International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-1, November 2019

Visual Analysis of Actions Performed with Big Graphs

Velin Spasov Kralev, Radoslava Stankova Kraleva

Abstract*: The basic concepts of using application development environments are presented in this paper. The way of using the GraphAnalyser application and its basic functions is also presented. All results of the experiments conducted are generated with this application. According to the experimental methodology, they fall into two groups: the first one includes actions related to the vertices of a graph, and the second one includes actions related to the edges and the dynamic allocation of memory to store the structure of a graph. The results show that when the number of vertices in a graph increases linearly, the time to add and remove these vertices also increases linearly. When the number of graph vertices increases linearly, the number of added vertices per millisecond remains relatively constant. However, the number of vertices removed for one millisecond for graphs containing between 10 and 70 million vertices varies. Similarly, when the number of graph edges increases linearly, the number of added edges per millisecond remains relatively constant. The summarized results of the two experiments show that the actions associated with adding, removing, and calculating the edge lengths are performed much more slowly than adding and removing the vertices.

***Keywords*: Graph, Graph Actions Monitoring, Visualization, Software Development.**

I. INTRODUCTION

The graph theory is a fundamental part of the discrete mathematics [1], [2], [3]. Many real problems can be modeled with graphs [4], [5] and others can be described with graph structures [6]. Different problems, such as finding the shortest paths [7], schedule theory problems [8], and many others (including NP-hard problems) can be described with graphs and solved by the corresponding algorithms [9]. Some more specific problems from other areas of science can also be modeled with graphs [10], [11], [12]. Therefore, developing software products for visualizing graph structures [13] and analysis of actions with them is an important issue [14], [15]. The options for describing graph structures with block-based programming languages [16] and domain-specific languages [17] are also interesting aspects of the development of research in this field. In some methods using graph structures, data are stored in databases [18] and can be accessed through cloud technologies and web services [19].

There are many programming environments suitable for developing applications for different operating systems. Moreover, most of these environments provide the ability to compile the same programming code for different operating systems. Visual development and event-oriented programming environments are becoming more and more widely used in the development of different types of applications with different uses. These environments usually use a number of pre-compiled modules (libraries) and components that enable rapid building of sophisticated software products. On the other hand, this is the reason why the generated code is large. For example, an application compiled for a 64-bit Windows operating system and containing only one main window (without any functionality) has a size of 10 megabytes.

From the point of view of programming languages, they are very well developed. For example, environments such as RAD Studio (including Delphi and C++ Builder applications) (www.embarcadero.com/products/rad-studio) and Visual Studio (www.visualstudio.com) give the possibility to initially design the user interface of an application, and then add specific functionality to it. This can be developed with a high-level programming language, depending on the programmer's preferences. For this purpose, development environments have multiple embedded compilers (one for each of the supported programming languages). This allows developers to focus on the application development process rather than on the control structures and syntax of the programming language. In addition, the rapid application development environments provide the opportunity for developing a software project by more developers at the same time. In this way, each programmer can write the program code in a preferred programming language. Finally, during the process of compiling and linking the different units in the project, the development environment combines the individual blocks into one executable file. This file usually contains instructions in an intermediate language. Later, they are compiled into machine instructions for the target operating system when the application starts.

The General Programming Languages (GPL) develop rapidly, and their capabilities, such as commands, reserved words and class libraries, are generally available in all of them [20].

Revised Manuscript Received on November 30, 2019.
* Correspondence Author
Velin Spasov Kralev*, Assis. Professor at Department of Informatics, South-West University "Neofit. Rilski", Blagoevgrad, Bulgaria. Email: velin_kralev@swu.bg
Radoslava Stankova Kraleva, Assis. Professor at Department of Informatics, South-West University "Neofit. Rilski", Blagoevgrad, Bulgaria. Email: rady_kraleva@swu.bg

As already mentioned, most integrated development environments (such as Visual Studio and RAD Studio) offer a choice of a programming language from among several possible ones [21]. This lets developer teams to use an integrated environment to develop a single project but to implement different modules in different languages [22]. In this study, the application used to perform the experiments was developed with IDE C++ Builder [23]. C++ Builder is part of the RAD Studio package. The compiler for this integrated application development environment is C++ based.

II. FUNCTIONAL CAPABILITIES OF THE GRAPHANALYSER APPLICATION

The **GraphAnalyser** application has been developed for the purpose of the planned experiments. A working session with the **GraphAnalyser** application is shown in Fig. 1.

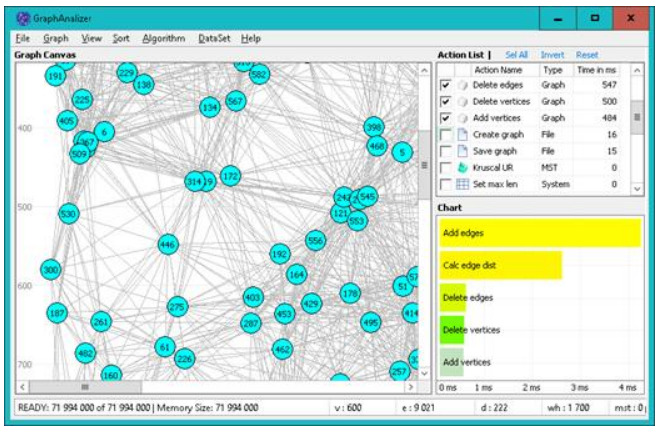

Fig. 1.A work session with the GraphAnalyser application

The developed software has the following functionalities: create, store, and load graphs from external text files that are easy to convert and transfer. It displays a dynamic list of vertices and their characteristics: index (number), coordinates (x and y) and degree (Fig. 1 the *Vertex List* panel). The degrees of the vertices are automatically calculated. This occurs when new edges are added to the graph (i.e., when certain pairs of vertices are associated with edges). The application also displays a dynamic list of edges and their characteristics: index (number), start vertex, end vertex and weight (Fig. 1 the *Edge List* panel). The application automatically calculates the weights of the edges at the same time of calculating the degrees of the vertices.

The **GraphAnalyser** application displays a dynamic list of actions and their characteristics: name, type, and run time in milliseconds (Fig. 1 the *Action List* panel). These characteristics are also used to sort the list of actions. In addition, they may or may not be marked for analysis. It shows a chart for comparing the run time of the marked actions (Fig. 1 the *Chart* panel). The application displays the graph visually in the *Graph Canvas* panel.

The main menu of the **GraphAnalyzer** application provides many other functions, such as: adding and removing vertices and edges; setting the size of the graph drawing canvas; hiding and showing different controls from the application's graphical user interface; selecting an algorithm for sorting the vertices or edges (by different criteria, for example, vertex degree or edge weight); selecting an algorithm for finding a minimum spanning tree, and others.

However, since this application is not an object of this study, but a tool for conducting experiments, it will not be discussed in detail.

III. A METHODOLOGY FOR CONDUCTING THE EXPERIMENTS

The experiments in this study were conducted with the software presented in the previous section. The **GraphAnalyser** application was run on a standard computer configuration with 64-bit Windows 10 operating system. The hardware configuration has the following parameters: Processor: Intel (R) Core (TM) i7-4712MQ CPU @ 2.30GHz 2.30GHz; RAM Memory: 8.00 GB.

The actions that have been analyzed have been divided into two groups. The first group includes actions that are related to creating and modifying a graph - adding and removing vertices and adding and removing edges. The second group includes actions that are related to calculating the amount of RAM required to store one graph in the computer memory. According to the methodology of the experiments, it is necessary to generate 30 graphs divided into two groups of 15. The corresponding graphs in each of the groups must have a measurable number of items. The first group will contain graphs that have only vertices, and the second group will contain graphs that have both vertices and edges. It is necessary that the number of vertices of a graph of the first group be equal to the number of vertices and edges of the corresponding graph in the second group. The first graph of each group will contain 10 million items. Each subsequent graph will have 10 million items more than the previous one. The main purpose is to determine how increasing the number of items in the graph affects the time for performing certain actions on it. The number of the edges in a complete graph (if the number of vertices is known) can be calculated by the formula (1):

$$edges = (vertices*(vertices - 1))/2 \quad (1)$$

In this case, it is necessary to solve equation (1) in terms of the number of vertices, because the number of edges is known. The goal is to define such a number of vertices in the graph that after connecting each vertex with all the others, to obtain a number of edges approximately equal (or exactly equal) to 10 million (20 million, 30 million, etc. every next number with 10 million more edges). In other words, it is necessary to solve the quadratic equation (2):

$$vertices^2 - vertices - 2*edges = 0 \quad (2)$$

In this quadratic equation:

$$a = 1;\ b = -1;\ c = 2*edges,\ D = b^2 - 4*a*c = (-1)^2 - 4*1*(-2*edges) = 1 + 8*edges$$

Since D > 0 for each natural number edges, the quadratic equation has two real roots. In this case, only the positive root of the quadratic equation will be used, since the number of vertices in a graph is a natural number, i.e. greater than or equal to one (the case when the graph is empty is not considered). Thus, the number of vertices in a graph can be calculated by the formula (3):

$$vertices = (1+\sqrt{(1+8*edges)})/2 \quad (3)$$

Therefore, in order to create a complete graph containing 10 million edges (or the nearest integer up to 10 million), the number of vertices can be calculated as follow:

$$vertices = (1+\sqrt{(1+8*10\ 000\ 000)})/2 = (1+\sqrt{(80\ 000\ 001)})/2 = 8945.27/2 = 4472.6$$

The resulting value must be rounded to the nearest integer (in this case 4473, since the number of vertices in the graph is a natural number). Thus, to create a complete graph containing the 4473 vertices, 10 001 628 edges will be needed because:

$$edges = (vertices*(vertices-1))/2 = (4\ 473*4\ 472)/2 = 10\ 001\ 628$$

The number of vertices and edges in the second group of graphs is calculated according to (3). The graphs in the first group contain only vertices. Their number is equal to the number of vertices plus the number of edges of the corresponding graphs in the second group.

Table- I: Summary information for the two graph groups

GV Graphs			GE Graph			
ID	*Vertices*	*Edges*	*ID*	*Vertices*	*Edges*	*Vertice+Edges*
1	*2*	*3*	*4*	*5*	*6*	*7*
GV_1	10 006 101	0	GE_1	4 473	10 001 628	10 006 101
GV_2	20 005 975	0	GE_2	6 325	19 999 650	20 005 975
GV_3	30 004 131	0	GE_3	7 746	29 996 385	30 004 131
GV_4	40 010 985	0	GE_4	8 945	40 002 040	40 010 985
GV_5	50 015 001	0	GE_5	10 001	50 005 000	50 015 001
GV_6	60 011 490	0	GE_6	10 955	60 000 535	60 011 490
GV_7	70 015 861	0	GE_7	11 833	70 004 028	70 015 861
GV_8	80 017 575	0	GE_8	12 650	80 004 925	80 017 575
GV_9	90 014 653	0	GE_9	13 417	90 001 236	90 014 653
GV_10	100 019 296	0	GE_10	14 143	100 005 153	100 019 296
GV_11	110 016 361	0	GE_11	14 833	110 001 528	110 016 361
GV_12	120 008 778	0	GE_12	15 492	119 993 286	120 008 778
GV_13	130 015 875	0	GE_13	16 125	129 999 750	130 015 875
GV_14	140 021 745	0	GE_14	16 734	140 005 011	140 021 745
GV_15	150 017 181	0	GE_15	17 321	149 999 860	150 017 181

The number of the vertices in the GVi graphs and the sum of the number of the vertices and edges in the GEi graphs are equal. This gives a reason to compare (in terms of time) the actions that are applied to the corresponding graphs of the two groups. Furthermore, indexing of elements in dynamic arrays that store vertices and edges is only related to the iterative process, not to accessing and processing the data stored in the computer's memory.

IV. EXPERIMENTAL RESULTS

The first experiment is related to an analysis of the time required to add vertices to the graph and the time required to remove the vertices from the graph. The effect of increasing the number of vertices in a graph over the time required to perform adding vertices and removing vertices actions is investigated. The summarized results of this experiment are shown in Table II.

Table- II: Summarized results from the first experiment

Graphs		Actions				
ID	*Vertices*	*Add vertices*	*Delete vertices*	*Col 3 / Col 4*	*Add vertices per ms*	*Delete vertices per ms*
1	*2*	*3*	*4*	*5*	*6*	*7*
GV_1	10 006 101	106	72	0.6792	94 397	138 974
GV_2	20 005 975	204	125	0.6127	98 069	160 048
GV_3	30 004 131	297	204	0.6869	101 024	147 079
GV_4	40 010 985	391	250	0.6394	102 330	160 044
GV_5	50 015 001	485	312	0.6433	103 124	160 304
GV_6	60 011 490	593	391	0.6594	101 200	153 482
GV_7	70 015 861	687	437	0.6361	101 915	160 219
GV_8	80 017 575	781	515	0.6594	102 455	155 374
GV_9	90 014 653	875	578	0.6606	102 874	155 735
GV_10	100 019 296	969	641	0.6615	103 219	156 036
GV_11	110 016 361	1 093	703	0.6432	100 655	156 496
GV_12	120 008 778	1 219	750	0.6153	98 449	160 012
GV_13	130 015 875	1 328	828	0.6235	97 904	157 024
GV_14	140 021 745	1390	890	0.6403	100 735	157 328
GV_15	150 017 181	1 484	938	0.6321	101 090	159 933

Column 5 shows the ratio between the time for removing all vertices from a graph against the time for adding the same number of vertices (in the same graph). If the action of adding vertices to a graph is assumed to be performed per unit of time, what part of this time (in %) the action for removing the same number of vertices from the same graph would take? The results show that this percentage varies between 61% and 69% for all graphs.

Column 6 gives information about the number of added vertices for 1 millisecond time. The values in this column are calculated by dividing the number of all added vertices (Column 2) by the corresponding values in Column 3. Column 3 gives the total time (in milliseconds) to add all vertices in the graph.

Similarly, Column 7 shows the number of the removed vertices for 1 millisecond.

The values in this column are calculated by dividing the number of all removed vertices (corresponding to the value in Column 2) by the corresponding values in Column 4. Column 4 gives the total time (in milliseconds) to remove all vertices from the graph.

The results show that when the number of vertices in a graph increases linearly, the time to add and remove these vertices is also linearly increasing. This trend can be seen in Fig. 2. Fig. 2 shows this trend.

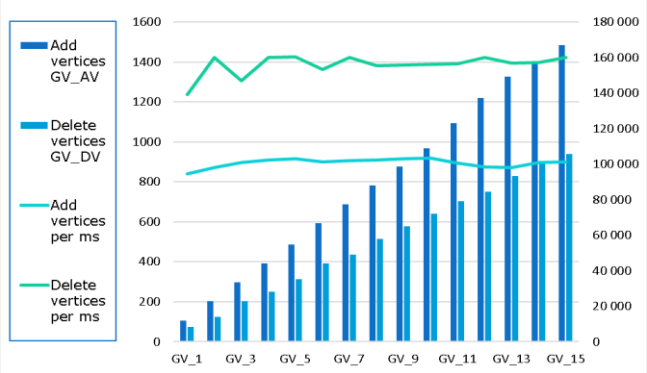

Fig. 2.The effect of increasing the number of vertices in a graph on the time for adding and removing vertices

The effect of increasing the number of vertices in a graph on the time for adding and removing vertices. show that when the number of vertices in a graph increases linearly, the number of vertices added per 1 millisecond remains relatively constant (Table II and Fig. 2). However, the number of vertices removed from graphs GV_1 – GV_7 at the same time (1 millisecond) varies.

The second experiment is related to an analysis of the time for adding and removing edges from a graph. The graphs for this experiment (GEi) have a total number of vertices and edges, just as many as the vertices of the graphs (GVi) in the first experiment.

It examines how the increase of the number of edges influences on the time for adding and removing edges. During this experiment, the available vertices in the graph are also removed. When a vertex of the graph is removed, the **GraphAnalyser** automatically removes the incident edges with this vertex. In this way, after removing all vertices in a graph, all its edges will be automatically removed as well.

When an edge is added to a graph, first, its weight is calculated. This weight is actually the distance between the two vertices that the corresponding edge connects. This distance (d) is calculated according to the following formula (Euclidean distance):

$$d = \sqrt{((x\_2 - x\_1)^2 + (y\_2 - y\_1)^2)},$$

where x1, y1 and x2, y2 are respectively the coordinates of the two vertices incident with the edge. The time to perform this action is part of the time to perform the "add edges" action.

In Table III the values in columns 3, 4 and 5 are related. Column 3 shows the time needed to add the corresponding number of edges to the graph. The values in column 3 also contain the times for calculating the weights of the edges (Column 4). The values in column 5 show the differences between the values in column 3 and column 4. This time is comparable to the time presented in column 3 of Table II because no preliminary calculations are made when adding vertices in a graph. Column 6 shows the times required to delete all edges (and vertices) in the investigated graphs. Column 7 shows the ratio between the edge removing time and the edge adding time. If the "add edges" action is assumed to be performed per unit of time, then what part of this time (in %) does the "remove edges" action take? The values show that this percentage varies between 12% and 20% for all graphs. The results from this experiment show that when the number of edges in a graph increases linearly, the time to add and remove these edges is also linearly increasing.

Table- III: Summarized results from the second experiment

Graphs		Actions				
ID	*Vertices + Edges*	*Add edges*	*Calc edge distance*	*Col 3 - Col 4*	*Delete vertices & edges*	*Col 6 / Col 5*
1	*2*	*3*	*4*	*5*	*6*	*7*
GE_1	10 006 101	796	500	296	52	0.1757
GE_2	20 005 975	1594	1000	594	104	0.1751
GE_3	30 004 131	2266	1516	750	147	0.1960
GE_4	40 010 985	3375	2000	1375	182	0.1324
GE_5	50 015 001	3985	2531	1454	245	0.1685
GE_6	60 011 490	4672	3000	1672	297	0.1776
GE_7	70 015 861	5359	3515	1844	334	0.1811
GE_8	80 017 575	6968	4031	2937	359	0.1222
GE_9	90 014 653	7594	4547	3047	375	0.1231
GE_10	100 019 296	8234	5000	3234	516	0.1596
GE_11	110 016 361	8922	5515	3407	531	0.1559
GE_12	120 008 778	9594	6016	3578	578	0.1615
GE_13	130 015 875	10312	6516	3796	641	0.1689
GE_14	140 021 745	10984	7031	3953	672	0.1700
GE_15	150 017 181	11734	7516	4218	703	0.1667

Fig. 3 shows that the "add edge" action takes the longest time to perform. This is understandable because this action actually contains two other actions. The first action is related to calculating the lengths of the edges, and the second action is related to allocating memory for storing the edges data.

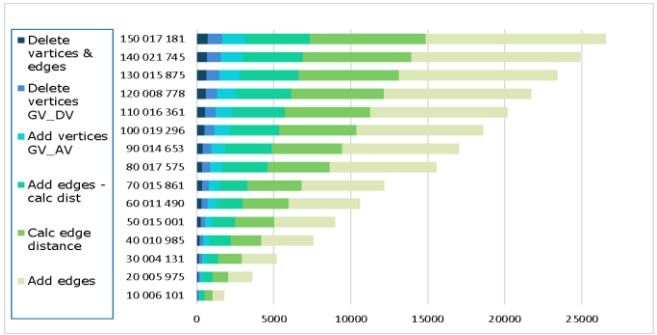

Fig. 3.The influence of the number of vertices and edges on the time to perform the analysed actions

The summarized results of the two experiments show that actions related to adding, removing, and calculating edge lengths are performed more slowly than actions related to adding and removing vertices.

The values in Table II and Table III were averaged by six independent starts of the **GraphAnalyser** application. This is necessary for more accurately measuring of the execution time of an action.

V. CONCLUSION

In this paper the basic concepts of using application development environments were presented. A brief analysis of the key features of the environment that was used to develop the **GraphAnalyser** application was made. This application provides an opportunity to make real-time benchmarking for the performance of several pre-selected actions (related to the graph structure). According to the experimental methodology, the experiments were grouped into two groups. The first group includes actions related to the graph vertices, and the second group includes actions related to the graph edges. The obtained results show that when the number of vertices in the graph increases linearly, the time for adding and removing these vertices also linearly increases. When the number of the graph vertices linearly increases, the number of the added vertices per millisecond remains relatively constant. However, the number of the removed vertices for a millisecond for graphs containing between 10 and 70 million vertices varies. Similarly, when the number of graph edges increases linearly, the number of added edges per millisecond remains relatively constant. The summarized results from the two experiments show that the actions related to adding and removing the edges in a graph are executed significantly more slowly than the actions associated with adding and removing the vertices in a graph.

ACKNOWLEDGMENT

The reported work is a part of and was supported by project DN02/3 "Modelling of voluntary saccadic eye movements during decision making" funded by the Bulgarian Science Fund.

REFERENCES

1. Grytczuk, J. Thue type problems for graphs, points, and numbers. Discrete Mathematics, 2008, vol. 308 (19), pp. 4419-4429.
2. R. J. Wilson. Introduction to Graph Theory, 5th ed., Prentice Hall, 2010.
3. Alekseev, V. E., Boliac, R., Korobitsyn, D. V., Lozin, V. V. NPhard graph problems and boundary classes of graphs. Theoretical Computer Science, vol. 389(1), pp. 219–236.
4. Kurapov, S.V., Davidovsky, M.V., Tolok, A.V. Visual algorithm for coloring planar graphs. Scientific Visualization, 2018, vol. 10(3), pp. 1-33.
5. Borisova N. An approach for Ontology Based Information Extraction (OBIE). Information Technologies and Control, 2015, vol. 12(1), pp. 15-20.
6. Kurapov, S.V., Davidovsky, M.V., Tolok, A.V. A modified algorithm for planarity testing and constructing the topological drawing of a graph. The thread method. Scientific Visualization, 2018, vol. 10(4), pp. 53-74.
7. Sapundzhi, F.I., Popstoilov, M.S. Optimization algorithms for finding the shortest paths. Bulgarian Chemical Communications, 2018, vol. 50(Special Issue B), pp. 115-120.
8. Kralev, V., Kraleva, R., Kumar, S. A modified event grouping based algorithm for the university course timetabling problem. International Journal on Advanced Science, Engineering and Information Technology, 2019, vol. 9(1), pp. 229-235.
9. Sabeen, S., Arunadevi, R., Kanisha, B., Kesavan, R. Mining of sequential patterns using directed graphs. International Journal of Innovative Technology and Exploring Engineering, 2019, vol. 8(11), pp. 4002-4007.
10. Sapundzhi, F. Scoring functions and modeling of structure-activity relationships for cannabinoid receptors. International journal of online and biomedical engineering, 2019, vol. 15 (11), pp. 139-145.
11. Kraleva, R. ChilDiBu - A mobile application for Bulgarian Children with special educational needs. International Journal on Advanced Science, Engineering and Information Technology, 2017, vol. 7(6), pp. 2085-2091.
12. Sapundzhi, F. Computer simulation and investigations of the roof mount photovoltaic system. International journal of online and biomedical engineering, 2019, vol. 15(12), pp. 88-96.
13. Agarwal, S., Aggarwal, A., Dixit, S. Model transformation of declarative user interface from platform specific model to platform independent model using graphical notation. International Journal of Innovative Technology and Exploring Engineering, 2019, vol. 8(12), pp. 2371-2377.
14. Gasparic, M., Gurbanov, T., Ricci, F. Improving integrated development environment commands knowledge with recommender systems. Proceedings - International Conference on Software Engineering, 2018, pp. 88-97.
15. Coenen, J., Gross, S., Pinkwart, N. Comparison of feedback strategies for supporting programming learning in integrated development environments (IDEs). Advances in Intelligent Systems and Computing, 2018, vol. 629, pp. 72-83.
16. Kraleva, R., Kralev, V., Kostadinova, D. A methodology for the analysis of block-based programming languages appropriate for children. Journal of Computing Science and Engineering, 2019, vol. 13(1), pp. 1-10.
17. Kosar, T., Gaberc, S., Carver, J.C., Mernik, M. Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments. Empirical Software Engineering, 2018, vol. 23(5), pp. 2734-2763.
18. Kraleva, R., Kralev, V., Sinyagina, N., Koprinkova-Hristova, P., Bocheva, N. Design and analysis of a relational database for behavioral experiments data processing. International Journal of Online Engineering, 2018, vol. 14(2), pp. 117-132.
19. Kralev, V., Kraleva, R., Sinyagina, N., Koprinkova-Hristova, P., Bocheva, N. An analysis of a web service based approach for experimental data sharing. International Journal of Online Engineering, 2018, vol. 14(9), pp. 19-34.
20. Donovan A, Kernighan B. The Go Programming Language (Professional Computing Series). Addison-Wesley Professional; 2015.
21. Scott ML. Programming language pragmatics. 4th Edition. Morgan Kaufmann; 2015 Dec.
22. Sebesta RW. Concepts of programming languages. Reading, Pearson; 2015 Feb.
23. Wang, Y.Z., Fan, Q. Practice and discussion on mixed teaching of c language programming. Computer Knowledge and Technology, 2017, vol. 42 (13), pp. 128-129.

AUTHORS PROFILE

Velin Kralev is an assist. professor of Computer Science at the Faculty of Mathematics and Natural Sciences, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria. He defended his Ph.D. thesis in 2010. His research interests include database systems development, optimization problems of the scheduling theory, graph theory, and component-oriented software engineering.

Radoslava Kraleva is an assist. professor of Computer Science at the Faculty of Mathematics and Natural Sciences, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria. She defended her Ph.D. thesis on "Acoustic-Phonetic Modeling for Children's Speech Recognition in Bulgarian" in 2014. Her research interests include child-computer interaction, speech recognition, mobile app development and computer graphic.