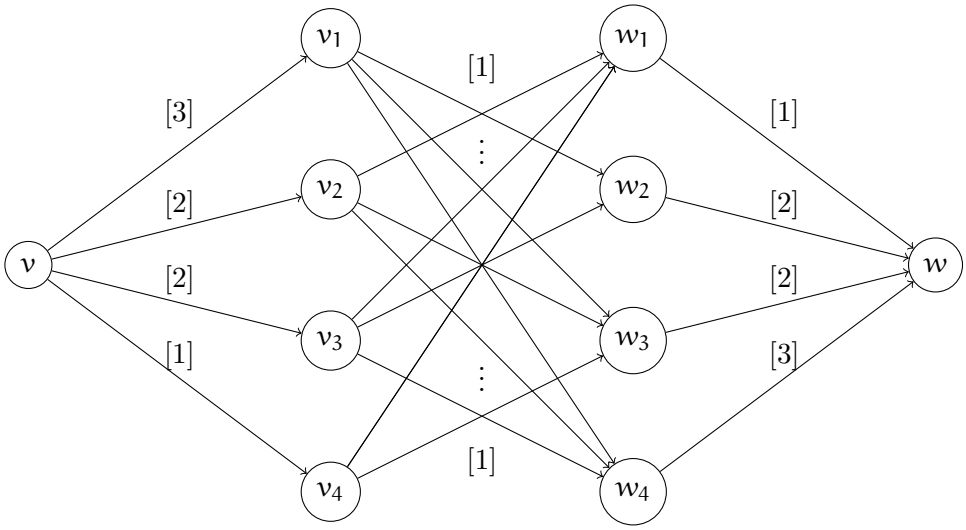


Figure 5: An example of flow network attached to the bigraphical list $(3, 1)$, $(2, 2)$, $(2, 2)$ $(1, 3)$

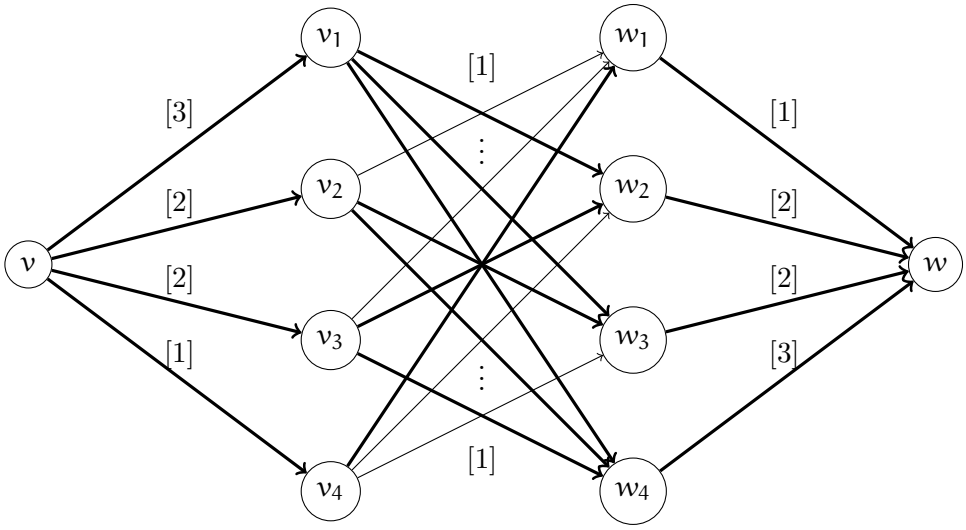


Figure 6: A solution of the example in Fig. 5, the saturated arcs are marked thick. The other arcs have flow equal to 0.

Example. Starting from the bigraphical list $(3,1)$, $(2,2)$, $(2,2)$, $(1,3)$ we obtain the flow network in Fig. 5. By the well-known Edmonds–Karp algorithm we find the maximum flow (see Fig. 6, where the thick arrows are saturated) which corresponds to the following solution, where $f(i,j)$ is the flow on the arc from vertex v_i to vertex w_j:

$f(i,j)$	w_1	w_2	w_3	w_4
v_1	0	1	1	1
v_2	0	0	1	1
v_3	0	1	0	1
v_4	1	0	0	0

and which is the adjacency matrix of the resulting graph (Fig. 7.a).

In the case of a bigraphical list with the equal out- and in-degree sequences (i.e. $a_i = b_i$, for $i = 1,\dots,n$) if the resulting graph is symmetric, then the solution can be considered as a solution for the undirected case (where $a_1, a_2, \dots, a_n$ is a graphical sequence). Such a case can be viewed in Fig. 7b. But Algorithm 4 does not always give a symmetric graph as solution. As an example consider the case of the degree sequence $(2,2,2,2,2)$. See Fig. 8a for the resulting graph. In Fig. 9 on the left we can see the maximum flow obtained. Using the following alternating semipath (which is a closed one)

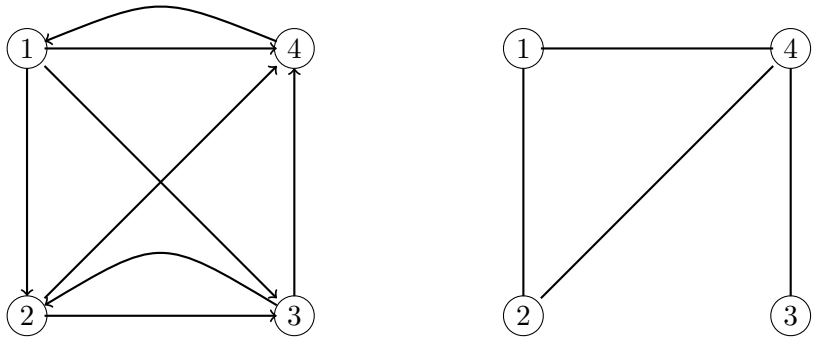
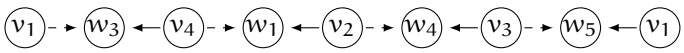


Figure 7: Solution for the bigraphical list: a) $(3,1)$, $(2,2)$, $(2,2)$, $(1,3)$, b) $(2,2)$, $(2,2)$, $(1,1)$, $(3,3)$ which corresponds to the graphical sequence $2,2,1,3$.



and changing in the maximum flow all dashed arcs from here with a thick one next to it, a new maximum flow will arise (on the right in Fig. 9). This solution yields a symmetric graph (Fig. 8.b), and the corresponding undirected graph is in Fig. 8.c. In the Appendix an algorithm for finding all closed alternating semipaths is given.

A different approach to get from a solution to another one, is based on the so called *square change* [5]. If we delete two arcs $(a,b),(c,d)$, then add the two arcs $(a,d),(c,b)$ the degree sequence does not change. In the case of Fig8*a)* the following square changes $(2,4),(3,5) \to (2,5),(3,4)$; $(1,3),(4,1) \to (1,1),(4,3)$; $(2,5),(1,1) \to (2,1),(1,5)$ will give the solution in Fig.8*b)*.

A modified Edmonds–Karp algorithm for undirected graphs. In the case of a bigraphical list with equal out- and in-degree sequences (i.e. $a_i = b_i$, for $i = 1,\ldots,n$) it is possible to modify the Edmonds–Karp algorithm in such a way that the resulting graph will be always symmetric without containing loops. In order to achieve this, we need to modify the breadth-first search by allowing only paths whose symmetric is also an augmenting path. Then each time we increase the flow along both paths.

As usual, given the edge capacities and flow values of the network, for every arc (x,y) in the original network by definition the residual network has an arc

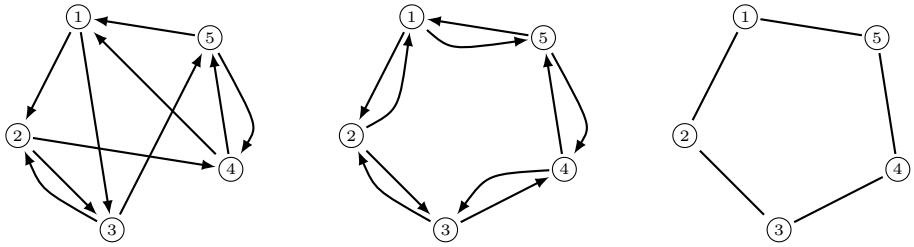


Figure 8: a) Solution for the bigraphical list $(2,2),(2,2),(2,2),(2,2),(2,2)$ given by the Algorithm 4. b) Solution after applying the alternating semipath method. c) In the solution of b) each pair of arcs (u,v) and (v,u) has been substituted by the edge $\{u,v\}$

.

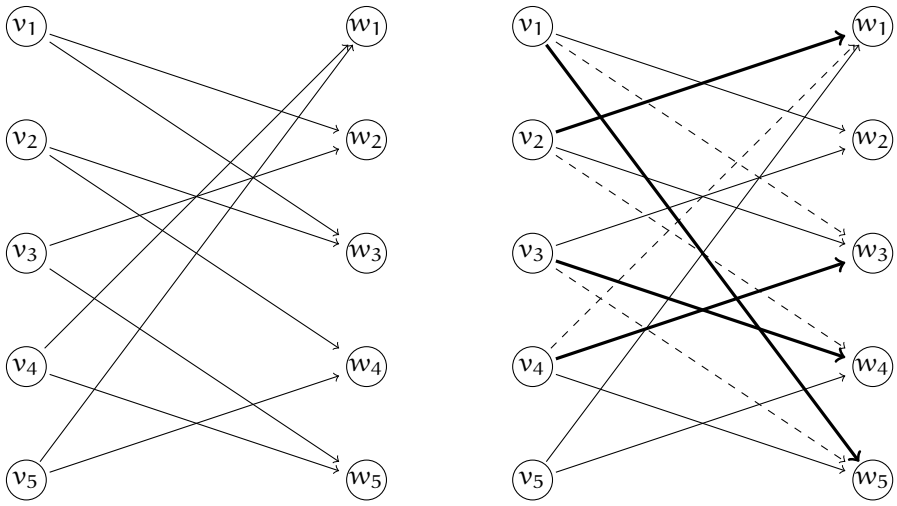


Figure 9: Using the alternating semipath method to obtain a new maximum flow which yields a symmetric graph as solution. Only the arcs which have flow equal to 1 are drawn. Eliminating the dashed arcs and introducing the thick ones, a new maximum flow is obtained.

(x,y) if $c(x,y) - f(x,y)$ is positive and an arc (y,x) if $f(x,y)$ is positive. In the residual network we set the capacities to the respective values.

Algorithm 5: Graph realization for the symmetric case

Input: Flow network (as described in (6) and (7))
Output: Adjacency matrix $A = (a_{ij})$ $(i, j = 1, 2, \ldots, n)$ of the solution graph
while BFS*(c, f, p)* **do**
  IncreaseFlow(c, f, p)
end
for $i := 1$ **to** n **do**
  for $j := 1$ **to** n **do**
    if $i = j$ **then** $a_{ii} = 0$
    else $a_{i,j} = f(v_i, w_j)$
  end
end

Algorithm 6: BFS(c, f)

Input: Flow network
Output: Parent sequence p describing one of the two paths
Push v to the end of q
while q *is not empty* **do**
  Pop the first element of q into x
  foreach y *such that* (x, y) *is an arc of the residual network* **do**
    if y *is not marked as visited* ***and*** Valid*(x, y, c, f, p, e)* **then**
      $p_y := x$
      if $x = v$ **then** $e_y := y$
      else $e_y := e_x$
      Mark y as visited and push it to the end of q
    end
  end
end
return True if w is marked as visited

In order to check whether the flow can be increased on the symmetric pair of a path, for each visited node x we store the value e_x which keeps the node that succeeds v on the path to x, thus the path will respect the following pattern: $v, e_x, \ldots, x$. The values of sequence e can be determined using simple recurrence relations, by applying dynamic programming.

Algorithm 7: VALID(x, y, c, f, p, e)

Input: Arc (x, y), flow network, sequences p and e
Output: TRUE if p_y should be set to x
if $y = w$ **then**
  Let $v_i = e_x$ and $w_j = x$
  if $v_j \neq e_x$ **then**
    return TRUE if arc (w_i, w) is in the residual network
  else
    return TRUE if arc (w_i, w) has capacity greater than 1 in the residual network
  end
else if $x = v_i$ ***and*** $y = w_j$ *for some* $i, j \in \{1, 2, \ldots, n\}$ **then**
  if (v_j, w_i) *is not in the residual network* **then return** FALSE
  Consider the path $B = v, v_{k_1} = e_x, w_{k_2}, v_{k_3}, w_{k_4}, \ldots, x$ ending in x according to p
  if $w_i \in B$ **then**
    return FALSE if v_j is the element before w_i in B
  end
else if $x = w_i$ ***and*** $y = v_j$ *for some* $i, j \in \{1, 2, \ldots, n\}$ **then**
  Check similarly to the previous case
end
return TRUE

We note that building path B introduces an additional linear factor to the time complexity to our algorithm compared to Algorithm 5, yielding to $O(n^6)$.

It's easy to see the correctness of the algorithm by the following argument. If there exists a solution, then at each step there must exist a pair of symmetric augmenting paths, and if such a pair of paths exist, the algorithm will always find one. After finding a pair of augmenting paths, the flow of the network will increase by 2, so by mathematical induction the algorithm will terminate correctly by finding a solution if one exists, when the flow of the network will be equal with twice the number of edges of the undirected graph we are looking for.

Algorithm 8: IncreaseFlow(c, f, p)

Input: Flow network, sequence p
Output: Flow network with increased flow
$y := w$
while $y \neq v$ **do**
  $x := p_y$
  Increase flow on arc (x, y)
  if $x = v_i$ *and* $y = w_j$ *for some* $i, j \in \{1, 2, \ldots, n\}$ **then**
    Increase flow on arc (v_j, w_i)
  else if $x = w_j$ *and* $y = v_i$ *for some* $i, j \in \{1, 2, \ldots, n\}$ **then**
    Increase flow on arc (w_i, v_j)
  else if $p_x = v$ **then**
    Let $v_i = x$
    Increase flow on arc (w_i, w)
  else if $y = w$ **then**
    Let $w_j = x$
    Increase flow on arc (v, v_j)
  end
  $y := x$
end

4 Conclusions

Despite the fact that the graph realization problem has been intensively studied, there are still many new ideas.

The necessary and sufficient conditions for the realization problem have been known for long, and from these it is easy to give algorithms, yet their exact description in pseudocode is not in vain, because it helps to investigate the complexity of these algorithms. We also examined the possibility of finding all solutions, excluding isomorphic graphs, and the possibility of a parallel approach for larger graph orders. The algorithm to solve the problem using network flows for directed graphs has been modified so that it can be applied to undirected graphs as well. In addition, we have presented an algorithm that solves the problem by integer linear programming.

In the Appendix a method to determine the closed alternating semipaths is presented, which can be used in the flow algorithm.

References

[1] T. M. Barnes, C. D. Savage, Efficient generation of graphical partitions, *Discrete Appl. Math.* **78,** 1-3 (1997) 17–26. ⇒269

[2] G. Chartrand, O. R. Oellermann: *Applied and Algorithmic Graph Theory*, McGraww-Hill, Inc.1993. ⇒269

[3] C. I. Del Genio, H. Kim, Z. Toroczkai, K.E. Bassler, Efficient and exact sampling of simple graphs with given arbitrary degree sequence. *PLoS ONE* 5,4 (2010) e10012. doi:10.1371/journal.pone.0010012 ⇒269

[4] J. Edmonds, R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, *Journal of the ACM* 19, 2 (1972) 248–264. ⇒282

[5] P. Erdős, T. Gallai, Gráfok előírt fokszámú pontokkal, *Matematikai Lapok* (in Hungarian), 11 (1960) 264–274. ⇒268, 269, 270, 285

[6] P. L. Erdős, I. Miklós: A simple Havel–Hakimi type algorithm to realize graphical degree sequences of directed graphs, *The Electronic Journal of Combinatorics* 17 (2010) #R66 ⇒269

[7] S. L. Hakimi, On realizability of a set of integers as degrees of the vertices of a linear graph I, *Journal of the Society for Industrial and Applied Mathematics*, 10 (1962) 496–506. ⇒268, 269

[8] V. Havel, A remark on the existence of finite graphs, *Časopis pro pěstování matematiky* (in Czech), 80 (1955) 477–480. ⇒268, 269

[9] A. Iványi, L. Lucz, T. F. Móri, P. Sótér: On Erdős–Gallai and Havel–Hakimi algorithms, *Acta Universitatis Sapientiae, Informatica*, **3,**, (2011) 230–268. ⇒268, 269, 271, 277, 278

[10] Z. Kása, On scattered subword complexity, *Acta Universitatis Sapientiae, Informatica*, **3,** 1 (2011) 127–136. ⇒291

[11] H. Kim, C. I. Del Genio, K. E. Bassler, Z. Toroczkai, Constructing and sampling directed graphs with given degree sequences, *New Journal of Physics* 14 (2012) 023012 (23pp) ⇒268

[12] H. Kim, Z. Toroczkai, P. L. Erdős, I. Miklós, L. A. Székely, Degree-based graph construction, *Journal of Physics A: Mathematical and Theoretical*, Volume 42, Number 39. ⇒268

[13] Kleitman, D. J.; Wang, D. L, Algorithms for constructing graphs and digraphs with given valences and factors, *Discrete Mathematics*, 6, 1 (1973) 79–88. doi:10.1016/0012-365x(73)90037-x ⇒268, 270

[14] A. Kohnert, Dominance order and graphical partitions, *Elec. J. Comb.* **11,** 1 (2004) No. 4. 17 pp. ⇒269

[15] V. M. Malhotra, M. P. Kumar, S. N. Maheshwari, An $O(|V|^3)$ algorithm for finding maximum flows in networks, *Information Processing Letters*, **7,** 6 (1978) 277–278. ⇒283
[16] M. Mihail, N. Vishnoi, On Generating Graphs with Prescribed Vertex Degrees for Complex Network Modeling, *Semantic Scolar*, Published 2003 Corpus ID: 52064211 ⇒280
[17] G. L. Nemhauser, L.A. Wolsey, *Integer and Combinatorial Optimization,* John Wiley & Sons, 1988. ⇒276
[18] F. Ruskey, R. Cohen, P. Eades, A. Scott, Alley CAT's in search of good homes, *Congr. Numer.* **102** (1994) 97–110. ⇒269
[19] A. Tripathi, H. Tyagi, A simple criterion on degree sequences of graphs, *Discrete Applied Mathematics* 156 (2008) 3513–3517. ⇒269
[20] A. Tripathi, S. Venugopalanb, D. B. West, A short constructive proof of the Erdös-Gallai characterization of graphic lists, *Discrete Math.* **310,** 4 (2010) 833–834. ⇒269
[21] L. A. Wolsey, *Integer Programming*, John Wiley & Sons, 1998. ⇒276

Appendix

Determining the closed alternating semipaths

Let $G = (V \cup W, E_1 \cup E_2)$ a bipartite digraph, where $V = \{v_1, v_2, \ldots, v_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$ are the set of vertices, E_1 the set of red arcs, E_2 the set of blue arcs, (an example is in Fig. 10). The arcs are (v_i, w_j) with $i \neq j$.

The problem is to find closed alternating semipaths in which the direction of the arcs and the colors also alternate. In Fig. 10 such a semipaths is:

Before presenting the algorithm, let us recall some notations ([10]) that will be used.

Let us consider a matrix $\mathcal{A}$ with the elements A_{ij} which are sets of strings. Initially elements of this matrix for $i, j = 1, 2, \ldots, n$ are defined as:

$$A_{ij} = \begin{cases} \{v_i w_j\}, & \text{if there exists an arc from } v_i \text{ to } w_j, \\ \emptyset, & \text{otherwise,} \end{cases} \tag{8}$$

If $\mathcal{A}$ and $\mathcal{B}$ are sets of strings, $\mathcal{AB}$ will be formed by the set of concatenation of each string from $\mathcal{A}$ with each string from $\mathcal{B}$, if they have no common letters:

$$\mathcal{AB} = \{ab \mid a \in \mathcal{A}, b \in \mathcal{B}, \quad \text{if } a \text{ and } b \text{ have no common letters}\}.$$

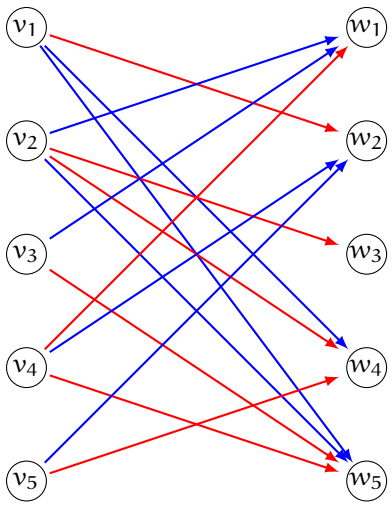


Figure 10:

If $s = s_1 s_2 \cdots s_p$ is a string, let us denote by $'s$ the string obtained from s by eliminating the first character: $'s = s_2 s_3 \cdots s_p$. Let us denote by $'A_{ij}$ the set A_{ij} in which we eliminate from each element the first character. In this case $'\mathcal{A}$ is a matrix with elements $'A_{ij}$.

Let us define for red and blue spanning subgraph of G respectively the matrices $\mathcal{R}$ and $\mathcal{B}$ as in the equation (8). $'\mathcal{R}$ and $'\mathcal{B}$ are defined as above. $\mathcal{B}^T$ represents the transposed matrix of $\mathcal{B}$ in which each element $v_i w_j$ is changed in $w_j v_i$.

The elements of the matrix

$$\mathcal{R}\Big('(\mathcal{B}^T)'\mathcal{R}\Big)^k, \qquad \text{for } k = 1, 2, \ldots, n-1$$

are sets of strings of the form $s_1 s_2 \cdots s_{2k+1}$ (an alternating semipath). If there exists a blue arc (s_1, a_{2k+1}) then $s_1 s_2 \cdots s_{2k+1} s_1$ is a closed alternating semipaths. Algorithm 9 can be easily generalized to matrices of type $m \times n$.

Algorithm 9: Finding closed alternating semipaths

Input: Matrices $\mathcal{R}$ and $\mathcal{B}$ of type $n \times n$
Output: The closed alternating semipaths
$\mathcal{Y} := \mathcal{R}$
$\mathcal{X} := {}'(\mathcal{B}^{T})\,'\mathcal{R}$
for $k := 1$ **to** $n - 1$ **do**
  $\mathcal{Y} := \mathcal{Y}\mathcal{X}$
  for *each string* $s_1 s_2 \cdots s_{2k+1}$ *in each element of* $\mathcal{Y}$ **do**
    if *there exists a blue arc* (s_1, s_{2k+1}) **then**
      print $s_1 s_2 \cdots s_{2k+1} s_1$
    end
  end
end

For the example in Fig. 10 the initial matrices are

$$\mathcal{R} = \begin{pmatrix} \emptyset & \{v_1, w_2\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{v_2, w_3\} & \{v_2, w_4\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{v_3, w_5\} \\ \{v_4, w_1\} & \emptyset & \emptyset & \emptyset & \{v_4, w_5\} \\ \emptyset & \emptyset & \emptyset & \{v_5, w_4\} & \emptyset \end{pmatrix}$$

$$\mathcal{B} = \begin{pmatrix} \emptyset & \emptyset & \emptyset & \{v_1, w_4\} & \{v_1, w_5\} \\ \{v_2, w_1\} & \emptyset & \emptyset & \emptyset & \{v_2, w_5\} \\ \{v_3, w_1\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \{v_4, w_2\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \{v_5, w_2\} & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

and the algorithm gives us the following closed alternating semipaths of length 4, 6 and 8 respectively (closed alternating semipath of length 10 can not exist):

- $v_1 w_2 v_5 w_4 v_1$, $v_5 w_4 v_1 w_2 v_5$,
- $v_1 w_2 v_4 w_1 v_2 w_4 v_1$, $v_1 w_2 v_4 w_5 v_2 w_4 v_1$, $v_2 w_4 v_1 w_2 v_4 w_1 v_2$, $v_2 w_4 v_1 w_2 v_4 w_5 v_2$, $v_4 w_1 v_2 w_4 v_1 w_2 v_4$, $v_4 w_5 v_2 w_4 v_1 w_2 v_4$,
- $v_1 w_2 v_4 w_1 v_3 w_5 v_2 w_4 v_1$, $v_2 w_4 v_1 w_2 v_4 w_1 v_3 w_5 v_2$, $v_3 w_5 v_2 w_4 v_1 w_2 v_4 w_1 v_3$, $v_4 w_1 v_3 w_5 v_2 w_4 v_1 w_2 v_4$.

From these only the following are different:
$v_1 w_2 v_5 w_4 v_1$, $v_1 w_2 v_4 w_1 v_2 w_4 v_1$, $v_1 w_2 v_4 w_5 v_2 w_4 v_1$, $v_1 w_2 v_4 w_1 v_3 w_5 v_2 w_4 v_1$.

Received: June 25, 2023 • Revised: September 25, 2023