Parallel Digraphs-Building Computer Algorithm For Finding a Set of Characteristic Polynomial Realisations of Dynamic System

Submitted: 8th August 2016; accepted: 1st September 2016

Krzysztof Hryniów, Konrad Andrzej Markowski

DOI: 10.14313/JAMRIS_3-2016/23

Abstract:
This paper presents in-depth the parallel computer algorithm for the determination of characteristic polynomial realisations of dynamic system. The main differences between the depicted method and other state-of-the-art solutions include finding not few realisations, but a whole set, and the fact that the found realisations are always minimal among all possible. As digraphs-building methods used in the algorithm are NP-complete or NP-hard problems, the algorithm is paralleled and GPGPU (General-Purpose computing on Graphics Processor Units) computation is proposed as the only feasible solution. The article describes in detail the proposed method, discusses it's complexity, presents optimisation solutions and still open problems. The working algorithm is illustrated with a numerical example and compared to results of other known methods.

Keywords: *dynamic system, GPGPU, characteristic polynomial, digraphs, algorithm*

1. Introduction

Many problems for dynamic systems (both positive and not) are still not completely solved, for example: the determination of lower and upper bounds of reachability index [8, 11], determination of reachability index set [7,12,34], etc. One of such problems is the realisation problem. In many research studies, we can find a canonical form of the system [27, 30, 31, 33] i.e. constant matrix form, which satisfies the system described by the transfer function. With the use of this form, we are able to write only one realisation of the system [4, 37, 39], while there exist many possible solutions. This means that we can find many sets of matrices which fit into the system transfer function, but methods other then the canonical forms are required.

The digraphs theory was applied to the analysis of dynamical systems was proposed for the first time in the paper [11] to the analysis of positive two-dimensional systems. Since then, there were more instances of using this theory in research on dynamic systems. Earlier in [19] and [20], a solution for finding a set of possible realisations of the characteristic polynomial was proposed, but due to a complicated nature of the problem further experimentation shown that it tended to find some improper solutions and the practical implementation was slow. In this article, we propose an upgraded and full version of the computer algorithm based on research results achieved during practical testing of earlier algorithms, optimised to be able to achieve results in a limited time. The algorithm finds a set of possible digraphs realisations of characteristic polynomial in spite of such task being a NP-hard problem. To achieve such a solution, parallel computing with the use of GPUs (Graphics Processing Units) is used.

1.1. Notion

In this paper the following notion will be used. The set $n \times m$ real matrices will be denoted by $\mathbb{R}^{n\times m}$ and $\mathbb{R}^n = \mathbb{R}^{n\times 1}$. If $\mathbf{G} = [g_{ij}]$ is a matrix, we write $\mathbf{G} \geqslant 0$ (matrix $\mathbf{G}$ is called non-negative), if $g_{ij} \geqslant 0$ for all i, j; $\mathbf{G} > 0$ (matrix $\mathbf{G}$ is called positive), if $\mathbf{G} \geqslant 0$ and any $g_{ij} > 0$; $\mathbf{G} \gg 0$ (matrix $\mathbf{G}$ is called strictly positive), if $g_{ij} > 0$ for all i, j. The set of $n \times m$ real matrices with non-negative entries will be denoted by $\mathbb{R}_+^{n\times m}$ and $\mathbb{R}_+^n = \mathbb{R}_+^{n\times 1}$. The $n \times n$ identity matrix will be denoted by $\mathbf{I}_n$. For more information about the matrix theory, an interested reader is referred, for instance, to [3, 16].

1.2. 2-D Systems

Consider the two-dimensional (2D) general model $\sum = (\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \mathbf{C}, \mathbf{D})$ [9] described by the equation:

$$\begin{aligned} x_{i+1,j+1} = \quad & \mathbf{A}_0 x_{i,j} + \mathbf{A}_1 x_{i,j+1} + \mathbf{A}_2 x_{i+1,j} + \\ & + \mathbf{B}_0 u_{i,j} + \mathbf{B}_1 u_{i,j+1} + \mathbf{B}_2 u_{i+1,j} \\ y(i,j) = \quad & \mathbf{C} x_{i,j} + \mathbf{D} u_{i,j} \end{aligned} \tag{1}$$

where $x_{i,j} \in \mathbb{R}^n$, $u_{i,j} \in \mathbb{R}^m$ and $y_{i,j} \in \mathbb{R}^p$ is state, input and output vector, respectively at the point (i, j) and $\mathbf{A}_k \in \mathbb{R}^{n\times n}$; $\mathbf{B}_k \in \mathbb{R}^{n\times m}$; $k = 0, 1, 2$; $\mathbf{C} \in \mathbb{R}^{p\times n}$, $\mathbf{D} \in \mathbb{R}^{p\times m}$.

From (1) for $\mathbf{B}_1 = \mathbf{B}_2 = 0$ we obtain the first Fornasini-Marchesini model [9] and for $\mathbf{A}_0 = 0$ and $\mathbf{B}_0 = 0$ the second Fornasini-Marchesini model [9].

The transfer matrix $\mathbf{T}(z_1, z_2) \in \mathbb{R}_+^{p\times m}$ of the model (1) is given by:

$$\begin{aligned} \mathbf{T}(w_1, w_2) = \mathbf{C}\left[\mathbf{I} - \mathbf{A}_0 w_1 w_2 - \mathbf{A}_1 w_1 - \mathbf{A}_2 w_2\right]^{-1} \cdot \\ \cdot \left[\mathbf{B}_0 w_1 w_2 + \mathbf{B}_1 w_1 + \mathbf{B}_2 w_2\right] + \mathbf{D}. \end{aligned} \tag{2}$$

In the paper we can assume without loss of generality that the $\mathbf{D} = 0$, as symbolic form of the transfer matrix will be the same, as in the model matrix $\mathbf{D}$ influences only numeric values of the terms.

1.3. Characteristic Polynomial

Let $\mathbb{F}$ be a field e.g., of the real number $\mathbb{R}$. The function $P(w_1, w_2)$ of the variable w_1, w_2, is called polyno-

mial:

$$p(w_1, w_2) = \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} a_{i_1,i_2} w_1^{i_1} w_2^{i_2} \tag{3}$$

in the variables w_1, w_2, over the field $\mathbb{F}$, where $a_{i_1,i_2} \in \mathbb{F}$ are called the coefficients of the polynomial.

The set of polynomial (3) over the field $\mathbb{F}$ will be denoted by $\mathbb{F}[w_1, w_2]$.

If $a_{n_1,n_2} \neq 0$, then the non-negative integer $n = n_1 + n_2$ is called the degree of a polynomial and is denoted $deg\, p(w_1, w_2)$, ie., $n = deg\, p(w_1, w_2)$. The polynomial is called monic, if $a_{n_1,n_2} = 1$ and zero polynomial, if $a_{i_1,i_2} = 0$.

For example, for a two-dimensional system the characteristic polynomial consists of two variables: z_1 and z_2 if we have a discrete time system; s_1 and s_2 if we have a continuous time system; z and s if we have a hybrid system.

Interested reader may find definition and properties of the characteristic polynomial in books on linear algebra, for example in [5, ch. 9].

1.4. Digraphs

A directed graph (called also digraph) $\mathcal{D}$ consists of a non-empty finite set $\mathbb{V}(\mathcal{D})$ of elements called vertices and a finite set $\mathbb{A}(\mathcal{D})$ of ordered pairs of distinct vertices called arcs. We call $\mathbb{V}(\mathcal{D})$ the vertex set and $\mathbb{A}(\mathcal{D})$ the arc set of $\mathcal{D}$. We will often write $\mathcal{D} = (\mathbb{V}, \mathbb{A})$ which means that $\mathbb{V}$ and $\mathbb{A}$ are the vertex set and arc set of $\mathcal{D}$, respectively. The order of $\mathcal{D}$ is the number of vertices in $\mathcal{D}$. The size of $\mathcal{D}$ is the number of arc in $\mathcal{D}$. For an arc (v_1, v_2), the first vertex v_1 is its tail and the second vertex v_2 is its head. More information about the digraph theory is given in [1, 13, 14, 36].

A two-dimensional digraphs $\mathcal{D}^{(2)}$ is a directed graph with two types of arcs and input flows. For the first time, this type of digraph was presented in papers [10] and [11]. When we generalise this approach, we can define n-dimensional digraphs. In this case we have q types of arcs and input flows. There exists $\mathcal{A}_1$–arc (or $\mathcal{A}_2$–arc,...,$\mathcal{A}_q$–arc) from vertex v_j to vertex v_i if and only if the (i, j)–th entry of the matrix $\mathbf{A}_1$ (or $\mathbf{A}_2$,...,$\mathbf{A}_q$) is non-zero. There exist $\mathcal{B}_1$–arc (or $\mathcal{B}_2$–arc,...,$\mathcal{B}_q$–arc) from source s_m to vertex v_j if and only if the (i, m)–th entry of the matrix $\mathbf{B}_1$ (or $\mathbf{B}_2$,...,$\mathbf{B}_q$) is non-zero.

For the system described by the matrices

$$\mathbf{A}_1 = \begin{array}{c} v_i \backslash v_j \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 0 & 2 & 0 \\ 3 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array}, \quad \mathbf{A}_2 = \begin{array}{c} v_i \backslash v_j \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 2 \\ 7 & 0 & 0 \end{bmatrix} \end{array} \tag{4}$$

we can draw two-dimensional digraph $\mathcal{D}^{(2)}$ presented in Figure 1 consisting of vertices v_1, v_2 and v_3.

1.5. GPGPU

GPU (Graphics Processing Unit) is a single-chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is dedicated to high-performance 3D graphics, but can also be used for GPGPU (General-Purpose computing on Graphics Processor Units) problems, as it is suited for parallel programming with coherent and predictable memory access. In such problems, GPU and CPU co-processing (sequential parts of algorithms are run on CPU and parallel parts are run on GPU) outstrips traditional CPU-based algorithms due to its high computational capacity [32].

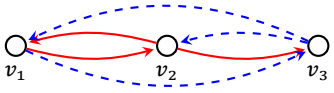

Fig. 1. Digraph $\mathcal{D}^{(2)}$ corresponding to (4)

GPGPU solutions allow to increase the speed of the algorithm (usually by one or two orders of magnitude, depending on problem and implementation), but parallel execution of the algorithm and unique GPU architecture require different construction of the algorithm and memory allocation solutions, as GPU architecture is optimised for executing many concurrent threads slowly, rather than executing a single thread quickly and its memory is constructed and allocated differently. One of important features of GPGPU algorithms are kernels – functions executed in parallel (synchronous or asynchronous) way only on device (GPU card) that are allocated into blocks and grids (2D and 3D structures) with shared memory. Kernels allow very fast concurrent computation, especially for a problem when tasks share some common features – as in graph exploration problems [15]. CUDA is a parallel computing platform and programming model developed by NVIDIA Corporation, increasing computing performance for parallel problems with the use of GPUs.

2. Problem Statement

Our task is the following: for a given characteristic polynomial (3) determine entries of the state matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ using two-dimensional $\mathcal{D}^{(2)}$ digraphs theory. The dimension of the state matrices must be the minimal among the possible ones and cannot contain additional coefficients of the characteristic polynomial.

2.1. State-of-the-art

Currently, entries of the state matrices are determined using canonical forms, a method which gives one of possible realisations of the characteristic polynomial [2, 28, 33]. State-of-the-art algorithms were proposed in [4,26,29,38–40] and are compared briefly to the algorithm presented in this paper in the latter part of the article (Section 4.4).

There are no known methods of finding a set of all possible realisations for a given characteristic polynomial, due to the complexity of the problem. First proposition of such a method was given in [19] and [20], but the proposed method was only theoretical and extensive testing showed that it was not feasible for practical implementation as the problem of finding all possible realisations of a given polynomial is

of such complexity that it cannot be solved in reasonable time even by the brute-force GPGPU method [18]. In [22, 24, 35] further improvements to the proposed algorithm were made and in [23] correspondence between digraph structures and ability to obtain complete results using the algorithm was discussed.

2.2. Proposed Solution

This paper presents for the first time the complete implementation of digraph-based algorithm for finding a set of solutions in form of state matrices of 2D dynamic system. The algorithm starts with creating digraphs for all monomials in the characteristic polynomial, then joins them by the use of disjoint union to create all possible variants of digraphs representing the polynomial realisation. The algorithm uses growth and prune steps to eliminate redundant solutions before the main computational step that is executed concurrently on multiple CUDA kernels to achieve speed needed due to the complexity of the problem (as explained in Section 4). Result of the algorithm's work is in the form of state matrices $\mathbf{A}_1$ and $\mathbf{A}_2$. Algorithm is explained in detail in Section 3.

3. Main Result

Remark 1 *In algorithm's pseudo-code, all arrays and vectors are numbered starting from 1 for clarity. Also, vectors that are used in the GPU-part of the algorithm are often presented as matrices (despite CUDA allowing only the use of vectors on the device part of the algorithm), so it is easier for the reader to understand how the algorithm works.*

Function 1 Main

Output: All possible polynomial realisations with information: {valid / no intersection / incorrect number of cycles}

1: $solutions = 0$; // Total number of solutions
2: $[number, colours, monomials] \leftarrow loadPolynomial()$; // $number$ – number of monomials in polynomial
3: Create $data$ and $legend$ matrices;
4: **for** $mon = 1$ **to** $number$ **do**
5: $size \leftarrow$ sub-graph size for given monomial;
6: $monomialVertexStart(monomials, mon, realisations, colours, data)$;
7: $monomialVertexPrune(data, legend, mon, size, colours, size - 1)$;
8: $vertices$ = max($size$);
9: Determine number of $cycles$ in characteristic polynomial;
10: $createDigraphs(data, legend, number, vertices, colours, cycles)$;

Function $loadPolynomial()$ (line 2) returns (through pointers) a number of monomials, number of variables (colours) in the polynomial and information about monomials in form $monomials[mon][colour\_number] = colour\_power$; matrix $data$ contains information about possible monomial realisations in the form $[solutions] = [realisations]$, while $legend$ matrix contains information about the number and size of those realisations in the form $[monomial] = [number of realisations; size of monomial]$.

Function 2 monomialVertexStart

Parameters: $monomials, mon, realisations, colours, data$
Output: $data, realisations$

1: $monomials2$ = $monomials[mon]$; // Vector is created from one row of the array
2: $i \leftarrow$ number of position (colour) of first non-zero value in $monomials[mon]$
3: $realisations[1] = i$;
4: $monomials2[i] = monomials2[i] - 1$;
5: **if** $(size! = 1)$ **then**
6: $monomialVertex(monomials2, mon, realisations, 2, size, colours, data)$;

Function 2 is the first part of the growth part of algorithm, used for creation of new realisations by means of DFS (Depth-first Search) on a sub-graph. It creates a part of realisation for the first vertex, which reduces the number of redundant realisations created by Function 3, as each realisation not starting with the first non-zero colour can be obtained from other realisations by means of swapping columns / rows.

Function 3 monomialVertex

Parameters: $monomials, mon, realisations, vertex, size, colours, data$
Output: $data$

1: **for** $i = 1$ **to** $colours$ **do**
2: **if** $(monomials[i]! = 0)$ **then**
3: **for** $j = 1$ **to** $colours$ **do**
4: **if** $(j! = i)$ **then**
5: $monomials2[j] = monomials[j]$;
6: **else**
7: $monomials2[j] = monomials[j] - 1$;
8: $realisations[vertex] = i$;
9: **if** $(vertex! = size)$ **then**
10: $monomialVertex(monomials2, mon, realisations, vertex + 1, size, colours, data)$;
11: **else**
12: $solutions + 1$;
13: $data[solutions] = realisations$; // Recurrent growth algorithm finished and all information about given realisation is obtained

Recurrent Function 3 forms the core of the growth part of the algorithm. It is executed for all possible combinations of monomial realisations for a given sub-graph size and number of colours. Some of the realisations obtained can be redundant as they will be removed by the prune step of the algorithm in Function 4. In lines 1–8, the function puts another element into $realisations$ vector and after it is full (line 11) a given realisation is added into the $data$ matrix as a new solution.

Function 4 monomialVertexPrune

Parameters: $data, legend, mon, size, colours, shift$
Output: $data, legend$

1: $legend[mon][1] = legend[mon-1][2]$; // in the case of first monomial, $legend[1][1] = 1$;
2: $sol = solutions - legend[mon][1] + 1$; // number of solutions for a given monomial
3: Create $hash$ for $data$ matrix;
4: **for** $x = 1$ **to** $shift$ **do**
5: $removals = 0$; // Number of removed solutions
6: Create $hash2$ for $data$ matrix shifted by x positions;
7: **for** $i = 2$ **to** sol **do**
8: **for** $j = 1$ **to** $(i - 1)$ **do**
9: **if** $(hash[i] == hash2[j])$ **then**
10: // If both hash functions have the same values, it means the second realisation can be achieved by shifting rows/columns and it is redundant
11: $hash[i] = 0$;
12: $solutions - 1$;
13: $removals + 1$;
14: $data$ corresponding to j is zeroed;
15: **BREAK**
16: **if** $(removals! = 0)$ **then**
17: Sort $data$ matrix, so all non-zero elements are before zeroes
18: // Information about realisations is inserted into $legend$ matrix;
19: $legend[mon][2] = solutions$;
20: $legend[mon][3] = size$;

Function 4 forms the prune part of a candidate generation algorithm. All redundant solutions generated earlier in Functions 2 and 3 are removed. To achieve this, the fast algorithm checks control sums (hashes) that are unique – if two hashes are the same, it means that the second solution can be obtained from the first by classical shifting of rows and/or columns and one of the solutions is redundant. To check for all possibilities, there is a need ($size$ - 1) shifts. Hashes are generated as follows:

$$\sum_{e=1}^{size} data[x][e] * colours^{e}, \tag{5}$$

where e is a number of elements in the solution vector, x is a number of solutions.

After each step, the $data$ matrix is sorted, so that non-empty elements are placed in a continuous form. If the function in any step removes solutions to just one for a given monomial (or if after the growth step there is only one solution), it breaks loops to avoid unnecessary work. After the process of pruning of generated solutions is finished, current information about solutions is stored in the $legend$ matrix, which contains information about a number of solutions and their size.

After creating and checking all monomial realisations, Function 5 creates all possible combinations of polynomial realisations. Each possible combination is assigned to a separate CUDA kernel on GPU for parallel computation. As each monomial is realised on a sub-graph, for sub-graphs with the size lower than the size of polynomial graph there are different possible variants of placement of the monomial (as explained in Remark 2), so the total number of kernels needed is a product of the products of the number of monomial realisations and variants of placement of monomial. Each kernel is assigned to its own information about numbers of monomial realisations and variants through kernel grid parameters, because of GPU memory allocation and sharing features.

Function 5 createDigraphs

Parameters: $data, legend, number, size, colours, cycles$
Output: Set of matrices A_1, A_2, B_1, B_2 for each realisation and information if the realisation is proper

1: Determine a number of possible variants of placement of each monomial, as $variants$;
2: $kernels = \prod_{mon=1}^{number} sol[mon] * variants[mon]$, where $sol[mon] = (legend[mon][2] - legend[mon][1]) + 1$;
3: Create A, B and $intersection$ matrices;
$A_n^{size \times size} = \begin{bmatrix} 0\ 0 \dots 0 \\ \vdots\ \vdots \ddots \vdots \\ 0\ 0 \dots 0 \end{bmatrix}$;
$B_n^{1 \times size} = [0\ 0 \dots 0]$; for $n = 1$ to $colours$;
4: Fill B matrices with data about arcs between sources and graph vertices;
5: Allocate and transfer data to GPU;
6: $createSolutionKernel(data, legend, number, solutions, vertices, colours, A, intersection, cycles)$;
7: Transfer data from GPU;
8: Synchronisation;
9: $(I - Az^{-1})$ check for every realisation where $intersection > 0$;

Remark 2 *Aside from a number of possible combinations of monomial realisations (which are stored in the data and legend matrices) there is a number of variants of possible adding sub-digraphs. First monomial of the same size as the size of digraphs representing polynomial realisation will always have only 1 variant, as each other variant can be obtained by re enumerating vertices. For each other monomial realisation, there is a number of variants represented by the equation*

$$variants = \prod_{i=0}^{n-1} (m - i), \tag{6}$$

where: n is the size of sub-graph, and m is the size of polynomial digraphs realisation.

Example 1 *1-vertex sub-graph added to 5-vertex digraphs will have 5 variants (as the only possibilities are placing the 1-cycle on one of five vertices of digraphs), while 3-vertex sub-graph will have 60 variants (*$5 \cdot 4 \cdot 3$*).*

First part of kernel fills $\mathbf{A}_1, \mathbf{A}_2$ matrices with connections between vertices in digraphs in such a way

Function 6 createSolutionKernel - Part One

Parameters: $data, legend, number, solutions, vertices, colours, A, intersection, cycles$

Output: Each kernel returns matrices A_1, A_2 for a given realisation r, along with information if realisation is proper

1: Realisation number vector R and matrix of variants V are assigned to each kernel;
2: Largest monomial is chosen from $data$ based on $legend[r_1]$;
3: **for** $vertex = 1$ **to** $vertices$ **do**
4: $c = data[legend[1][1] + r_1][vertex]$;
5: **if** $(vertex == vertices)$ **then**
6: $next\_vertex = 1$
7: **else**
8: $next\_vertex = vertex + 1$
9: $A_c[vertex][next\_vertex] = 1$;
10: $intersection[vertex] = 1$;
11: **for** $i = 2$ **to** $number$ **do**
12: $common2 = 0$
13: **for** $j = 1$ **to** $legend[i][3]$ **do**
14: // $legend$ stores size of all monomials
15: $variant = legend[i][1] + r_i$;
16: $c = data[variant][j]$
17: $vertex = V[number][j]$
18: **if** $(j == legend[i][3])$ **then**
19: $next\_vertex = V[number][1]$
20: **else**
21: $next\_vertex = V[number][j + 1]$
22: $A_c[vertex][next\_vertex] = 1$;
23: $intersection2[vertex] = 1$;
24: **for** $j = 1$ **to** $vertices$ **do**
25: **if** $(intersection2[j] == 0)$ **then**
26: $intersection[j] = 0$;
27: **for** $j = 1$ **to** $vertices$ **do**
28: **for** $k = 1$ **to** $vertices$ **do**
29: $P[j][k] = \sum_{i=1}^{colours} A_i[j][k]$
30: $Q \leftarrow testCyclesKernel(P)$;
31: **if** $(\sum_{i=1}^{size} intersection[i] == 0)$ **then**
32: **print** There is no common part of digraphs
33: **return** $A, intersection$;
34: **BREAK;**

that each 1D digraph is represented by A matrix for a given colour. Then matrix **P** is created as a sum of $\mathbf{A}_1$ and $\mathbf{A}_2$ matrices as it represents all connections in digraphs and using it instead of **A** matrices in the latter part of the algorithm (second part of Function 6) allows for much faster computation of a number of existing cycles. Also, the function checks intersection of disjoint unions of sub-graphs creating digraphs and all realisations where intersection is ∅ are improper.

Function $testCyclesKernel(P)$ in line 30 is used to create in a parallel way all needed matrices Q_n presented in Function 6 (part two) line 17. Matrices Q_n are created by removing from square matrix **P** (representing all connection in digraphs, regardless of colour) all rows and columns with the exception of i_1-th, ..., i_n-th, where $n \leq size, i_m < i_m + 1; m = 1, \ldots, n-1$.

Function 6 is executed in a parallel way – each kernel calculates one realisation and one of variants that are assigned through kernel grid addressing as presented in [17] allowing fast access to shared memory features of GPU, consistent address pool and easy translation between nD matrices (used by CPU) and 1D vectors (used by GPU). Each kernel is assigned to its own realisation vector *R* which consists of numbers of chosen realisations for each monomial.

Example 2 *Let's assume that for tested polynomial*

$$d(z_1^{-1}, z_2^{-1}, z_3^{-1}) = 1 - z_1^{-2}z_2^{-1}z_3^{-1} - z_1^{-1}z_2^{-1}z_3^{-1} - z_1^{-1}z_2^{-1} - z_3^{-1}$$

$$data = \begin{bmatrix} 1\,1\,2\,3 \\ 1\,1\,3\,2 \\ 1\,2\,1\,3 \\ 1\,2\,3 \\ 1\,3\,2 \\ 1\,2 \\ 3 \end{bmatrix} \text{ and } legend = \begin{bmatrix} 1\,3\,4 \\ 4\,5\,3 \\ 6\,6\,2 \\ 7\,7\,1 \end{bmatrix},$$

then first kernel will be assigned to realisation vector in form $R = [0\,0\,0\,0]$*;* $(n+1)$*-th kernel, where* n *is number of variants possible, will be assigned realisation vector* $R = [0\,1\,0\,0]$*;* $(2n+1)$*-th kernel will be assigned* $R = [1\,0\,0\,0]$*, ..., and finally* $(5n+1)$*-th kernel will be assigned* $R = [2\,1\,0\,0]$*. In this example, 3-rd and 4-th positions in R vector do not change, as there is only one possible realisation for those monomials.*

Each kernel has also its own vector V, which stores positions of vertices for variants of monomial realisations.

Example 3 *Continuing with the polynomial from Example 2, we have 1 variant for first monomial (as explained earlier in Remark 2) and for next monomials 24, 12 and 4 variants (according to equation (6)). In this case, 6 912 kernels will be used (6 realisations ·24·12·4) and for one of sample kernels vector V presented as matrix will take the form:*

$$V = \begin{bmatrix} 3\,2\,1 \\ 3\,1 \\ 2 \end{bmatrix}$$

in this case, the second monomial is represented by arcs between vertices 3-2, 2-1, 1-3. The third monomial is represented by arcs between 3-1 and 1-3 and the fourth monomial is represented by arc 2-2. All arc colours are determined by information from the $data$ *matrix chosen by the realisation vector.*

Element $p_{i,j} \in \mathbf{P}$ represents the number of different arcs beginning in vertex i and ending in vertex j. The algorithm checks if the number of n-vertex cycles in realisation is proper for the characteristic polynomial. It starts with shorter cycles first, because it is faster, and if the algorithm finds even one additional cycle, it means that the realisation is not valid and then kernel stops working and frees memory. For 1-vertex (line 2) and 2-vertex cycles (line 9), there are

Function 7 createSolutionKernel - Part Two

1: // Realisation is proper only if the number of cycles is the same for digraphs and the characteristic polynomial
2: **if** $(\sum_{i=1}^{size} P[i,i]! = cycles[1])$ **then**
3: **print** Number of 1-vertex cycles is incorrect
4: **return** $A, -1$;
5: **BREAK;**
6: $cycles_2 = 0$;
7: **for all** $i, j \in [0, 1, \dots, size]$, where $i < j$ **do**
8: // $Q_{i,j} = \begin{bmatrix} p_{i,i} & p_{i,j} \\ p_{j,i} & p_{j,j} \end{bmatrix}$;
9: $cycles_2 = cycles_2 + (p_{i,j} * p_{j,i})$;
10: **if** $(cycles_2! = cycles[2])$ **then**
11: **print** Number of 2-vertex cycles is incorrect
12: **return** $A, -2$;
13: **BREAK;**
14: **for** $n = 3$ **to** $sizeof(cycles)$ **do**
15: $cycles_n = 0$;
16: **for all** $i_1, \dots, i_n$ where $n \leq size, i_m < i_m + 1; m = 1, \dots, n-1$ **do**
17: // $Q_n^{n \times n} = \begin{bmatrix} p_{i_1,i_1} & p_{i_1,i_2} & \dots & p_{i_1,i_n} \\ p_{i_2,i_1} & p_{i_2,i_2} & \dots & p_{i_2,i_n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{i_n,i_1} & p_{i_n,i_2} & \dots & p_{i_n,i_n} \end{bmatrix}$;
// All possible n-cycles are checked and a number of existing ones is counted with the use of recurrent Function 7
18: $countCycles(1, 1, 1, n-1)$;
19: **if** $(cycles_n! = cycles[n])$ **then**
20: **print** Number of n-vertex cycles is incorrect
21: **return** $A, -n$;
22: **BREAK;**

fast methods of checking the number of cycles, for n-vertex (where $n \geq 3$) cycles Function 7 is recurrently executed to check all possible cycles and count their sum. There are some additional constraints used to speed up the algorithm, for example if any column is empty, it means that every excision of **P** containing that column cannot have n-vertex cycles and can be automatically discarded.

Function 8 countCycles

Parameters: $X, result, step, stop$
Output: $result$
1: // $X^{1\times(n-1)} = [x_1, x_2, \dots, x_{n-1}]$ and $x_0 = 1$
2: $i = step$;
3: **for** $x_i = 2$ **to** n where $x_i! = x_j, \forall j = 1, \dots, i-1$ **do**
4: $result = result * p_{i_{x_i}, i_{x_{i-1}}}$;
5: **if** $(result == 0)$ **then**
6: **BREAK;**
7: **if** $(step == stop)$ **then**
8: $cycles_n = cycles_n + result * p_{i_{x_{n-1}}, i_{x_1}}$;
9: **else**
10: $countCycles(x, result, step + 1, stop)$;

Function 8 is used for counting fast the number of cycles present in digraphs basing on matrices Q created earlier. It is recurrently executed, so that all possible cycle combinations will be checked, and it returns the sum of existing cycles, which is later checked with the number of cycles in the characteristic polynomial. If the number of cycles is different, it means that adding sub-graphs created some additional cycles, and realisation is improper and should be discarded. If at any point the value of $result$ equals zero (line 5), it means that one of the elements of the determinant is zero and the final value of the given determinant will be zero (so there will be no cycle for this combination), and there is no further need of checking that possibility and the algorithm can move to check for the next combination.

4. Problem Complexity and Algorithm Optimisation

Many graph problems (like graph orientation, coverage, colouring, decomposition, disjunction, connection and finding both the shortest paths and cycles) are proposed or proven as NP-complete or NP-hard for 1D undirected graphs, sometimes even with constraints added [1]. It can be assumed, that for arbitrary multi-dimensional digraphs, those tasks are even more complex and time-consuming. As creating digraphs realisation for the characteristic polynomial includes many operations that are considered NP-complete or NP-hard problems, it is also a NP-hard problem [22, 24]. It can be assumed that its computational complexity will be very high, as:

- it consists of set of NP-complete or NP-hard problems,
- proposed solution is based on operating on arbitrary two-dimensional digraphs, making it more complicated than 1D undirected graphs,
- in [25] it is conjectured that for many NP-complete problems (including NP-complete graph problems and set coverage problems) there are no solutions that are capable of solving the problem under exponential time, and all well-known algorithms have at least exponential computational complexity.

Because of the complexity of the proposed problem, there is a constant need of optimisation – and for that, it is needed to precisely identify complexity of each part of the algorithm, on what elements it depends and how the algorithm can be made faster.

4.1. Algorithm Optimisation

In the algorithm proposed in [21] first optimisations were performed in form of growth/prune steps, which restrict number of solutions that are fully created and tested and in creating fixed solution start, which reduces number of solutions created $\frac{m}{x_1}$ times, as can be seen in equation (7). In [24] there was proposed a set of modifications that causes the algorithm to work faster (better variant of fixed start, omitting prune step when not needed, and different calculation for 1D monomials). In [22] further modifications were proposed in form of modification of creation of intersection set, removal of redundant variants before parallel calculation and skipping the 1-arc cycles, that further improved the speed of the algorithm and reduced the complexity for the most complicated cases. Still, due

to the NP-hard complexity of the problem, the overall computational complexity of the algorithm wasn't lowered below factorial for 2D polynomials and linearithmic for 1D polynomials.

4.2. Computational Complexity of the Growth/Prune Part

The growth part of the algorithm (Functions 2 and 3), in which possible solutions are created, is performed in $c^2(m-1)$ operations for each monomial in the characteristic polynomial, where: c - number of colours; m - size of the monomial. This step creates a number of solutions that form a multiset permutation represented by equation

$$s = \binom{m-1}{(x_1-1), x_2, \ldots, x_c} = \\ = \frac{(m-1)!}{(x_1-1)!\,x_2!\ldots x_c!}, \tag{7}$$

where x_i represents a number of occurrences of i-th colour in the monomial and $x_1 + x_2 + \ldots + x_c = m$. In equation (7), decrementation of x_1 and m represents reduction of a number of the possible solutions by the fixed first choice in Function 2.

For optimisation purposes it is important to be able to determine a number of solutions that are created after the prune step of the algorithm, as it allows us to prepare a number of kernels for each monomial in advance and it is essential for full parallelism of the algorithm.

Theorem 1 *There exist*

$$p = \binom{m-1}{x_1, x_2, \ldots, x_c} = \frac{(m-1)!}{x_1!\,x_2!\ldots x_c!}, \tag{8}$$

possible distinct digraphs solutions for given monomial, where x_i represents number of occurrences of i-th variable in the monomial and $x_1 + x_2 + \ldots + x_c = m$.

Proof Number of distinct digraphs solutions is the same problem as determination of arrangement of n distinct objects along a fixed circle. As we have more than one variable, it is a case of a multi-set permutation which gives $(n!)/(m_1!\,m_2!\ldots m_l!)$ solutions. As the permutation is cyclic, $n!$ is reduced to $(n-1)!$ as the circle can be rotated [6].

Remark 3 *Circle rotation used for digraphs representations of polynomials is synonymous with swapping columns / rows of* **A** *matrix that is used in the control theory.*

When we know the characteristic polynomial, we can determine at the start of the algorithm the number of solutions generated during the growth step (described by equation (7)) and the number of solutions generated after the prune step (described by equation (8)) for each of the monomials. With such information, we can compute different monomials that the characteristic polynomial consists of simultaneously by assigning each to kernel block of the precalculated size and preparing memory blocks for each of the final solutions, which can be calculated independently of others, without incurring the synchronisation problem.

The prune step (Function 4), where redundant solutions are removed, is based heavily on the number of solutions that are being reduced. For each monomial, it is performed in $(m-1)(s\log s + m)$ steps, where s is determined as in equation (7). As the part for each monomial can be run in a parallel way (as we can determine the number of kernels), the algorithm needs

$$c^2(m-1) + (m-1)(s\log s + m) \tag{9}$$

operations in case when enough kernels are available for full parallelisation.

To estimate computational complexity, we should assume the worst case scenario. In such a scenario, we imply that monomials are of size $m \to V$, where V is the number of vertices in polynomial digraphs, and we can transform equation (9) into

$$(V-1)(c^2 + s\log s + V). \tag{10}$$

As can be seen in equation (10), there are two factors that can generate the largest computational effort: c^2 and $s\log s$. Both parts are maximised when $c = V$ (there are as many colours in each monomial as vertices in polynomial digraphs), but for such $x_1 = x_2 = \ldots = x_c = 1$ scenarios prune step can be omitted as presented in [24] and the algorithm needs only $c^2(m-1)$ operations.

For the worst-case condition that is not a $c = m$ scenario, we can assume that $s \to (V-1)!$ and sequential part's computational complexity is factorial and can be presented as $\mathbf{T}(\mathbf{V}) = \mathbf{O}(V!)$ in big O notation.

4.3. Computational Complexity of Digraph Creation Part

Digraph creation part of the algorithm is executed on

$$\prod_1^n \left(\frac{(m-1)!}{x_1!\,x_2!\ldots x_c!} \cdot \frac{V!}{(V-m)!} \right) \tag{11}$$

kernels, each performing

$$\sum_{i=1}^{n} (V(m_i + V) + m_i \log m_i) + V^2 + V\log V^n \tag{12}$$

operations. In the worst case scenario, $m_i \to V$ and equation (12) can be presented as

$$(2n+1)V^2 + nV\log V + V\log V^n, \tag{13}$$

which makes parallel part's computational complexity $\mathbf{T}(\mathbf{V}) = \mathbf{O}(V\log V^n)$ and makes the part of algorithm solvable in linearithmic time, if there are enough kernels available.

4.4. Comparison with Other Algorithms

Using proposed algorithm we can determine all possible multi-dimensional digraph structures that satisfy the characteristic polynomial (3). The number

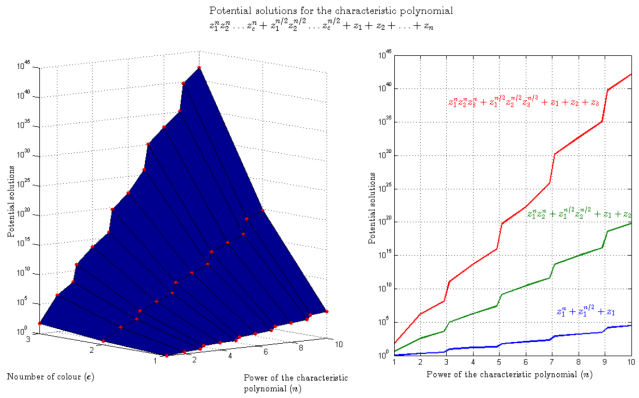

Fig. 2. Potential number of solutions of the polynomial $z_1^n z_2^n \dots z_c^n + z_1^{n/2} z_2^{n/2} \dots z_c^{n/2} + z_1 + z_2 + \dots + z_c$

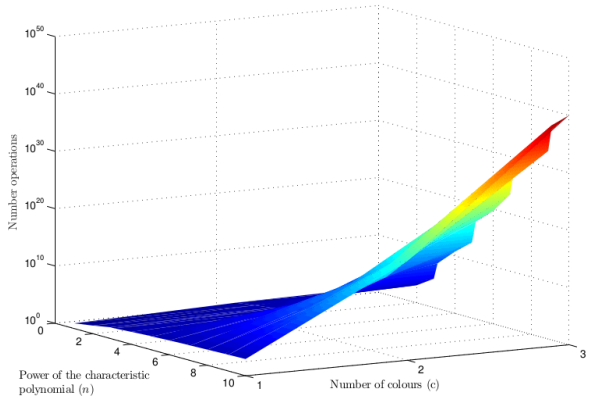

Fig. 3. Number of operations performed for the polynomial $z_1^n z_2^n \dots z_c^n + z_1^{n/2} z_2^{n/2} \dots z_c^{n/2} + z_1 + z_2 + \dots + z_c$ ***with use of Tesla C2050 GPU with*** 10^6 ***parallel threads***

of the solution is bigger with the size of the characteristic polynomial and number of the variables. Let be given the polynomial $z_1^n z_2^n \dots z_c^n + z_1^{n/2} z_2^{n/2} \dots z_c^{n/2} + z_1 + z_2 + \cdot + z_c$ where: n is the power of the polynomial ($n = 1, 2, \dots, 10$); c is the number of colour ($c = 1, 2, 3$). In the Figure 2, we have presented a number of the possible digraph structure. For example, if we have a polynomial in the form:

- $z_1^6 + z_1^3 + z_1$ we have 120 possible digraphs structures;
- $z_1^6 z_2^6 + z_1^3 z_2^3 + z_1 + z_2$, we have 29, 506, 498, 560 possible digraphs structures;
- $z_1^6 z_2^6 z_3^6 + z_1^3 z_2^3 z_3^3 + z_1 + z_2 + z_3$, we have $1.83361492 \cdot 10^{22}$ possible digraphs structures.

The number of the possible digraph structures is growing very fast as we can see in Figure 2.

In the Figure 3, we see, that the algorithm for determination of all possible digraphs representations of the characteristic polynomial is performed in a larger number of operations. The number of operations depends on maximal power of the characteristic polynomial and number of colours. For example if we have the polynomial that:

- consist of one variable in the following form $z_1^6 + z_1^3 + z_1$, then we must make about 1000 operations;
- consist of two variables in the following form $z_1^6 z_2^6 + z_1^3 z_2^3 + z_1 + z_2$, then we must make $1.97 \cdot 10^{17}$ operations;
- consist of three variables in the following form $z_1^6 z_2^6 z_3^6 + z_1^3 z_2^3 z_3^3 + z_1 + z_2 + z_3$, then we must make $1.05 \cdot 10^{40}$ operations.

Tab. 1. Number of the digraph structures and operations

Characteristic polynomial	Digraph structures	Number of operations
$z_1^6 + z_1^3 + z_1$	120	1000
$z_1^6 z_2^6 + z_1^3 z_2^3 + z_1 + z_2$	$29,506,498,560$	$1.97 \cdot 10^{17}$
$z_1^6 z_2^6 z_3^6 + z_1^3 z_2^3 z_3^3 + z_1 + z_2 + z_3$	$1.83361492 \cdot 10^{22}$	$1.05 \cdot 10^{40}$

Presented in the Table 1 numbers of operations were performed using computer with use of graphic card Tesla C2050 with 10^6 parallel threads.

In Table 2, we presented a potential and real number of digraph structures for examples considered in the state-of-the-art papers: [4, 26, 29, 38–40]. Potential solutions hold the solutions that can be created by the algorithm. It should be noted that the potential and real solutions does not contain realisations that we can receive by re-encumbering the vertices in the digraphs. This operation is very simple, but it is not included in the algorithm, as it leads to unnecessary operations and can be obtained by transposition of **A** matrices after the algorithm is finished.

Tab. 2. Potential and real number of the digraph structures

Characteristic Polynomial	Grow	Prune	Solutions	
			Potential	Real
$1 + a_{20}z_2^2 + a_{11}z_1z_2 + a_{10}z_1 + a_{01}z_2$ [(a)]	1	1	4	2
$1 - a_5z_1^2z_2 - a_4z_2^2 - a_3z_1^2 - a_2z_2 - a_1z_1$ [(b)]	1	1	81	2
$1 + d_{21}z_1^2z_2 + d_{30}z_1^3 + d_{11}z_1z_2 + d_{10}z_1 + d_{01}z_2$ [(c)]	1	1	54	4
$1 - \beta_1z_1 - \beta_2z_2$ [(d)] [(h)]	1	1	1	1
$1 + a_{12}z_1z_2^2 + a_{11}z_1z_2 + a_{01}z_2$ [(e)]	1	1	18	6
$1 + z_1^2z_2^2 + z_1z_2$ [(f)]	3	2	24	8
$1 - d_{22}z_1^2z_2^2 - d_{12}z_1z_2^2 - d_{21}z_1^2z_2 - d_{11}z_1z_2 - d_{10}z_1 - d_{01}z_2$ [(g)]	3	2	129 024	44

Examples from: [(a)] [39, pp. 634-635]. [(b)] [40, pp. 1463]. [(c)] [40, pp. 1461]. [(d)] [38, pp. III-292]. [(e)] [39, pp. 637]. [(f)] [26, pp. 636]. [(g)] [29, pp. 6-7]. [(h)] [4, pp. 207].

Tab. 3. Comparison the size of the realisation and the number of the solutions

Characteristic Polynomial	Size ($n \times n$)	Solutions (number)	Proposed algorithm	
			Size	Solutions
$1 + a_{02}z_2^2 + a_{11}z_1z_2 + a_{10}z_1 + a_{01}z_2$ [(a)]	4	1	2	4
$1 - a_5z_1^2z_2 - a_4z_2^2 - a_3z_1^2 - a_2z_2 - a_1z_1$ [(b)]	4	1	3	2
$1 + d_{21}z_1^2z_2 + d_{30}z_1^3 + d_{11}z_1z_2 + d_{10}z_1 + d_{01}z_2$ [(c)]	3	1	3	12
$1 - \beta_1z_1 - \beta_2z_2$ [(d)] [(h)]	2	1	1	1
$1 + a_{12}z_1z_2^2 + a_{11}z_1z_2 + a_{01}z_2$ [(e)]	3	1	3	6
$1 + z_1^2z_2^2 + z_1z_2$ [(f)]	4	1	4	8
$1 - d_{22}z_1^2z_2^2 - d_{12}z_1z_2^2 - d_{21}z_1^2z_2 - d_{11}z_1z_2 - d_{10}z_1 - d_{01}z_2$ [(g)]	4	1	4	44

Examples from: [(a)] [39, pp. 634-635]. [(b)] [40, pp. 1463]. [(c)] [40, pp. 1461]. [(d)] [38, pp. III-292]. [(e)] [39, pp. 637]. [(f)] [26, pp. 636]. [(g)] [29, pp. 6-7]. [(h)] [4, pp. 207].

In Table 3, we presented the comparison of the size of the realisation and the number of the solutions for the examples presented in the papers considered in Table 2. After looking at the table, we can say that the algorithm proposed in this paper gives more realisations and additionally all of them are the minimal among all possible and thus can be proven as superior on grounds of achieved solutions to other state-of-the-art methods. In Table 3 the values marked by red colour are those that give a better results compared to the methods described in the state-of-the art papers.

5. Numerical Example

Let us consider the following example. For the given characteristic polynomial

$$d(z_1^{-1}, z_2^{-1}) = 1 - z_1^{-2}z_2^{-2} - z_1^{-1}z_2^{-1} - z_1^{-1} \quad (14)$$

determine all possible realisations of the state matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ using digraph theory and GPGPU computing method. To solve this problem, we use the parallel algorithm presented in Section 3.

In the first step, we must write the following initial conditions: number of colours in digraphs: $colours = 2$; monomials: $M_1 = z_1^{-2}z_2^{-2}$, $M_2 = z_1^{-1}z_2^{-1}$ and $M_3 = z_1^{-1}$.

Figure 4 presents algorithm structure used to determine characteristic polynomial realisation described by the equation (14). The algorithm starts from function $Main()$ in which monomials in the following form are constructed:

$$monomials = \left\{ \begin{matrix} [1]\,[1] = 2 \\ [1]\,[2] = 2 \\ [2]\,[1] = 1 \\ [2]\,[2] = 1 \\ [3]\,[1] = 1 \end{matrix} \right\}.$$

In the next step for the first monomial $M_1 = z_1^{-2}z_2^{-2}$ from function $Main()$ is starting the function $monomialVertexStart()$ (see box $Mon = 1$ and $Start$ in the Figure 4). In this function, we write the following initial conditions:

$$size = monomials[1][1] + monomials[1][2] = 4;$$
$$monomials2 = \left\{ \begin{matrix} [1] = 2 \\ [2] = 2 \end{matrix} \right\}; \quad realisations[1] = 1.$$

Then from function $monomialVertexStart()$ recurrent function $monomialVertex()$ (named $Vertex$ in latter part of the paper) is started. After this step we obtain all possible combinations of the M_1 monomial realisations for given $size = 4$ and number of colours $colour = 2$. In the considered example, we have:

- $Vertex(1)$ – for $i = 1, j = 1 \rightarrow monomials[1] = 0$; for $i = 1, j = 2 \rightarrow monomials[2] = 2$; $realisations[2] = 1$.
- $Vertex(2)$ – for $i = 2, j = 1 \rightarrow monomials[1] = 0$; for $i = 2, j = 2 \rightarrow monomials[2] = 1$; $realisations[3] = 2$.
- $Vertex(3)$ – for $i = 2, j = 1 \rightarrow monomials[1] = 0$; for $i = 2, j = 2 \rightarrow monomials[2] = 0$; $realisations[4] = 2$.

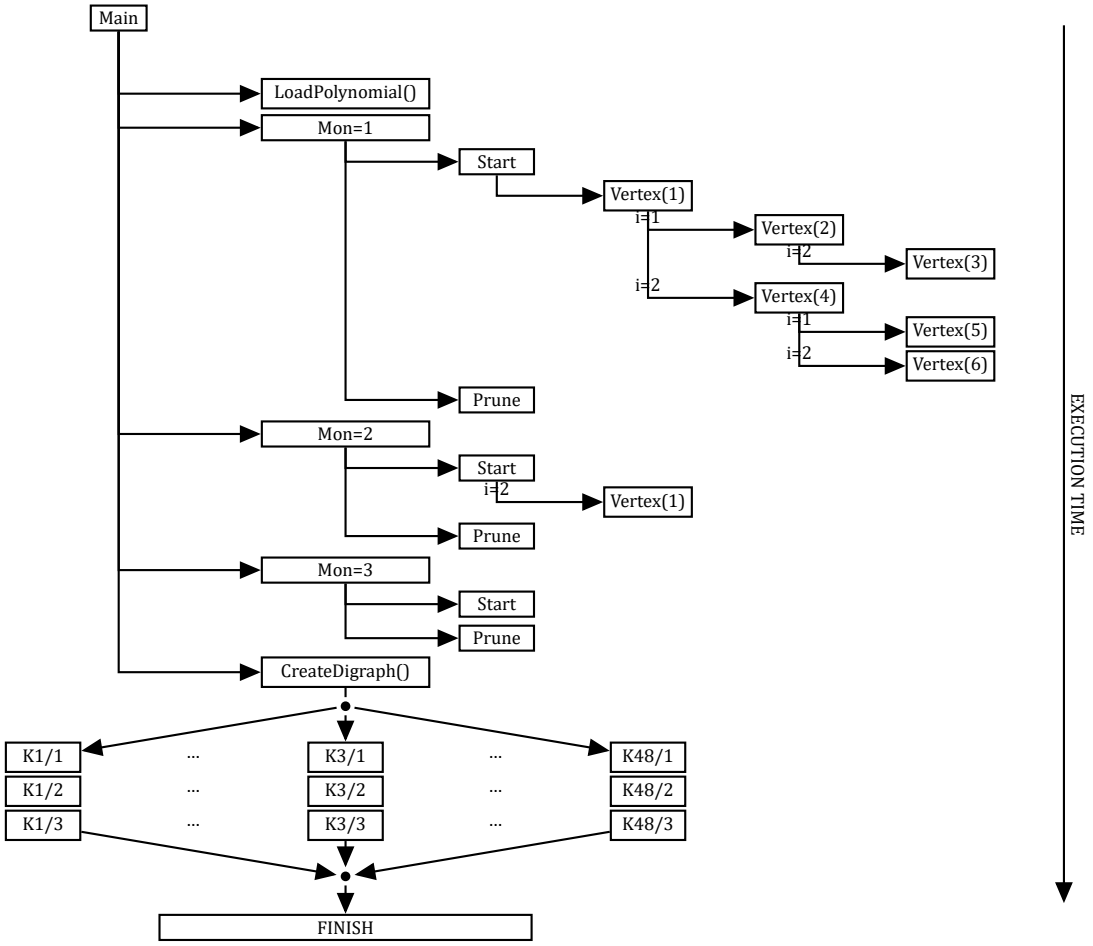

Fig. 4. Workings of the algorithm for example presented in Section 5

In this moment, the condition $vertex! = size$ is not met, and we write the first solution in the form: $solution[1] = [1\ 1\ 2\ 2]$. In the next step, we jump to function $Vertex(1)$ and grow index i to 2.

- $Vertex(1)$ – for $i = 2, j = 1 \rightarrow monomials[1] = 1$; for $i = 2, j = 2 \rightarrow monomials[2] = 1$; $realisations[2] = 2$.
- $Vertex(4)$ – for $i = 1, j = 1 \rightarrow monomials[1] = 0$; for $i = 1, j = 2 \rightarrow monomials[2] = 1$; $realisations[3] = 1$.
- $Vertex(5)$ – for $i = 2, j = 1 \rightarrow monomials[1] = 0$; for $i = 2, j = 2 \rightarrow monomials[2] = 0$; $realisations[4] = 2$.

As $vertex! = size$ is not met, we write the second solution in the form: $solution[2] = [1\ 2\ 1\ 2]$ and jump to function $Vertex(4)$ with index $i = 2$.

- $Vertex(4)$ – for $i = 2, j = 1 \rightarrow monomials[1] = 1$; for $i = 2, j = 2 \rightarrow monomials[2] = 0$; $realisations[3] = 2$.
- $Vertex(6)$ – for $i = 1, j = 1 \rightarrow monomials[1] = 0$; for $i = 2, j = 2 \rightarrow monomials[2] = 0$; $realisations[4] = 1$.

As again $vertex! = size$ is not met, third solution takes the form: $solution[3] = [1\ 2\ 2\ 1]$ and final data matrix takes the form:

$$data = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 2 \\ 1 & 2 & 2 & 1 \end{bmatrix}. \qquad (15)$$

In this moment, we have finished the function $monomialVertex()$, and we return to function $monomialVertexStart()$ and then to function $main()$, from which we start function $monomialVertexPrune()$. In this function, we write the following initial conditions:

$solutions = 3$; $legend[1][1] = 1$; $sol = solution - legend[1][1] + 1 = 3 - 1 + 1 = 3$;

Determine $shift$: $shift = size - 1 = 4 - 1 = 3$.

Determine $hash$ for data matrix (15) using equation (5):

$$hash = \begin{bmatrix} 1 \cdot 2^1 + 1 \cdot 2^2 + 2 \cdot 2^3 + 2 \cdot 2^4 \\ 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 2 \cdot 2^4 \\ 1 \cdot 2^1 + 2 \cdot 2^2 + 2 \cdot 2^3 + 1 \cdot 2^4 \end{bmatrix} = \begin{bmatrix} 54 \\ 50 \\ 42 \end{bmatrix}.$$

In the next step, we create $hash2$ for data matrix (15) shifted by x positions. We obtain:

- for $x = 1$ we have:

$$shift(x = 1) = \begin{bmatrix} 1 & \overbrace{1 & 2 & 2 & 1}^{x=1} & 1 & 2 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 \\ 1 & 2 & 2 & 1 & 1 & 2 & 2 \end{bmatrix} \Rightarrow \qquad (16)$$

$$\Rightarrow hash2 = \begin{bmatrix} 42 \\ 38 \\ 36 \end{bmatrix};$$

(a) for $i = 2, j = 1$, we compare $hash(2)$ with $hash2(1)$. As $50! = 40$ we compare the next pair.

(b) for $i = 2, j = 2$, we compare $hash(2)$ with $hash2(2)$. As $50! = 38$ we compare the next pair.

(c) for $i = 3, j = 1$, we compare $hash(3)$ with $hash2(1)$. In this situation $42 == 42$; therefore we set $solution = 2$ and $removals = 1$ and change $hash$ and $data$ into the following:

$$hash = \begin{bmatrix} 54 \\ 50 \\ 0 \end{bmatrix}, \quad data = \begin{bmatrix} 1\,1\,2\,2 \\ 1\,2\,1\,2 \\ 1\,2\,2\,1 \end{bmatrix}.$$

- for $x = 2$, we have:

$$shift(x=2) = \begin{bmatrix} 1 & 1 & \overset{x=2}{\overbrace{\begin{matrix} 2 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 1 & 2 \end{matrix}}} & \begin{matrix} 2 \\ 1 \\ 2 \end{matrix} \\ 1 & 2 & & \\ 1 & 2 & & \end{bmatrix} \Rightarrow \tag{17}$$

$$\Rightarrow hash2 = \begin{bmatrix} 36 \\ 50 \\ 48 \end{bmatrix}$$

(a) for $i = 2, j = 1$, we compare $hash(2)$ with $hash2(1)$.As $50! = 36$ we compare the next pair.

(b) for $i = 2, j = 2$, we compare $hash(2)$ with $hash2(2)$. As $50 == 50$ we set $solution = 1$ and $removals = 2$ and change $hash$ and $data$ into:

$$hash = \begin{bmatrix} 54 \\ 0 \\ 0 \end{bmatrix}, \quad data = \begin{bmatrix} 1\,1\,2\,2 \\ 1\,2\,1\,2 \\ 1\,2\,2\,1 \end{bmatrix}.$$

Finally, we write $data$ and $legend$ which contain information about realisation in the following form:

$$data = [1\,1\,2\,2]$$
$$legend[mon][2] = solution \Rightarrow legend[1][2] = 1$$
$$legend[mon][3] = size \Rightarrow legend[1][3] = 4.$$

Now we break function $monomialVertexPrune()$, and we come back to function $Main()$. In the same way we follow with monomial M_2 and M_3. Finally, we obtain matrix $data$ and matrix $legend$ in the form:

$$data = \begin{bmatrix} 1\,1\,2\,2 \\ 1\,2 \\ 1 \end{bmatrix}, \quad legend = \begin{bmatrix} 1\,1\,4 \\ 2\,2\,2 \\ 3\,3\,1 \end{bmatrix}.$$

In this moment, we start function $createDigraphs()$ which creates all possible combinations of polynomial realisation. In this function, we write the following initial conditions:

1) Determine number of possible variants (6) of placement of each monomial as $variants[mon]$:
   - For the monomial M_1, we have $variants[1] = 1$; corresponding digraph is presented in Figure 5;
   - For the monomial M_2, we have $variants[2] = 12$; Digraph starting from the first vertex corresponding to monomial M_2 is presented in Figure 6. In this same way, we can draw nine remaining digraphs starting from the second, third and fourth vertex respectively.

Fig. 5. Two-dimensional digraphs corresponding to monomial M_1

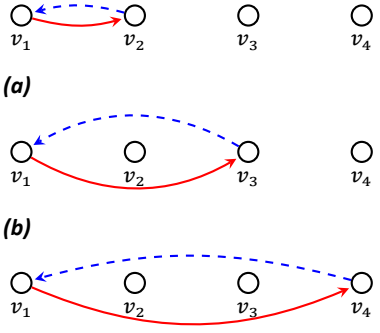

Fig. 6. Two-dimensional digraphs corresponding to monomial M_2

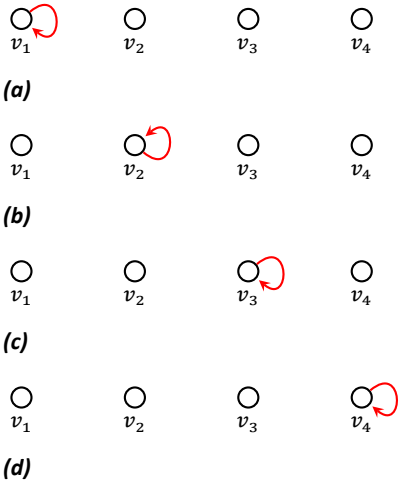

Fig. 7. Two-dimensional digraphs corresponding to monomial M_3

   - For the monomial M_3, we have $variants[3] = 4$; corresponding digraph is presented in Figure 7

2) The number of vertices of digraphs for polynomial realisation is equal to the size of maximal digraphs for monomial realisations, in this example $vertices = 4$, the size of digraphs for M_1;

3) Determine number of kernels needed for computation:$kernels = \prod_{mon=1}^{3} sol[mon] \cdot variants[mon] = (1 \cdot 1) \cdot (1 \cdot 12) \cdot (1 \cdot 4) = 48$;

4) Create $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{B}_1$ and $\mathbf{B}_2$ zeros matrices.

At this moment, we can: create digraphs as a combination each of the monomial realisation, write matrix $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{P}$, write $cycles$ vector and $intersection$ vector. For example:

- For the $kernel[1]$, we take the first realisation of the monomial M_1 (Figure 5), the first realisation of the monomial M_2 (Figure 6a) and the first realisation of the monomial M_3 (Figure 7a). Digraph correspond-

ing to $kernel[1]$ is presented in Figure 8.

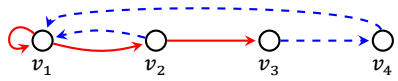

Fig. 8. Two-dimensional digraphs corresponding to $kernel[1]$

$$\underbrace{\begin{bmatrix}1&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&0&0\end{bmatrix}}_{\mathbf{A}_1},\quad \underbrace{\begin{bmatrix}0&1&0&1\\0&0&0&0\\0&0&0&0\\0&0&1&0\end{bmatrix}}_{\mathbf{A}_2},\quad \underbrace{\begin{bmatrix}1&1&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}}_{\mathbf{P}}, \tag{18}$$

$$cycles = [1\,1\,0\,1], intersection = [1\,0\,0\,0].$$

In this situation, we have a common part of monomial digraphs (vertex v_1), and we can assume that $kernel[1]$ is one of possible realisations of characteristic polynomial.

- For the $kernel[3]$, we take the first realisation of the monomial M_1 (Figure 5), the first realisation of the monomial M_2 (Figure 6a) and the third realisation of the monomial M_3 (Figure 7c). Digraph corresponding to $kernel[3]$ is presented in Figure 9.

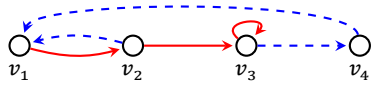

Fig. 9. Two-dimensional digraphs corresponding to $kernel[3]$

$$\underbrace{\begin{bmatrix}0&0&0&0\\1&0&0&0\\0&1&1&0\\0&0&0&0\end{bmatrix}}_{\mathbf{A}_1},\quad \underbrace{\begin{bmatrix}0&1&0&1\\0&0&0&0\\0&0&0&0\\0&0&1&0\end{bmatrix}}_{\mathbf{A}_2},\quad \underbrace{\begin{bmatrix}0&1&0&1\\1&0&0&0\\0&1&1&0\\0&0&1&0\end{bmatrix}}_{\mathbf{P}}, \tag{19}$$

$$cycles = [1\,1\,0\,1], intersection = [0\,0\,0\,0].$$

In this situation, we do not have a common part of monomial digraphs and $kernel[3]$ is not realisation of characteristic polynomial.

- For the $kernel[6]$, we take the first realisation of the monomial M_1 (Figure 5), the first realisation of the monomial M_2 from the second vertex and the second realisation of the monomial M_3 (Figure 7b). Digraph corresponding to $kernel[6]$ is presented in Figure 10.

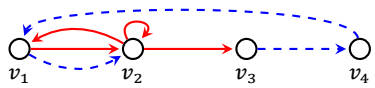

Fig. 10. Two-dimensional digraphs corresponding to $kernel[6]$

$$\underbrace{\begin{bmatrix}0&1&0&0\\1&1&0&0\\0&1&0&0\\0&0&0&0\end{bmatrix}}_{\mathbf{A}_1},\quad \underbrace{\begin{bmatrix}0&0&0&1\\1&0&0&0\\0&0&0&0\\0&0&1&0\end{bmatrix}}_{\mathbf{A}_2},\quad \underbrace{\begin{bmatrix}0&1&0&1\\2&1&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}}_{\mathbf{P}}, \tag{20}$$

$$cycles = [1\,1\,0\,1], \quad intersection = [0\,1\,0\,0].$$

In this situation, we have common part of monomial digraphs (vertex v_2), and we can assume that $kernel[6]$ is one of possible realisations of the characteristic polynomial.

In this same way, we determine the remaining kernels. In our example, we have $kernel[1], \ldots, kernel[48]$.

Using the second part of the function $createSolutionKernel()$ for all kernels which have common part, we check the number of n-vertex cycles in a characteristic polynomial realisation. At the beginning, we check, number of cycles consisting of one vertices, then two vertices, three vertices and four vertices respectively. In our example, we will check the number of cycles for:

1) Digraphs corresponding to $kernel[1]$ are presented in Figure 8. Matrix $\mathbf{P}$ and vector $cycles$ are described by the equation (18).
   - Check condition for simple cycle consisting of one vertex – $\sum_{i=1}^{size} p_{i,i} = \sum_{i=1}^{4} p_{i,i} = 1 = cycles[1]$;
   - Check condition for simple cycle consisting of two vertices (see Function 6 lines 7-9).

$$\underbrace{\begin{bmatrix}1&1\\1&0\end{bmatrix}}_{Q_{1,2}} = 1;\ \underbrace{\begin{bmatrix}1&0\\0&0\end{bmatrix}}_{Q_{1,3}} = 0;\ \underbrace{\begin{bmatrix}1&1\\0&0\end{bmatrix}}_{Q_{1,4}} = 0;$$
$$\underbrace{\begin{bmatrix}0&0\\1&0\end{bmatrix}}_{Q_{2,3}} = 0;\ \underbrace{\begin{bmatrix}0&0\\0&0\end{bmatrix}}_{Q_{2,4}} = 0;\ \underbrace{\begin{bmatrix}0&0\\1&0\end{bmatrix}}_{Q_{3,4}} = 0; \tag{21}$$
$$cycles[2] = 1+0+0+0+0+0 = 1$$

   - Check condition for simple cycle consisting of three vertices (see Function 6 lines 14-18).

$$\underbrace{\begin{bmatrix}0&0&0\\1&0&0\\0&1&0\end{bmatrix}}_{Q_1} = 0,\ \underbrace{\begin{bmatrix}1&0&1\\0&0&0\\0&1&0\end{bmatrix}}_{Q_2} = 0,\ \underbrace{\begin{bmatrix}1&2&1\\1&0&0\\0&0&0\end{bmatrix}}_{Q_3} = 0,\ \underbrace{\begin{bmatrix}1&2&0\\1&0&0\\0&1&0\end{bmatrix}}_{Q_4} = 0$$

   - Check condition for simple cycle consisting of four vertices (see Function 6 lines 7-9).

$$Q = \mathbf{P} = \begin{bmatrix}1&2&0&1\\1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix} = 1, \quad cycles[4] = 1$$

At this moment, we can say that digraphs presented in Figure 8 do not appear addition cycles.

2) Digraphs corresponding to $kernel[6]$ are presented in Figure 10. Matrix **P** and vector $cycles$ are described by the equation (20).
   - Check condition for simple cycle consisting of one vertex – $\sum_{i=1}^{size} p_{i,i} = \sum_{i=1}^{4} p_{i,i} = 1 = cycles[1]$;
   - Check condition for simple cycle consisting of two vertices (see Function 6 lines 7-9).

$$\underbrace{\begin{bmatrix}0&1\\2&1\end{bmatrix}}_{Q_{1,2}} = 2;\ \underbrace{\begin{bmatrix}0&0\\0&0\end{bmatrix}}_{Q_{1,3}} = 0;\ \underbrace{\begin{bmatrix}0&1\\0&0\end{bmatrix}}_{Q_{1,4}} = 0;$$

$$\underbrace{\begin{bmatrix}1&0\\1&0\end{bmatrix}}_{Q_{2,3}} = 0;\ \underbrace{\begin{bmatrix}1&0\\0&0\end{bmatrix}}_{Q_{2,4}} = 0;\ \underbrace{\begin{bmatrix}0&0\\1&0\end{bmatrix}}_{Q_{3,4}} = 0; \qquad (22)$$

$$cycles[2] = 2 + 0 + 0 + 0 + 0 + 0 = 2$$

This means that in digraph corresponding to $kernel[6]$ (see Figure 10) additional cycle consisting of two vertices appears, and digraph does not satisfy the characteristic polynomial (14).

6. Concluding Remarks

The paper presents in-depth examination of the parallel computer algorithm for finding a set of characteristic polynomial realisations of dynamic system. The algorithm is based on the multi-dimensional digraphs theory to allow creation of a set of solutions of characteristic polynomial realisations instead of just one solution – this is what makes the main difference between the proposed method and other state-of-the-art solutions. Moreover, the presented algorithm created solutions that are always minimal in terms of size of the state matrices. The algorithm gives as a result a set of realisations of characteristic polynomial, both in the form of digraphs and state matrices representation. Due to NP-hard complexity of the problem, as digraphs-building methods used in the algorithm are NP-complete or NP-hard problems, as presented in Section 4, the algorithm needs to be paralleled using GPGPU (General-Purpose computing on Graphics Processor Units) computation as it is impossible to sequentially check all needed solutions. Even for parallel implementation the method is still time consuming and of factorial computational complexity. The paper presents optimisation of the algorithm, based on mathematical principles of circular multi-set permutations, that eliminates the problem with the worst-case scenario of the basic version of the algorithm, reduces a number of unnecessary created potential solutions (and thus complexity and working time of the algorithm) both in growth and prune steps, allows for parallel execution of both growth and prune steps and simplifies the algorithm workings for 1D monomials and polynomials. Proposed modifications greatly reduce time needed for candidate generation.

Further work, apart from optimisation, includes extension of the algorithm to find all possible solutions (not only minimal), solve the realisation problem, reachability and controllability of systems using the fast graph-based method. There is also an open problem of the analysis of system dynamics for realisations on a different number of nodes in digraphs.

AUTHORS

Krzysztof Hryniów – Faculty of Electrical Engineering, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland, e-mail: Krzysztof.Hryniow@ee.pw.edu.pl.

Konrad Andrzej Markowski* – Faculty of Electrical Engineering, Warsaw University of Technology, Koszykowa 75, 00-662 Warsaw, Poland, e-mail: Konrad.Markowski@ee.pw.edu.pl, www: http://nas.isep.pw.edu.pl.

*Corresponding author

REFERENCES

[1] J. Bang-Jensen, G. Gutin, *Digraphs: Theory, Algorithms and Applications (2nd Edition)*, Springer-Verlag: London, 2009.

[2] L. Benvenuti, L. Farina, "A tutorial on the positive realization problem", *IEEE Transactions on Automatic Control*, vol. 49, no. 5, 2004, 651–664.

[3] A. Berman, M. Neumann, R. J. Stern, *Nonnegative Matrices in Dynamic Systems*, Wiley: New York, 1989.

[4] M. Bisiacco, E. Fornasini, G. Marchesini, "Dynamic regulation of 2D systems: A state-space approach", *Linear Algebra and Its Applications*, vol. 122–124, 1989, 195–218.

[5] T. Blyth, E. Robertson, *Basic Linear Algebra (2nd Edition)*, Springer: London, 2002.

[6] M. Bona, *Combinatorics of Permutations(Second Edition)*, Chapman Hall, CRC Press, 2012.

[7] R. Bru, C. Coll, S. Romero, E. Sanchez, "Reachability indices of positive linear systems", *Electronic Journal of Linear Algebra*, vol. 11, 2004, 88–102.

[8] R. Bru, S. Romero-Vivo, E. Sanchez, "Reachability indices od periodic positive systems via positive shift-similarity", *Linear Algebra and Its Applications*, vol. 429, 2008, 1288–1301.

[9] E. Fornasini, G. Marchesini, "State-space realization theory of two-dimensional filters", *IEEE Trans, Autom. Contr.*, vol. 21, 1976, 481–491.

[10] E. Fornasini, M. E. Valcher, "Directed graphs, 2D state models, and characteristic polynomials of irreducible matrix pairs", *Linear Algebra and Its Applications*, vol. 263, 1997, 275–310.

[11] E. Fornasini, M. E. Valcher, "On the positive reachability of 2D positive systems", *LCNIS*, 2003, 297–304.

[12] E. Fornasini, M. E. Valcher, "Controllability and reachability of 2D positive systems: a graph theoretic approach", *IEEE Transaction on Circuits and Systems I*, no. 52, 2005, 576–585.

[13] L. R. Foulds, *Graph Theory Applications*, Springer Verlag, 1991.

[14] C. Godsil, G. Royle, *Algebraic Graph Theory*, Springer Verlag, 2001.

[15] S. Hong, T. Oguntebi, K. Olukotun, "Efficient parallel graph exploration on multi-core CPU and GPU". In: *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, 2011, 78–88.

[16] R. A. Horn, C. R. Johnson, *Topics in Matrix Analysis*, Cambridge Univ. Press, 1991.

[17] K. Hryniów, "Parallel pattern mining on Graphics Processing Units". In: *Proceedings of 2013 14th International Carpathian Control Conference (ICCC)*, 2013, 134–139.

[18] K. Hryniów, K. A. Markowski, "Conditions for digraphs representation of the characteristic polynomial". In: *Young Scientists Towards the Challenges of Modern Technology*, 2014, 77–80.

[19] K. Hryniów, K. A. Markowski, "Parallel digraphs-building algorithm for polynomial realisations". In: *Proceedings of 2014 15th International Carpathian Control Conference (ICCC)*, 2014, 174–179.

[20] K. Hryniów, K. A. Markowski, "Reachability index calculation by parallel digraphs-building". In: *19th International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland, September 2-5, 2014*, 2014, 808–813.

[21] K. Hryniów, K. A. Markowski, "Digraphs-building of complete set of minimal characteristic polynomial realisations as means for solving minimal realisation problem of nD systems", *International Journal of Control*, 2015, (Submitted to).

[22] K. Hryniów, K. A. Markowski, "Optimisation of digraphs creation for parallel algorithm for finding a complete set of solutions of characteristic polynomial". In: *20th International Conference on Methods and Models in Automation and Robotics (MMAR), 2015*, 2015, 1139–1144.

[23] K. Hryniów, K. A. Markowski. "Classes of digraph structures corresponding to characteristic polynomials". In: R. Szewczyk, C. Zieliński, and M. Kaliczyńska, eds., *Challenges in Automation, Robotics and Measurement Techniques*, Advances in Intelligent Systems and Computing, 329–339. Springer International Publishing, 2015.

[24] K. Hryniów, K. A. Markowski. "Optimisation of digraphs-based realisations for polynomials of one and two variables". In: R. Szewczyk, C. Zieliński, and M. Kaliczyńska, eds., *Progress in Automation, Robotics and Measuring Techniques*, volume 350 of *Advances in Intelligent Systems and Computing*, 73–83. Springer International Publishing, 2015.

[25] R. Impagliazzo, R. Paturi, "On the complexity of k-SAT", *Journal of Computer and System Sciences*, vol. 62, 2001, 367–375.

[26] T. Kaczorek, "Realization problem for general model of two-dimensional linear systems", *Bulletin of the Polish Academy of Sciences, Technical Sciences*, vol. 35, no. 11–12, 1987, 633–637.

[27] T. Kaczorek, *Positive 1D and 2D systems*, Springer Verlag: London, 2001.

[28] T. Kaczorek, "Positive realization for 2D systems with delays". In: *Proceedings of 2007 International Workshop on Multidimensional (nD) Systems*, 2007, 137 – 141.

[29] T. Kaczorek, "Positive realization of 2D general model", *Logistyka*, vol. nr 3, 2007, 1–13.

[30] T. Kaczorek, M. Busłowicz, "Minimal realization problem for positive multivariable linear systems with delay", *Int. J. Appl. Math. Comput. Sci.*, no. 14(2), 2004, 181 – 187.

[31] T. Kaczorek, L. Sajewski, *The Realization Problem for Positive and Fractional Systems*, Springer International Publishing: Berlin, 2014.

[32] D. Luebke and G. Humphreys, "How GPUs work", *IEEE Computer*, vol. 40, 2007, 96–100.

[33] K. A. Markowski, "Determination of positive realization of two dimensional systems using digraph theory and GPU computing method". In: *International Symposiumon Theoretical Electrical Engineering, 24th – 26th June 2013: Pilsen, Czech Republic*, 2013, II7–II8.

[34] K. A. Markowski, "Determination of Reachability Index Set of Positive 2D System Using Digraph Theory and GPU Computing Method". In: *18th International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland, August 26-29, 2013*, 2013, 705–710.

[35] K. A. Markowski, "Determination of minimal realisation of one-dimensional continuous-time fractional linear system", *International Journal of Dynamics and Control*, 2016 (Accepted).

[36] W. D. Wallis, *A Beginner's Guide to Graph Theory*, Biiokhäuser, 2007.

[37] L. Xu, H. Fan, Z. Lin, N. Bose, "A direct-construction approach to multidimensional realization and LFR uncertainty modeling", *Multidimensional Systems and Signal Processing*, vol. 19, no. 3–4, 2008, 323–359.

[38] L. Xu, L. Wu, Q. Wu, Z. Lin, Y. Xiao, "Reduced-order realization of Fornasini-Marchesini model for 2D systems". In: *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 3, 2004, III–289–292.

[39] L. Xu, L. Wu, Q. Wu, Z. Lin, Y. Xiao, "On realization of 2D discrete systems by Fornasini-Marchesini model", *International Journal of Control, Automation, and Systems*, vol. 4, no. 3, 2005, 631–639.

[40] L. Xu, Q. Wu, Z. Lin, Y. Xiao, Y. Anazawa, "Futher results on realisation of 2D filters by Fornasini-Marchesini model". In: *8th International Conference on Control, Automation, Robotics and Vision, Kunming, China, 6-9th December*, 2004, 1460–1464.