SOFTWARE SOLUTION OF THE ALGORITHM OF THE CYCLIC-ORDER GRAPH

Štefan BEREŽNÝ*,**, Ján BUŠA Jr.**,***, Michal STAŠ*
*Department of Mathematics and Theoretical Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Němcovej 32, 042 00 Košice, Slovak Republic, Tel.: +421 55 602 2447, E-mail: stefan.berezny@tuke.sk; michal.stas@tuke.sk
** LIT, Joint Institute for Nuclear Research, 141980 Dubna, Moscow Region, Russia
*** IEP, Slovak Academy of Sciences, Košice, Slovak Republic, E-mail: busa@jinr.ru

ABSTRACT

In this paper we describe by pseudo-code the "Algorithm of the cyclic-order graph", which we programmed in MATLAB 2016a and which is also possible to be executed in GNU Octave. We describe program's functionality and its use. The program implementing this algorithm is an indispensable tool during proofs in the field of graph theory, especially when dealing with crossing numbers for join products of graphs with paths of given numbers of vertices.

Keywords: *crossing number, cyclic permutations, algorithm, graph, distances in graph*

1. INTRODUCTION

In [1] was shown the proof of the values of crossing numbers for join products of the graph G on five vertices with the discrete graph D_n and the path P_n on n vertices. The proof was done with the algorithm that is described in this article. The algorithm generates all cyclic permutations for a given number n. For cyclic permutations from P_1 up to P_m (m being the number of cyclic permutations) it creates a graph in which it calculates the distances between all vertices of the graph. These are used in proof of crossing numbers for presented graphs.

In article [1] was shown the correct proof of the theorem from [2]. Using the given algorithm for $n = 5$, we got the names of the cyclic permutations and the distance between the cyclic permutations. The maximum distance between two vertices in such graph is equal to four. With this information we were able to use arguments described in [3,4].

2. MATHEMATICAL BACKGROUND

For the algorithm we will use few facts from the graph theory:
Let B be an adjacency matrix corresponding to the graph G with m elements and let $B^{(1)}$ be the matrix obtained from adjacency matrix B by adding ones to the main diagonal. Let us consider the matrix $B^{(2)} = \{b_{ij}^{(2)}\}_{i,j=1}^{m}$, such that

$$B^{(2)} = B^{(1)} \cdot B^{(1)}. \tag{1}$$

From the matrix multiplication it is obvious that

$$b_{ij}^{(2)} = \sum_{k=1}^{m} b_{ik}^{(1)} \cdot b_{kj}^{(1)}, \tag{2}$$

but in this matrix we use the Boolean addition and multiplication ($1 \cdot 1 = 1$, $0 \cdot 1 = 1 \cdot 0 = 0 \cdot 0 = 0$, $1 + 0 = 0 + 1 = 1 + 1 = 1$, and $0 + 0 = 0$). In general we consider matrix:

$$B^{(m)} = B^{(m-1)} \cdot B^{(1)}. \tag{3}$$

Theorem 2.1. *Let B be the adjacency matrix of the connected graph $G = (V,H)$, $|V| = m$ is given. Then for arbitrary $k = 1,2,\ldots,m$, the element $b_{ij}^{(k)}$ of the matrix $B^{(k)}$ is equal to one if $d(v_i,v_j) \leq k$, where $d(v_i,v_j)$ is a distance between two vertices v_i and v_j, respectively.*

Corollary 2.1. *The graph $G = (V,H)$, $|V| = m$, is connected only if all elements of the matrix $B^{(m-1)}$ are equal to one.*

Corollary 2.2. *For each two different vertices of the graph $G = (V,H)$, $d(v_i,v_j) = \min_{k\in\{1,2,\ldots,m\}}\{k; b_{ij}^{(k)} = 1\}$.*

3. ALGORITHM'S OVERVIEW

The input for the algorithm is the number n, which represents an n-element set $\{1,2,3,\ldots,n\}$. The algorithm selects all *cyclic* permutations from the set of all permutations of the n-element set $\{1,2,3,\ldots,n\}$ (calculates or reads precalculated from input). It then marks these permutations with symbols $P_1,\ldots,P_m$, where $m = (n-1)!$. These permutations correspond to the vertices $V = \{P_1,P_2,\ldots,P_m\}$ of the graph G which we study. In this graph an edge is created between two vertices, if they correspond to the permutations P_i and P_j, which can be transformed from one to other by interchanging of exactly two elements of the n-tuple (i. e. an ordered set with n elements). This graph is represented by a square symmetrical adjacency matrix. The distance between each pair of vertices is calculated using the properties of the cyclic-order graph CO_5 defined in [5].

For algorithm to run, user has to choose proper number of elements n. n equaling to 2 is a minimal possible value. As to the upper boundary, on recent hardware it is possible to use n up to 10. Since the complexity of the adjacency matrix grows with the factor of n^2, changes in algorithm's data storage and matrix multiplication would be necessary to account for this computational increase.

4. DESCRIPTION OF THE ALGORITHM

This algorithm consists of the main procedure and four functions. Input is from keyboard and defined files. Output is to the screen. It uses the basics from [6]. All functions are written in such a manner, that after putting them in appropriate `*.m` files they can be run without any change (except for commenting out comments). This approach forced us to be sometimes more verbal but the provided code is better readable and easier to use.

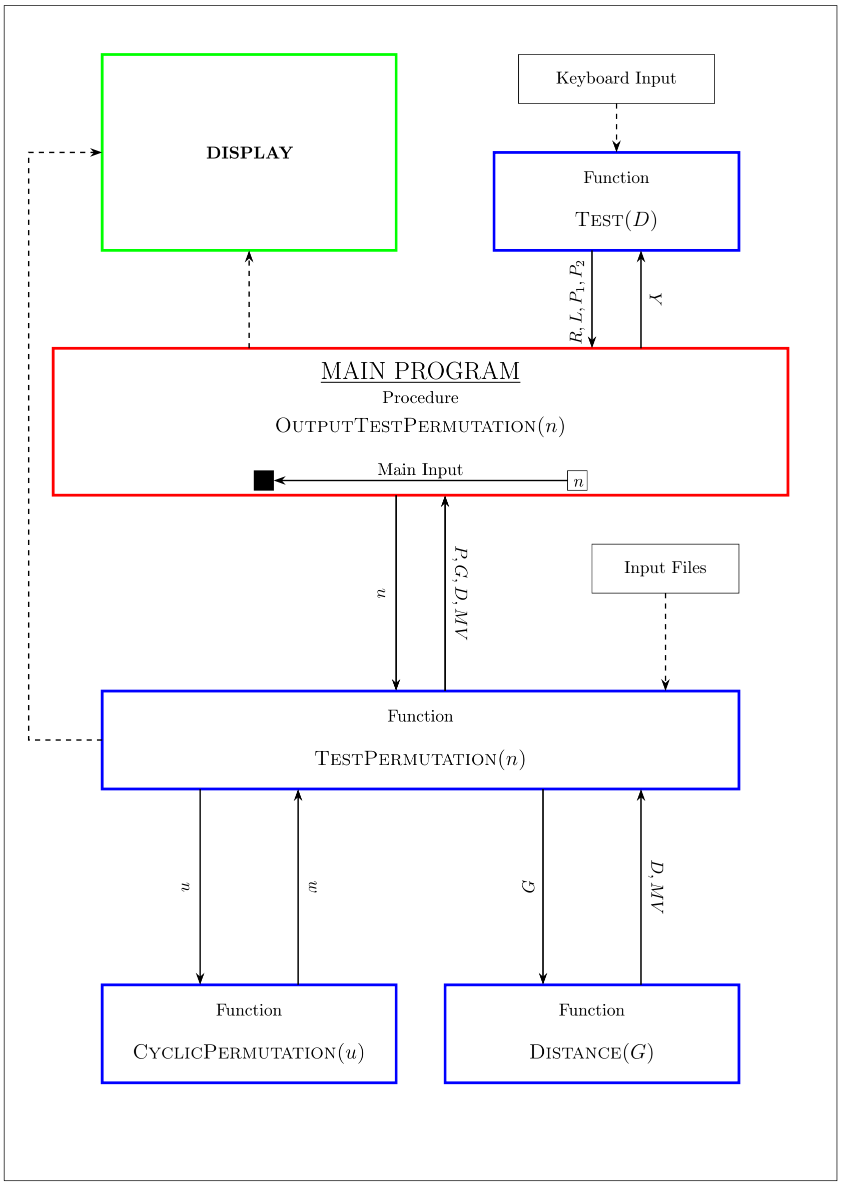

Fig. 1 Scheme of the algorithm

For schematic diagram of the Algorithm of the cyclic-order graph see Fig. 1. There are displayed procedures and functions that were used, their mutual relations, inputs, and outputs. The main algorithm starts by using procedure OUTPUTTESTPERMUTATION(n) (see Algorithm 5.1), where n is an positive integer between 2 and 10, respectively. This procedure calls two functions TESTPERMUTATION(n) (see Algorithm 5.2) and TEST(D) (see Algorithm 5.3).

The function TESTPERMUTATION(n), based on input n, loads a list of cyclic permutations P_i from the appropriate file. These lists of cyclic permutations are stored in files, because in MATLAB we were not able to guarantee the same ordering of permutations for subsequent calls to permutations function. This is not acceptable, since the proofs of theorems in graph theory require the labeling of permutations to be constant. It also generates a graph of cyclic permutations, which is represented by the adjacency matrix G.

Here it is necessary to perform the test, which vertices of the graph G are to be joined by the edge and which are not. The function TESTPERMUTATION(n) calls CYCLICPERMUTATION(u) (see Algorithm 5.4), which has an input value a permutation u and as output gives the cyclic permutation w to which permutation u corresponds. Based on this output we can assign label to permutation P_i. Similarly, function TESTPERMUTATION(n) calls DISTANCE(G) (see Algorithm 5.5), which has as input an adjacency matrix G and the output is matrix of distances D and the maximal distance MV in the graph G. The matrix D contains at position ij (i-th row and j-th column) the element d_{ij} corresponding to the distance between the vertices v_i and v_j in the graph G, where the vertex v_i corresponds to the cyclic permutation P_i and the vertex v_j corresponds to the cyclic permutation P_j.

All important information about the generated graph G is sent by the TESTPERMUTATION(n) to the main procedure OUTPUTTESTPERMUTATION(n), namely:

P	– list of cyclic permutations with their designation P_i,
G	– adjacency matrix of the graph G,
D	– distance matrix of the graph G,
MV	– maximum distance in the graph G (graph diameter).

The function TEST(Y) is a function that gives the output of specific information about the distance of two selected cyclic permutations. The input for TEST(Y) is a matrix Y corresponding to matrix D. Permutations P_i and P_j and the natural number $L \geq 1$ are entered by keyboard. The output will be a vector R – list of permutations P_k that satisfy the conditions:

(1) distance(P_i, P_k) + distance(P_k, P_j) $< L$,

(2) distance(P_i, P_k) > 0 and distance(P_k, P_j) > 0.

All calculated information is displayed on the monitor screen.

5. ALGORITHM OF THE CYCLIC-ORDER GRAPH – PSEUDOCODE

Algorithm 5.1 Calculate: The distances between cyclical permutations.

Require: $n > 0$ an positive integer, $d > 0$ an positive integer, serial numbers of two vertices of the graph G

Ensure: List of cyclic permutations, adjacency matrix of the graph G, maximum distance in graph G, distance between each pairs of vertices of the graph G, set of vertices whose sum of distances from the two given vertices is less than or equal to the given constant d

```
1:  procedure OUTPUTTESTPERMUTATION($n$)                              ▷ Main procedure.
2:                                              ▷ It gives at the output the required values.
3:      [P, G, D, MV] = TESTPERMUTACION($n$);                          ▷ See line 101
4:      $s = 1$;                                                       ▷ Input variable.
5:      display('Menu for output:')
6:      while $s > 0$ do
7:          display('Choose one of the options:')
8:          display('0 = End the program')
9:          display('1 = Listing of adjacency matrix of the graph $G$')
10:         display('2 = Listing of cyclic permutations (vertices of the graph $G$)')
11:         display('3 = Listing the maximum distance in the graph $G$')
12:         display('4 = List all distances between the vertices of the graph $G$')
13:         display('5 = List of all vertices that satisfy the defined inequality in the graph $G$')
14:         $s$ = input('Enter a number: ');                           ▷ Expected Keyboard Value
15:         switch $s$ do
16:             case 0
17:                 display('The program is terminated.')
18:             end case
19:             case 1
20:                 display('The adjacency matrix of the graph $G$ has the form:')
21:                 display($G$)
22:             end case
23:             case 2
24:                 display('Label of permutation P:')
25:                 [$u$, $v$] = size(P);
26:                 $Z$ = [];
27:                 for $i$ = 1..$u$ do
28:                     $Z$ = ['Permutation P', $i$, ' = ', '('];
29:                     for $j$ = 2..$v$ do
30:                         $Z$ = [Z, ' ', P($i$, $j$)];
31:                     end for
32:                     $Z$ = [Z, ' )'];
33:                     display($Z$)
```

```
34:         end for
35:       end case
36:       case 3
37:         $Output$ = ['The maximum distance in the graph $G$ = ', $MV$];
38:         display($Output$)
39:       end case
40:       case 4
41:         for $k = 1..MV$ do
42:           $X$ = ['Permutations at distance d = ', $k$];
43:           display($X$)
44:           $Y$ = [];
45:           for $i = 1..m$ do
46:             $Y$ = ['Permutation ', ' P', $i$,' $-->$ ',' Permutations ', ' P:'];
47:             for $j = 1..n$ do
48:               if $D(i, j) == k$ then
49:                 $Y$ = [$Y$, ' ', $j$,', '];
50:               end if
51:             end for
52:             display($Y$)
53:             $Y$ = [];
54:           end for
55:           $X$ = [];
56:         end for
57:       end case
58:       case 5
59:         [$R, L, P1, P2$] = TEST($D$);                                   ▷ See line 251
60:         $Text$ = ['Permutation P$_k$, having the sum of the distances not greater than ', $L$,
61:         ' from permutation P', $P1$, ' and P', $P2$, ' are: '];
62:         display($Text$)
63:         display(['P$_k$ $-->$ ', $R$])
64:       end case
65:       case Otherwise
66:         display('Such a choice is not between the options.
67:         Select option from the list.')
68:       end case
69:     end switch
70:   end while
71:   display('END OF PROGRAM!')
72: end procedure
```

Algorithm 5.2 PART – Function: TESTPERMUTATION(n) (continued)

```
101: function [$P, G, D, MV$] = TESTPERMUTATION($n$)

102:   Test whether the input values are correct:
103:   clear $P, N, G, D, Z$;
104:   if or($n <= 1$, $n > 10$) then
105:     display('Entered number must be an integer value from 2 to 10!');
106:     $P$ = [];
107:     return ;
108:   else if round($n$) $\sim=$ $n$ then
109:     display('The value $n$ must be integer!');
110:     $P = 0$;
111:     quit cancel;
112:   else
113:     clear $A, AC$;
114:   end if
115:   switch $n$ do
116:     case {2, 3}
117:       $N$ = [1:$n$]; $N1$ = [2:$n$];
```

```
      $A$ = **perms**($N$);
      $AC$ = **perms**($N1$);
      $[m1, n1]$ = **size**($AC$);
      $AC$ = [$AC$ **ones**($m1, 1$)];
      $[m, n]$ = **size**($A$);
    **end case**
    **case** 4
      $filename$ = 'Permutacia-4.mat';
      **load**($filename$)
    **end case**
    **case** 5
      $filename$ = 'Permutacia-5.mat';
      **load**($filename$)
    **end case**
    **case** 6
      $filename$ = 'Permutacia-6.mat';
      **load**($filename$)
    **end case**
    **case** 7
      $filename$ = 'Permutacia-7.mat';
      **load**($filename$)
    **end case**
    **case** 8
      $filename$ = 'Permutacia-8.mat';
      **load**($filename$)
    **end case**
    **case** 9
      $filename$ = 'Permutacia-9.mat';
      **load**($filename$)
    **end case**
    **case** 10
      $filename$ = 'Permutacia-10.mat';
      **load**($filename$)
    **end case**
    **case** Otherwise
      **warning**('For such values of $n$ this program doesn't work.')
    **end case**
  **end switch**
  $G$ = **uint8**(**zeros**($m1$));
  **for** $k = 1..m$ **do**
    **for** $h = (k+1)..m$ **do**
      $v = A(k,:) - A(h,:)$;
      $s$ = **sum**($v$);
      $nuls = 0$;
      **for** $j = 1..n$ **do**
        **if** $v(j) == 0$ **then**
          $nuls = nuls + 1$;
        **end if**
      **end for**
      **if and**($s == 0, nuls == (n-2)$) **then**
        **for** $i = 1..(n-1)$ **do**
          **if** $(v(i) * v(i+1) == 0)$ **then**
            $s = s + (v(i) * v(i+1))$;
          **else**
            $s = s + (v(i) * (1/v(i+1)))$;
          **end if**
        **end for**
        **if** $(v(n) * v(1) == 0)$ **then**
          $s = s + (v(n) * v(1))$;
        **else**
```

```
                $s = s + (v(n) * (1/v(1)));$
            end if
            if $s == -1$ then
                for $j = 1..m1$ do
                    $[w]$ = CyclicPermutation(A(k,:));    ▷ See line 301
                    if $w == AC(j,:)$ then
                        $k1 = j;$
                    end if
                    $[w]$ = CyclicPermutation(A(h,:));    ▷ See line 301
                    if $w == AC(j,:)$ then
                        $h1 = j;$
                    end if
                end for
                $G(k1,h1) = 1;$
                $G(h1,k1) = 1;$
            end if
        end if
    end for
  end for
  $T = [[1:m1]'AC];$
  clear $ACO$;
  $ACO$ = zeros$(m1, n1+1);$
  for $j = 1..(n1+1)$ do
      $ACO(:,j) = AC(:,n1+2-j);$
  end for
  $Z = [];$
  for $j = 1..m1$ do
      $x = ACO(j,:);$
      $Z(j,:) = [j,x];$
  end for
  $P = Z;$
  $[D,MV]$ = Distance($(G)$);    ▷ See line 351
  return $P, G, D, MV$
end function
```

Algorithm 5.3 PART – Function: Test(Y) (continued)

```
function $[R, L, P1, P2]$ = Test($Y$)

  The input for the Test function is a matrix $Y$ of distances between each pair of permutations.
  Permutations $P_i$ and $P_j$ and the natural number $L \geq 1$ are then entered.
  The output will be a list of permutations $P_k$ that satisfy the condition:
  distance($P_i$, $P_k$) + distance($P_k$, $P_j$) $< L$,
  distance($P_i$,$P_k$) $> 0$ and distance($P_k$,$P_j$) $> 0$.
  clear $P1, P2, L, T, cifra$;
  $[m,\sim]$ = size$(Y)$;
  $R = [];$
  $cifra = 0;$
  while $cifra == 0$ do
      display('Input:')
      display('The numbers $i$, $j$ and $L$ must be an positive integers and
      number $i$ must be different from number $j$!')
      display(['Numbers $i$ and $j$ must be in the range from 1 to', $m$)])
      $P1$ = input('Choose the permutation $P_i$ as the order number $i$ = ');
      $P2$ = input('Choose the permutation $P_j$ as the order number $j$ = ');
      L = input('Select the limit value $L$ = ');
      if or(or($P1 < 0$, $P2 < 0$), $L < 0$) then
          display('At least one of the input values is a negative.')
      else if or(or($P1 == 0$, $P2 == 0$), $L == 0$) then
          display('At least one of the input values is zero.')
```

```
        **else**
            **if or**(**or**($P1 \; = $ **round**($P1$), $P2 \; = $ **round**($P2$)), $L \; = $ **round**($L$)) **then**
                **display**('At least one of the input values is non-integer.')
            **else if or**($P1 > m$, $P2 > m$) **then**
                **display**('At least one value $i$, rep. $j$ is not from the input interval.')
            **else**
                **if** $P1 == P2$ **then**
                    **display**('Values $i$ and $j$ must be different.')
                **else**
                    $cifra = 1$;
                **end if**
            **end if**
        **end if**
    **end while**
    **for** $k$ = $1..m$ **do**
        **if and**($k \; = P1$, $k \; = P2$) **then**
            $T = Y(P1,k) + Y(k,P2)$;
            **if** $T <= L$ **then**
                $R = [R,k]$;
            **end if**
        **end if**
    **end for**
  **return** $R, L, P1, P2$
**end function**
```

Algorithm 5.4 PART – Function: CYCLICPERMUTATION(u) (continued)

```
**function** [$w$] = CYCLICPERMUTATION($u$)

    [$\sim$, $n$] = **size**($u$);
    $w = u$;
    **while** $w(n) \; = 1$ **do**
        $a = w(1)$;
        **for** $i$ = $1..(n-1)$ **do**
            $w(i) = w(i+1)$;
        **end for**
        $w(n) = a$;
    **end while**
    **return** $w$
**end function**
```

Algorithm 5.5 PART – Function: DISTANCE(G) (continued)

```
**function** [$D$, $MV$] = DISTANCE($G$)

    This function calculates the distance in the graph G between all its vertices.
    **clear** $nula, D, MV$;
    [$\sim$,$n$] = **size**($G$);
    $D = G$;
    $MV = 1$;
    $k = 2$;
    $nula = 0$;
    $H(:,:,1) = G$ + **uint8**(**diag**(**ones**($1,n$)));
    $H(:,:,1)$ = **logical**($H(:,:,1)$);
    $H(:,:,2) = H(:,:,1)$;
    **while or**($nula == 0$, $k == n$) **do**
        $nula = 1$;
        **for** $i$ = $1..n$ **do**
            **for** $j$ = $1..n$ **do**
                $z = []$;
                $z$ = **any**(**and**($H(i,:,2)'$, $H(:,j,1)$));
                $H(i,j,3) = \mathbf{max}(z)$;
                if $\mathbf{max}(z) == 0$ then
                    $nula = 0$;
                end if
                if and$(H(i,j,2) == 0, H(i,j,3) == 1)$ then
                    $D(i,j) = k$;
                end if
            end for
        end for
        $H(:,:,2) = H(:,:,3)$;
        $MV = k$;
        $k = k+1$;
    end while
    return $D$, $MV$
end function
```

6. CONCLUSIONS

In this article, we have shown the algorithm, which is used to help to prove results on crossing numbers in graph theory and its implementation in MATLAB. More significant usage of this algorithm occurs for values of n larger than five. We get 120 cyclic permutations for $n = 6$ or 720 for $n = 7$, which is significantly more than 24 cyclic permutations for $n = 5$ used in article [1]. For such values, software is an indispensable tool since, we get considerably more complicated graph of distances between cyclic permutations and calculating distances for such large graphs by hand can lead to many calculation errors.

Even though use of computer program speeds up the calculation, as the problem grows factorially. This opens new questions for further investigation, namely optimal storage of adjacency matrix and parallel calculation of adjacency matrix either using MPI or some GPU based techniques. While the multiplication itself can be easily parallelized, organization of storage and operations can be challenging.

ACKNOWLEDGEMENT

The research was supported by the Slovak VEGA grant No. 1/0389/15. The research was also supported by the internal faculty research project No. FEI-2017-39.

REFERENCES

[1] BEREŽNÝ, Š. – STAŠ, M.: *On the crossing number of the join of five vertex graph G with the discrete graph D_n*, In: Acta Electrotechnica et Informatica, No. 3, Vol. 17, 2017, pp. 27-32, ISSN 1335-8243.

[2] LI, B. – WANG, J. – HUANG, Y.: *On the crossing number of the join of some 5-vertex graphs and P_n*, In: International J. Math. Combin., Vol. 2, 2008, pp. 70-77.

[3] HERNÁNDEZ-VÉLEZ, C. – MEDINA, C. – SALAZAR, G.: *The optimal drawing of $K_{5,n}$*, In: Electron. J. Combin., No. 4, Vol. 21, 2014, pp. 29, paper 4.1.

[4] ZARANKIEWICZ, K.: *On a problem of P. Turán concerning graphs*, In: Fund. Math., Vol. 41, 1955, pp. 137-145.

[5] WOODALL, D. R.: *Cyclic-order graphs and Zarankiewicz's crossing number conjecture*, In: Journal of Graph Theory, No. 6, Vol. 17, 1993, pp. 657-671.

[6] BEREŽNÝ, Š.: *Laboratories for teaching of mathematical subjects*, In: Acta Didactica Napocensia, No. 1, Vol. 10, 2017, pp. 35-46.

Received October 30, 2017, accepted January 16, 2018

BIOGRAPHIES

Štefan Berežný was born in 1974. In 1998 he graduated (MSc) at the Faculty of Science, P.J. Šafárik University in Košice in the field of discrete mathematics. He defended his PhD in the field of mathematical optimization in 2005; his thesis title was "Optimization problems on graphs with categorization of edges". Since 1998 he is working as a assistant professor at the Department of Mathematics and Physics on the Air Force Academy in Košice. His scientific research is focusing on mathematical optimization on graphs, complexity of algorithms, and applied mathematics. In 2004 he went to the Department of Mathematics FEEI TUKE, where he is still a lecturer at the Department of Mathematics and Theoretical Informatics FEEI TUKE. In addition, he also investigates questions related to the use of mathematical software in teaching mathematics courses and appropriate creating of computer laboratories at the department.

Ján Buša Jr. was born in 1981. In 2005 he graduated at the FMPI at Comenius University in Bratislava, Slovakia and in 2009 he defended his PhD from Numerical Mathematics in Ghent University in Ghent, Belgium. After working at DMTI FEEI TUKE, he is now member of IEF SAS Košice, currently working at Laboratory of Information Technologies at Joint Institute of Nuclear Research in Dubna, Russia.

Michal Staš was born in 1983. In 2006 he graduated (MSc) at the Faculty of Science, P.J. Šafárik University in Košice in the field of discrete mathematics. He defended his PhD in the field of discrete mathematics - set theory in 2008; his thesis title was "Optimization problems on graphs with categorization of edges". Since 2010 he is working as a assistant professor at the Department of Mathematics and Theoretical Informatics FEEI TUKE. His scientific research is focusing on mathematical optimization on graphs, complexity of algorithms, and applied mathematics.