Łukasz CZEPIELIK[1], Konrad BOROŃ[1], Dominik PEZDA[1]

Opiekun naukowy: Stanisław ZAWIŚLAK[2]

THE PROBLEM OF GRAPH TRIPARTITION IN A TWO-CRITERIA EVOLUTIONARY APPROACH

Summary: The article discusses the problem of the tripartition (tripartite division) of a randomly generated, undirected graph using an evolutionary algorithm. The designed genetic algorithm is based on a two-criteria function of adaptation of population members. To solve that issue a dedicated computer program has been prepared, which allows investigating the behaviour of the algorithm for different sets of input data. The work cycle of the created research application is based on the following steps: defining the input parameters, individual iterations of the designed source code (genetic algorithm) solving the problem, visualization of partial results, evaluation and registration of the obtained final results.

Keywords: non-directional graph, graph partitioning, tripartition, evolutionary approach, multi-criteria optimization, genetic algorithm, pareto front, searching for extremum

PROBLEM TRÓJPODZIAŁU GRAFU W DWUKRYTERIALNYM UJĘCIU EWOLUCYJNYM

Streszczenie: W artykule został omówiony problem trójpodziału losowo generowanego, nieskierowanego grafu przy wykorzystaniu algorytmu ewolucyjnego. Zaprojektowany algorytm genetyczny został oparty na dwukryterialnej funkcji przystosowania osobników danej populacji. Do rozwiązania omawianego problemu został przygotowany dedykowany program komputerowy, który pozwala na badanie zachowania algorytmu dla różnych zestawów danych wejściowych. Cykl pracy utworzonego nośnika badawczego oparty jest na następujących krokach: zdefiniowanie parametrów wejściowych, poszczególne iteracje zaprojektowanego kodu źródłowego (algorytmu genetycznego) rozwiązującego badany problem, wizualizacja wyników cząstkowych, ewaluacja oraz zapis osiągniętych wyników.

Słowa kluczowe: nieskierowany graf, podział grafu, trójpodział, podejście ewolucyjne, optymalizacja wielokryterialna, algorytm genetyczny, zbiór kompromisów Pareto, poszukiwanie ekstremum

[1] University of Bielsko-Biala, Faculty of Mechanical Engineering and Computer Science, Computer Science, Software Development Techniques, email l.czepielik@gmail.com, konradmb@o2.pl, dominik.pezda96@gmail.com;
[2] dr hab. inż., prof. ATH, University of Bielsko-Biala, Faculty of Mechanical Engineering and Computer Science, email szawislak@ath.bielsko.pl;

1. Introduction

The aim of the project is to create a program that will allow to solve the problem of graph tripartition in a two-criteria approach using an evolutionary algorithm. The graph tripartition problem is one of the typical graph theory issues and operations that are associated with them.

The problem of graph tripartition is to assign each vertex of a considered graph to one of three available groups [3]. In case of developed application the described graph problem will be examined from a two-criteria aspect, which means that the task will be related to the topic of multi-criteria optimization. The objective that the algorithm should be able to perform a double minimum task, i.e. searching for a minimum of both defined criteria simultaneously. The first optimization criterion is the total number of edges in the graph, which connect vertices that do not belong to the same group, while the second criterion of the minimization problem is the total sum of weights of this type of edge.

The presented definition of the multi-criteria optimization task communicate that the result of the solution could not be just one point, which can be classified as the best point, or the closest to ideal point, without taking into consideration the limitations. However in some cases such solution could be obtain. The result of the two-criteria optimization of the graph tripartition problem will be the Pareto compromise set. This set of compromises contains solutions that can be called optimal, but from the group of which it is not possible to select a point that would be ideal in terms of all analyzed criteria. Each of the problem solutions that are on the Pareto front (the set of compromises) are optimal according to some criteria and are not optimal for others. In order to select the point which is the expected point of settling the task, one of the methods of arbitrary weighting of particular criteria should be used. In the case of a graph tripartition problem, the dominant criterion is the total sum of the weights of the edges between two vertices that do not belong to the same group, and it is this criterion that is used to finally determine the one closest to the ideal solution [1].

The discussed problem of graph tripartition with searching for a minimum based on two criteria must in some way be tested repeatedly using different configurations of vertex assignments to groups. In order to make a comprehensive analysis, an evolutionary algorithm will be adopted, which will let to search the available solutions to the problem in a very structured way. The genetic algorithm allows to reproduce the processes of evolution of living beings such as mutation, crossover and selection in a strictly simulated test environment which is naturally the application environment. Moreover the classical algorithms are formulated for a single criterion therefore their generalization is difficult and requires a new proof of correctness. Thanks to the use of a genetic algorithm, a well-functioning mechanism will be implemented, which should allow for an in-depth search for a solution to the problem. A great advantage of using evolutionary algorithms is the possibility to improve the results obtained with numerous algorithm iterations [3].

Due to the specificity of the created program, which requires interaction from the user, its work cycle can be divided into three main parts. In the first, preparatory stage, the user using the application adjusts the parameters used by internal procedures. The user enters data related to the graph to be generated and completes the properties which characterize the evolutionary algorithm. An important parameter entered by the user is the boundary number of iterations for the genetic algorithm (so called 'stop condition'). In practice, the algorithm always performs the given number of iterations. In the next step, when the appropriate button is pressed, the graph generation procedure is started randomly based on the settings entered. The finished graph is shown on the user interface as a matrix of incidents and weights, and also in an interactive visual form [6].

In the second stage of application work, the user initiates the execution of the genetic algorithm by pressing the appropriate button. The genetic algorithm performs each of the iterations one by one. The interruptions in execution of particular iterations are intended to provide a better possibility for the end user to observe the work of the algorithm and the effects of its work achieved in subsequent iterations. Time for which the execution of the algorithm is stopped can be changed via the appropriate field in the user interface. During this break from performing subsequent iterations of the genetic algorithm, the following sections of the graphical layer of the program window are refreshed: interactive visualization of the graph tripartition of the best individuals from the current iteration, Pareto compromise set together with the selected best individual as a point on the chart, minimum fluctuation charts and average value of the adaptation function in the domain of iteration [9].

In the last stage of the application work, which starts from the termination of the genetic algorithm at the level of border iteration, whose number was previously entered by the user, a presentation of the final results obtained is made for the problem of graph tripartition in a two-criteria evolutionary perspective. At this point, the user can decide to save the application history to a text file in the .txt extension for a particular scenario that has just been performed. After selecting the appropriate button, all the most important data about the graph under consideration and the individuals created, together with their adaptation ratings and solutions in the iterations will be saved to a new text file.

2. Block Figure of program operation and genetic algorithm

The evolutionary algorithm created to solve the problem of graph tripartition in a two-criteria approach contains operations typical for genetic algorithms. In the analyzed problem of a single individual, it is not possible (and not needed) to describe a set of attributes in the form of binary notation, so typical genetic operators do not apply here [10]. The individual in the created evolutionary algorithm is a graph tripartition, i.e. assigning graph vertices to equal (usually but not always) sets. Due to the different representation within the source code of a single individual, particular operators required adequate transformations in the implementation. Figure 1 shows a block diagram of the designed application, while the Figure 2 shows a schematic diagram of the operation of the genetic algorithm together with the selected successive component operations [2].

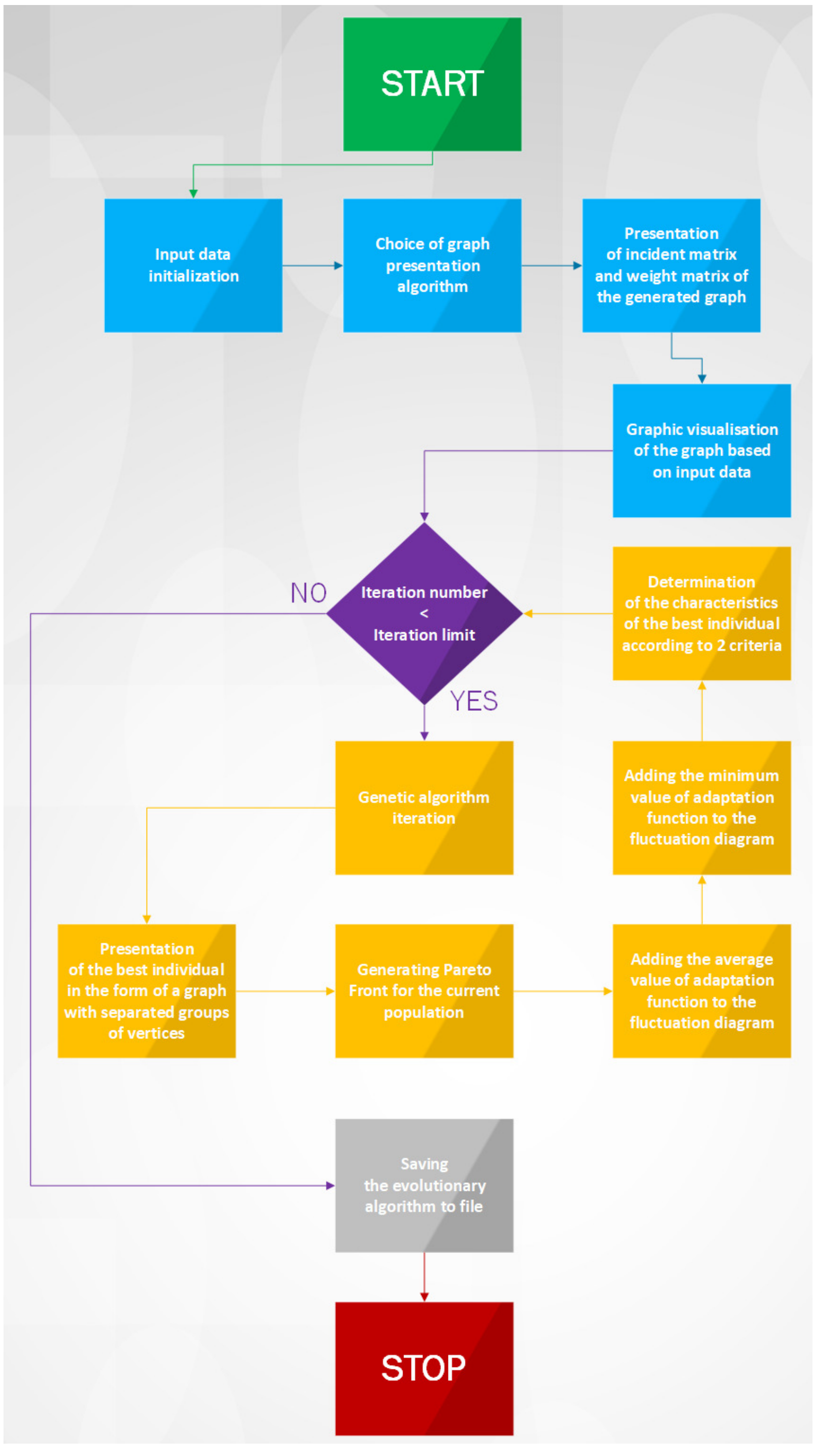

START
Input data initialization
Choice of graph presentation algorithm
Presentation of incident matrix and weight matrix of the generated graph
Graphic visualisation of the graph based on input data
Iteration number < Iteration limit
NO
YES
Determination of the characteristics of the best individual according to 2 criteria
Genetic algorithm iteration
Adding the minimum value of adaptation function to the fluctuation diagram
Presentation of the best individual in the form of a graph with separated groups of vertices
Generating Pareto Front for the current population
Adding the average value of adaptation function to the fluctuation diagram
Saving the evolutionary algorithm to file
STOP

Figure 1. Block scheme of the prepared computer program

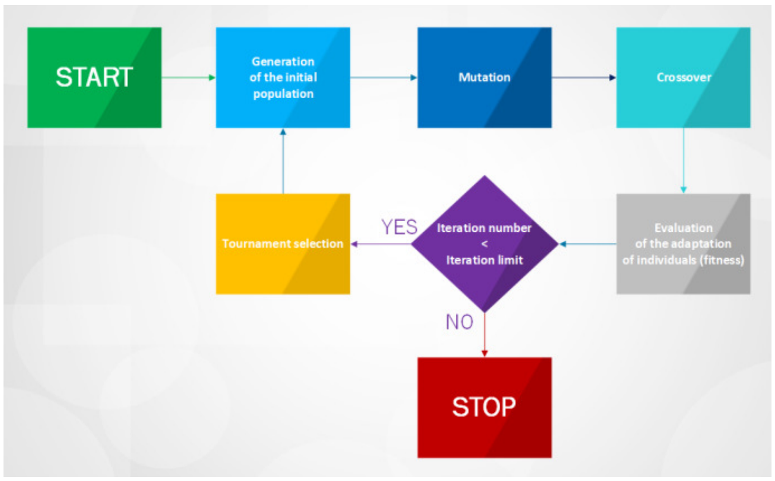

START
Generation of the initial population
Mutation
Crossover
Evaluation of the adaptation of individuals (fitness)
Iteration number < Iteration limit
YES
Tournament selection
NO
STOP

Figure 2. Block scheme of the genetic algorithm

3. USE-CASE diagram of the UML standard

In Figure 3, a UML Use-Case diagram is shown, which presents the executable actions available to the user of the designed application.

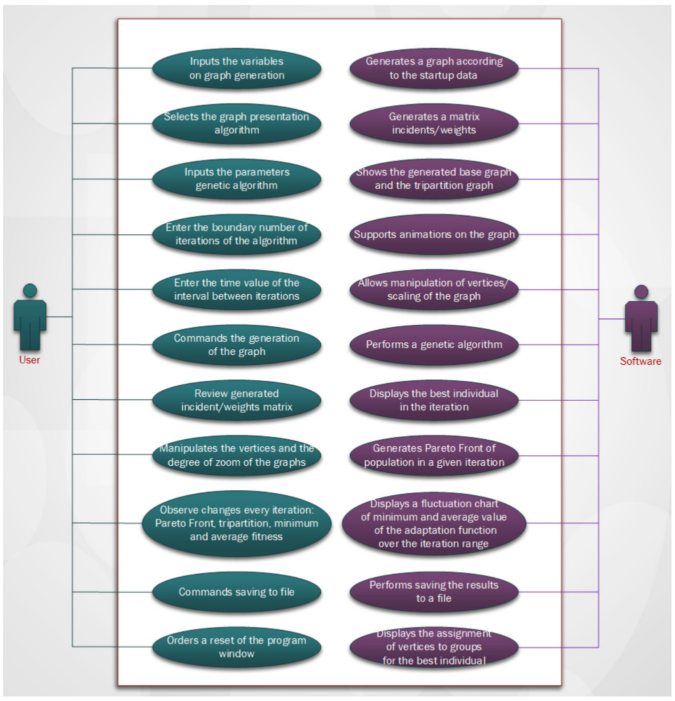
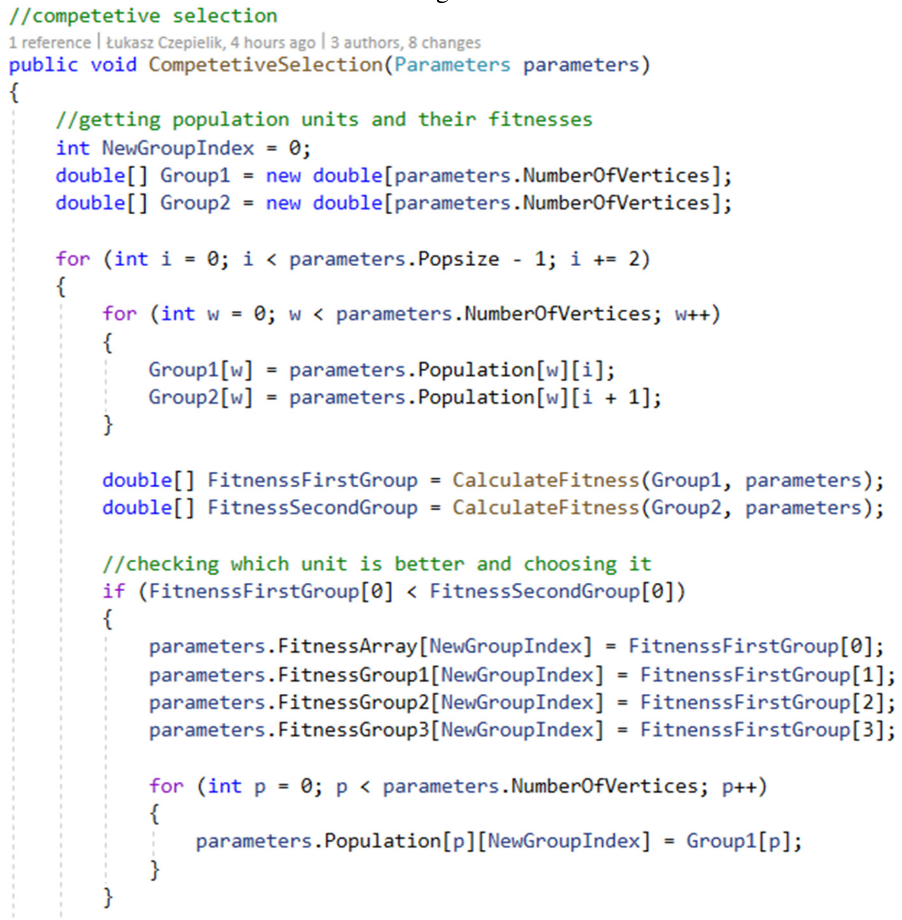

User
Inputs the variables on graph generation
Selects the graph presentation algorithm
Inputs the parameters genetic algorithm
Enter the boundary number of iterations of the algorithm
Enter the time value of the interval between iterations
Commands the generation of the graph
Review generated incident/weights matrix
Manipulates the vertices and the degree of zoom of the graphs
Observe changes every iteration: Pareto Front, tripartition, minimum and average fitness
Commands saving to file
Orders a reset of the program window
Generates a graph according to the startup data
Generates a matrix incidents/weights
Shows the generated base graph and the tripartition graph
Supports animations on the graph
Allows manipulation of vertices/ scaling of the graph
Performs a genetic algorithm
Displays the best individual in the iteration
Generates Pareto Front of population in a given iteration
Displays a fluctuation chart of minimum and average value of the adaptation function over the iteration range
Performs saving the results to a file
Displays the assignment of vertices to groups for the best individual
Software

Figure 3. USE-CASE UML diagram

4. Genetic operators

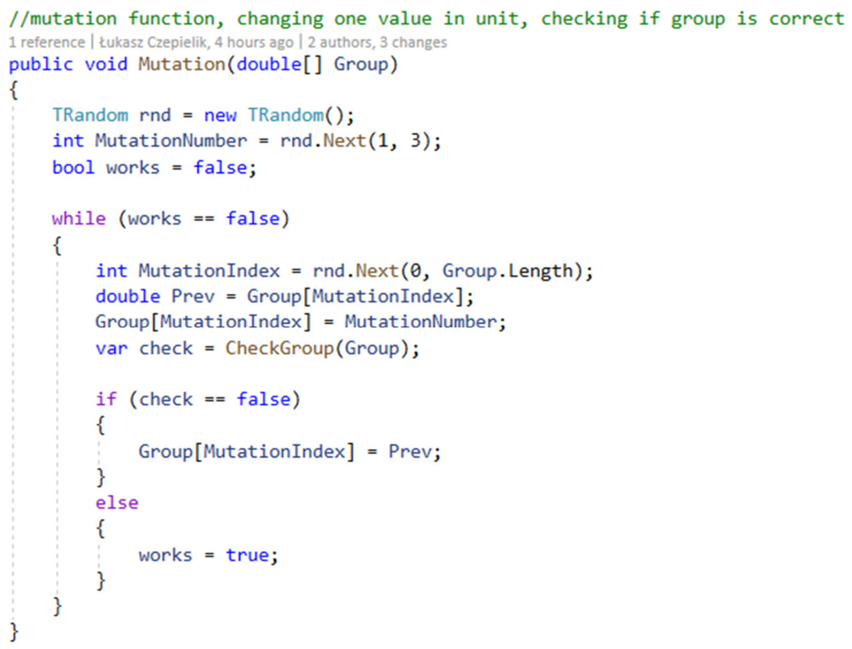
In the designed genetic algorithm, the tournament selection method was used to solve the problem of graph tripartition. After calculating the adaptation function for each of the individuals in a given iteration, the pairing of individuals is made. The next step is to remove an individual from the population from the pair, which has a lower value of the adaptation function. At the tournament selection operation, we receive a population reduced by half, which will be supplemented by new, randomly generated individuals at the beginning of the next iteration. A fragment of the tournament selection code is shown in Figure 4.



```
//competetive selection
1 reference | Łukasz Czepielik, 4 hours ago | 3 authors, 8 changes
public void CompetetiveSelection(Parameters parameters)
{
    //getting population units and their fitnesses
    int NewGroupIndex = 0;
    double[] Group1 = new double[parameters.NumberOfVertices];
    double[] Group2 = new double[parameters.NumberOfVertices];

    for (int i = 0; i < parameters.Popsize - 1; i += 2)
    {
        for (int w = 0; w < parameters.NumberOfVertices; w++)
        {
            Group1[w] = parameters.Population[w][i];
            Group2[w] = parameters.Population[w][i + 1];
        }

        double[] FitnenssFirstGroup = CalculateFitness(Group1, parameters);
        double[] FitnessSecondGroup = CalculateFitness(Group2, parameters);

        //checking which unit is better and choosing it
        if (FitnenssFirstGroup[0] < FitnessSecondGroup[0])
        {
            parameters.FitnessArray[NewGroupIndex] = FitnenssFirstGroup[0];
            parameters.FitnessGroup1[NewGroupIndex] = FitnenssFirstGroup[1];
            parameters.FitnessGroup2[NewGroupIndex] = FitnenssFirstGroup[2];
            parameters.FitnessGroup3[NewGroupIndex] = FitnenssFirstGroup[3];

            for (int p = 0; p < parameters.NumberOfVertices; p++)
            {
                parameters.Population[p][NewGroupIndex] = Group1[p];
            }
        }
```

Figure 4. Tournament selection

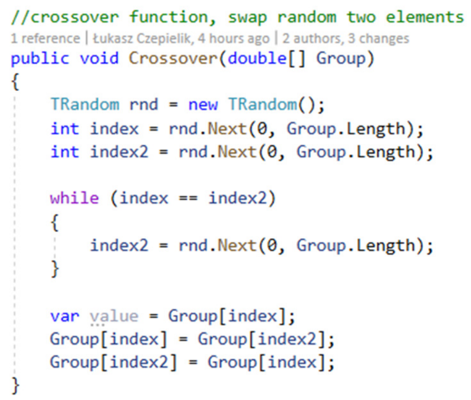
The mutation operation in the implemented evolutionary algorithm relies on selection the individual (the single chromosome) and then randomly determining the vertex number. Due to the fact, that the mutation should - by definition - introduce a slight modification in the individuals subjected to this procedure, therefore in this case, on the basis of a randomly selected vertex, the group to which the vertex was originally assigned is converted. For the reason of the usefulness of the solution, we assume that the vertex must change its assignment to the group (after the mutation operation, it cannot be in the same group as before the procedure). Fragment of the mutation procedure code has been placed in Figure 5.



```
//mutation function, changing one value in unit, checking if group is correct
1 reference | Łukasz Czepielik, 4 hours ago | 2 authors, 3 changes
public void Mutation(double[] Group)
{
    TRandom rnd = new TRandom();
    int MutationNumber = rnd.Next(1, 3);
    bool works = false;

    while (works == false)
    {
        int MutationIndex = rnd.Next(0, Group.Length);
        double Prev = Group[MutationIndex];
        Group[MutationIndex] = MutationNumber;
        var check = CheckGroup(Group);

        if (check == false)
        {
            Group[MutationIndex] = Prev;
        }
        else
        {
            works = true;
        }
    }
}
```

Figure 5. Mutation operation

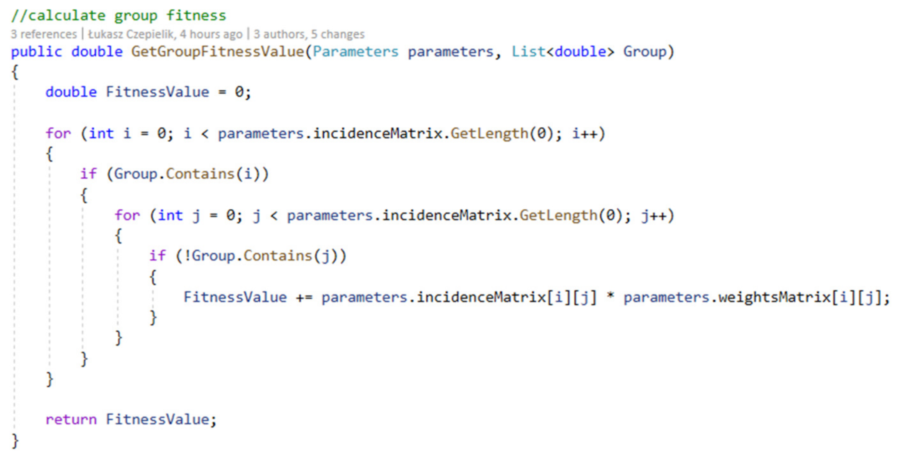
The crossover (crossing) operation that was created for the essence of the problem of graph tripartition depends on selecting a pair of individuals, and then choosing a random vertex based on which the group of origin pair of individuals will be replaced [7]. In order to achieve the desired result, on the basis of the drawn vertices, two descendants are created, which are copies of their parents with the difference that they are assigned to a group of a specific selected vertex, not to their ancestors, and to the other individual forming a couple with a particular parent [10]. Figure 6 shows a listing of the source code fragment containing the crossing operation.



```
//crossover function, swap random two elements
1 reference | Łukasz Czepielik, 4 hours ago | 2 authors, 3 changes
public void Crossover(double[] Group)
{
    TRandom rnd = new TRandom();
    int index = rnd.Next(0, Group.Length);
    int index2 = rnd.Next(0, Group.Length);

    while (index == index2)
    {
        index2 = rnd.Next(0, Group.Length);
    }

    var value = Group[index];
    Group[index] = Group[index2];
    Group[index2] = Group[index];
}
```

Figure 6. Crossover operation

The listing section of the program containing one of the components of the adaptation factor calculation is shown in Figure 7.



```
//calculate group fitness
3 references | Łukasz Czepielik, 4 hours ago | 3 authors, 5 changes
public double GetGroupFitnessValue(Parameters parameters, List<double> Group)
{
    double FitnessValue = 0;

    for (int i = 0; i < parameters.incidenceMatrix.GetLength(0); i++)
    {
        if (Group.Contains(i))
        {
            for (int j = 0; j < parameters.incidenceMatrix.GetLength(0); j++)
            {
                if (!Group.Contains(j))
                {
                    FitnessValue += parameters.incidenceMatrix[i][j] * parameters.weightsMatrix[i][j];
                }
            }
        }
    }

    return FitnessValue;
}
```

Figure 7. Calculation of fitness function

In relation to the formulation of the problem of graph tripartition in the two-criteria evolutionary approach, the function of evaluating the adaptation of an individual should be designed on the basis of the criterion of the number of edges connecting vertices belonging to different groups and the criterion of the sum of weights of this type of edges [10].

The following function was proposed on the basis of these two criteria [10]:

$$f(P) = \sum_{i=1}^{n} \big(c(i) * W_c(i)\big)$$

where:
P - individual (chromosome),
E_c - edges connecting vertices not belonging to the same group,
W_c - the weights of the edges belonging to the set E_c,
n - volume of set E_c.
To convert the considered problem to the more simple case.

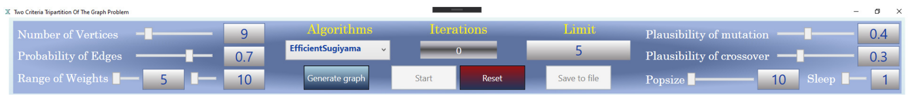
5. Development environment

The program for solving the problem of two-criteria graph tripartition problem is made with the use of C# language in .NET Framework technology, which is responsible for the entire logic module and using WPF technology for user-friendly and intuitive work with the program via the user interface. In order to define the interpreted problem we used the environment Visual Studio 2019 Enterprise and a remote repository to synchronize the software between different platforms during its development.

6. User interface

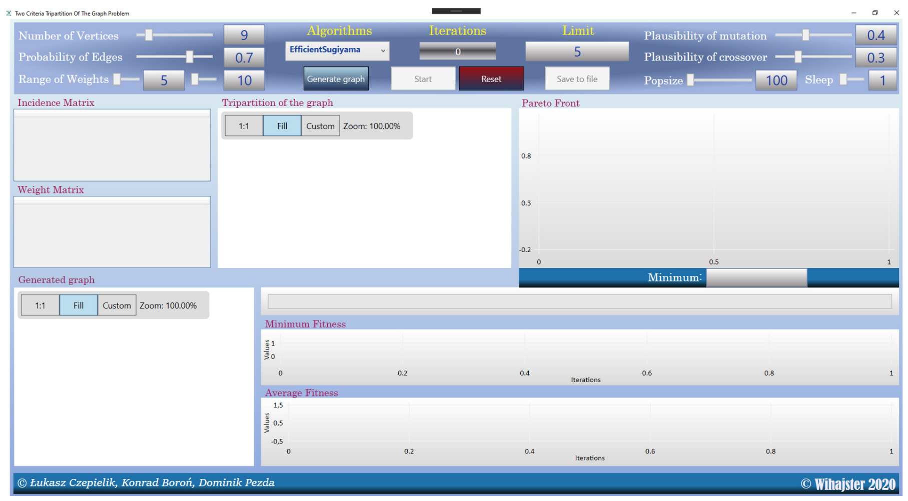
The end-user interface is divided into three sections, where each section corresponds to the next steps to be taken in order to solve the problem of the two-criteria graph tripartition. The sections of the application listed cover issues:
- the input of data necessary to perform the genetic algorithm based on a randomly generated graph,
- visualization of a base-generated random graph in the form of a matrix and the graphical structure,
- visualization of the genetic algorithm and its results in each of the iterations.

The first section of the user interface is the configuration panel of the graph tripartition problem in a two-criteria approach. The whole panel is shown in Figure 8. Figure 9 shows all designed sections of the user interface connected in one program window.


Two Criteria Tripartition Of The Graph Problem

Number of Vertices 9
Probability of Edges 0.7
Range of Weights 5 10
Algorithms: EfficientSugiyama
Iterations: 0
Limit: 5
Plausibility of mutation 0.4
Plausibility of crossover 0.3
Popsize 10
Sleep 1

Figure 8 – Graph tripartition problem factors configuration panel.


Two Criteria Tripartition Of The Graph Problem

Number of Vertices 9
Probability of Edges 0.7
Range of Weights 5 10
Algorithms EfficientSugiyama
Iterations 0
Limit 5
Plausibility of mutation 0.4
Plausibility of crossover 0.3
Popsize 100
Sleep 1

© Łukasz Czepielik, Konrad Boroń, Dominik Pezda
© Wihajster 2020

Figure 9. Application user interface.

The panel shown in Figure 9 can be divided into three sections each of which is responsible for a different aspect of solving the graph tripartition problem. The first of the fragments is the area of determining the characteristics describing the randomly generated graph and the way it is presented. The coefficients that we can determine are:
- number of vertices (it does not have to be a number divided by the number of generated sets (3)),
- the probability of creating an edge between two vertices,
- the range of allowed values of randomly drawn weights for the edge of the graph,
- graphical presentation methods.

The described fragment of the panel, apart from the possibility of entering configuration data, also allows to start the graph generation procedure and its visualization by pressing the "Generate graph" button. A fragment of the panel is shown in Figure 10.

Another panel, which is a part of the generally named configuration section of the graph tripartition problem is the part that determines the behavior of the genetic algorithm. This part includes the following information:
- the probability of an individual mutation operation,
- the probability of an individual being crossed,
- population size,
- the time between iterations (used to pause the algorithm to observe changes in subsequent iterations).

This panel for entering configuration data is shown in Figure 11.

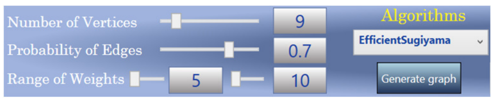

Number of Vertices 9
Probability of Edges 0.7
Range of Weights 5 10
Algorithms EfficientSugiyama

Figure 10. Configuration panel for graph generation process and its presentation

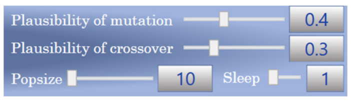

Plausibility of mutation 0.4
Plausibility of crossover 0.3
Popsize 10
Sleep 1

Figure 11. Genetic algorithm parameters configuration panel

The last part of the user interface which is part of the input section is the part controlling the genetic algorithm, where information on the limit number of iterations must be entered. In addition, this section stores a counter which indicates in which iteration of the genetic algorithm the graph being processed is currently located. This section contains the following control buttons, which close the control of procedures sequentially called up within a running application:
- “Start” — launches the genetic algorithm responsible for solving the two-criteria graph division problem,
- “Reset” — resets the application window with all entered/received data. Default data is entered into configuration fields,
- “Save to file” — It allows to capture the genetic algorithm and its results into a .txt file.
The control section of the part of the user interface responsible for configuration of the graph tripartition problem is shown in Figure 12.

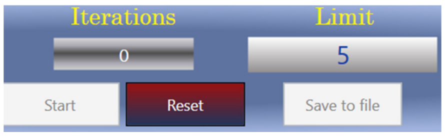
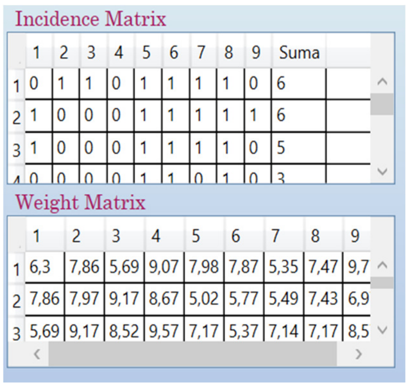

Iterations 0
Limit 5

Figure 12. Section which controls the graph tripartition problem solving

Another large section of the user interface are the panels responsible for presenting a randomly generated graph, which in the future will become the basis for the analysis of the graph tripartition problem. The first of the panels included in this section is a panel that stores an incident matrix and a weight matrix of the created graph object. Representation of the graph in the form of an array is one of the most basic and at the same time easy-to-read forms of saving the graph structure. Incident and weighting matrices have been included in Figure 13.
The second of the offered forms of the visualization of the basically generated graph is its structure in graphic form. Figure 14 shows a graph that can be freely moved by manipulating individual vertices or the whole object. Additionally, it is possible to change the scale of the observed structure, and there is also a handy panel at your disposal when you hover the cursor over the selected edge. The handy panel contains information about vertex numbers that are connected by a specific edge. The user can also check the weight of such connection.



Incidence Matrix

	1	2	3	4	5	6	7	8	9	Suma
1	0	1	1	0	1	1	1	1	0	6
2	1	0	0	0	1	1	1	1	1	6
3	1	0	0	0	1	1	1	1	0	5
4	0	0	0	0	1	1	0	1	0	3

Weight Matrix

	1	2	3	4	5	6	7	8	9
1	6,3	7,86	5,69	9,07	7,98	7,87	5,35	7,47	9,7
2	7,86	7,97	9,17	8,67	5,02	5,77	5,49	7,43	6,9
3	5,69	9,17	8,52	9,57	7,17	5,37	7,14	7,17	8,5

Figure 13. Incident matrix and randomly generated graph weights

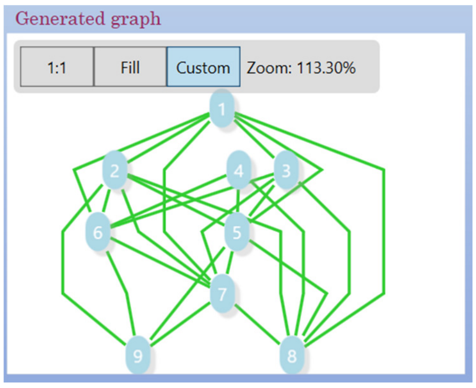


Figure 14. Panel displaying the base-generated graph in graphic form

The last section into which the whole user interface is divided are the panels responsible for presenting the progress related to the genetic algorithm being performed. This section includes the following objects:
- an indicator of the progress of the genetic algorithm,
- visualization of the best individual from the population (in a given iteration) in the form of a graphical representation of the graph tripartition,
- a table of details of the best individual in a given iteration containing a detailed clustering of vertices into appropriate groups,
- minimum value fluctuation graphs and average value of the adaptation function in individual iterations of the genetic algorithm,
- chart of the Pareto compromise set together with the best individual in a given iteration, which this time is presented as a point in two-dimensional space.

It should be noted that all elements of the user interface section currently under discussion are refreshed every iteration of the genetic algorithm and contain transient values which may or may not be temporary, occurring only during one iteration of the genetic algorithm. The first object discussed in this part of the user panel is an indicator to inform about the progress of the genetic algorithm. The indicator has a purely informative role and provides information on how many of the planned iterations of the genetic algorithm have already been performed. An example of a progress indicator is shown in Figure 15.

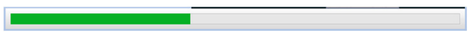
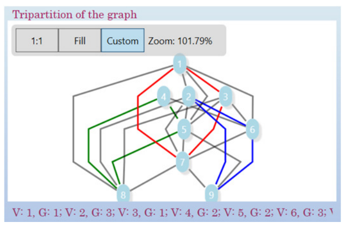


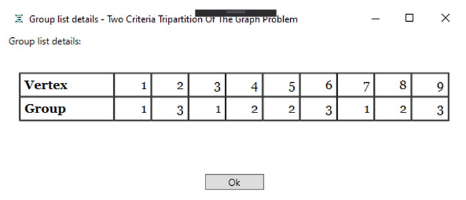
Figure 15. Progress indicator of the genetic algorithm being performed

The next panel that is refreshed every iteration is a panel visualizing the graph tripartition for the best individual in the iteration. Individual edges connecting two vertices from the same group are marked with separate colours. Colours of groups are red, green and blue. The edges that connect two vertices belonging to different groups are shown in grey. The aim of our genetic algorithm is to minimize the number of such edges, thus obtaining the smallest possible sum of their weights. Figure 16 shows a panel visualizing the graph tripartition for the best individual from a given iteration of the evolutionary algorithm. Additionally, from the level of this panel it is possible to call up a window containing detailed assignment of vertices to particular groups (shown in Figure 17) by pressing anywhere on the hierarchy of vertices placed below the graph.



V: 1, G: 1; V: 2, G: 3; V: 3, G: 1; V: 4, G: 2; V: 5, G: 2; V: 6, G: 3;

Figure 16. Visualization of the tripartition of the best individual


Group list details - Two Criteria Tripartition Of The Graph Problem
Ok

Group list details:

Vertex	1	2	3	4	5	6	7	8	9
Group	1	3	1	2	2	3	1	2	3

Figure 17. Window for assigning individual vertices to specific groups of the graph tripartition

The next panel contained in the described section of data updated every iteration of the evolutionary algorithm are the minimum fluctuation and average value of the adaptation function of individuals (fitness) charts over all previous iterations. With the help of these diagrams, it is easy to see the trend of changes and to capture the iterations that have brought about a noticeable improvement in solving the problem of the two-criteria graph tripartition in genetic approach. The charts are shown in Figure 18.

The last, but one of the most important elements of the discussed section of the user interface panel of the created application is the Pareto compromise set graph, which is refreshed every iteration of the genetic algorithm. Additionally, under the graph there are separated coordinates on a two-dimensional surface that define the best individual in a given iteration. On the horizontal axis X of the Pareto diagram there is a criterion of the number of edges that connect vertices assigned to separate groups of the graph's tripartition (these are intersections between groups), while the vertical axis Y is responsible for the criterion of the sum of weights of edges connecting elements from two different sets of tripartition. Pareto front is shown in Figure 19.

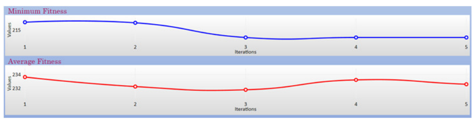


Figure 18. Charts of changes in the value of the adaptation function over the iteration space

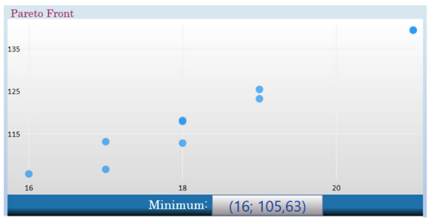
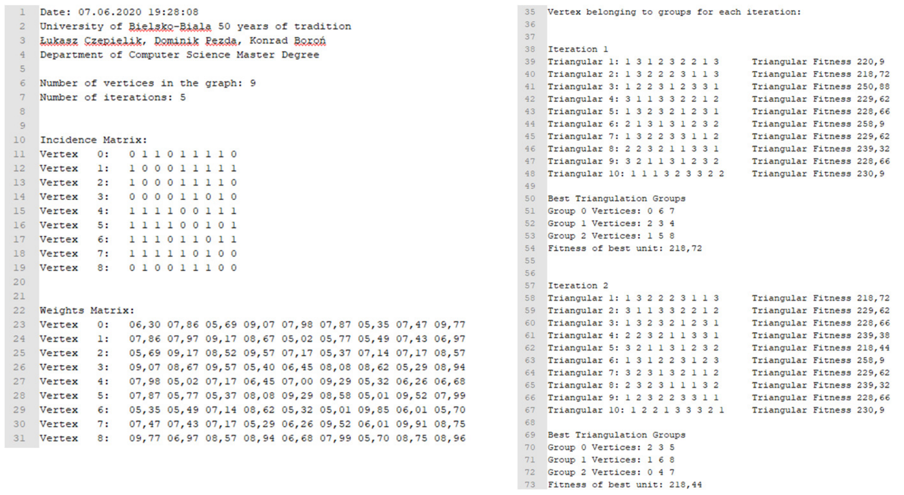


Figure 19. Graphic representation of Pareto Front

7. Generated output file

In the case where a user of an application solving a two-criteria graph tripartition problem would like to save the results obtained in particular iterations of the genetic algorithm, the functionality of generating an output file in .txt format may be useful. This document allows you to record all the most important information resulting from the evolutionary algorithm.
The structure of the generated file contains the following elements:
- date of the document generation,
- the authors of the applications and the scientific unit which associates them,
- data of the base-generated graph,
- graph tripartition data of each individual from a given iteration together with the calculated adaptation function and the best individual selected. An example of the generated result file is shown in Figure 20.



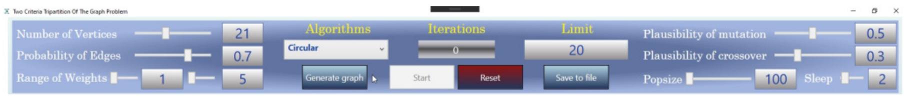
```
Date: 07.06.2020 19:28:08
University of Bielsko-Biala 50 years of tradition
Łukasz Czepielik, Dominik Pezda, Konrad Boroń
Department of Computer Science Master Degree

Number of vertices in the graph: 9
Number of iterations: 5


Incidence Matrix:
Vertex   0:   0 1 1 0 1 1 1 1 0
Vertex   1:   1 0 0 0 1 1 1 1 1
Vertex   2:   1 0 0 0 1 1 1 1 0
Vertex   3:   0 0 0 0 1 1 0 1 0
Vertex   4:   1 1 1 1 0 0 1 1 1
Vertex   5:   1 1 1 1 0 0 1 0 1
Vertex   6:   1 1 1 0 1 1 0 1 1
Vertex   7:   1 1 1 1 1 0 1 0 0
Vertex   8:   0 1 0 0 1 1 1 0 0


Weights Matrix:
Vertex   0:   06,30 07,86 05,69 09,07 07,98 07,87 05,35 07,47 09,77
Vertex   1:   07,86 07,97 09,17 08,67 05,02 05,77 05,49 07,43 06,97
Vertex   2:   05,69 09,17 08,52 09,57 07,17 05,37 07,14 07,17 08,57
Vertex   3:   09,07 08,67 09,57 05,40 06,45 08,08 08,62 05,29 08,94
Vertex   4:   07,98 05,02 07,17 06,45 07,00 09,29 05,32 06,26 06,68
Vertex   5:   07,87 05,77 05,37 08,08 09,29 08,58 05,01 09,52 07,99
Vertex   6:   05,35 05,49 07,14 08,62 05,32 05,01 09,85 06,01 05,70
Vertex   7:   07,47 07,43 07,17 05,29 06,26 09,52 06,01 09,91 08,75
Vertex   8:   09,77 06,97 08,57 08,94 06,68 07,99 05,70 08,75 08,96
```

```
Vertex belonging to groups for each iteration:


Iteration 1
Triangular 1: 1 3 1 2 3 2 2 1 3      Triangular Fitness 220,9
Triangular 2: 1 3 2 2 2 3 1 1 3      Triangular Fitness 218,72
Triangular 3: 1 2 2 3 1 2 3 3 1      Triangular Fitness 250,88
Triangular 4: 3 1 1 3 3 2 2 1 2      Triangular Fitness 229,62
Triangular 5: 1 3 2 3 2 1 2 3 1      Triangular Fitness 228,66
Triangular 6: 2 1 3 1 3 1 2 3 2      Triangular Fitness 258,9
Triangular 7: 1 3 2 2 3 3 1 1 2      Triangular Fitness 229,62
Triangular 8: 2 2 3 2 1 1 3 3 1      Triangular Fitness 239,32
Triangular 9: 3 2 1 1 3 1 2 3 2      Triangular Fitness 228,66
Triangular 10: 1 1 1 3 2 3 3 2 2     Triangular Fitness 230,9

Best Triangulation Groups
Group 0 Vertices: 0 6 7
Group 1 Vertices: 2 3 4
Group 2 Vertices: 1 5 8
Fitness of best unit: 218,72


Iteration 2
Triangular 1: 1 3 2 2 2 3 1 1 3      Triangular Fitness 218,72
Triangular 2: 3 1 1 3 3 2 2 1 2      Triangular Fitness 229,62
Triangular 3: 1 3 2 3 2 1 2 3 1      Triangular Fitness 228,66
Triangular 4: 2 2 3 2 1 1 3 3 1      Triangular Fitness 239,38
Triangular 5: 3 2 1 1 3 1 2 3 2      Triangular Fitness 218,44
Triangular 6: 1 3 1 2 2 3 1 2 3      Triangular Fitness 258,9
Triangular 7: 3 2 3 1 3 2 1 1 2      Triangular Fitness 229,62
Triangular 8: 2 3 2 3 1 1 1 3 2      Triangular Fitness 239,32
Triangular 9: 1 2 3 2 2 3 3 1 1      Triangular Fitness 228,66
Triangular 10: 1 2 2 1 3 3 3 2 1     Triangular Fitness 230,9

Best Triangulation Groups
Group 0 Vertices: 2 3 5
Group 1 Vertices: 1 6 8
Group 2 Vertices: 0 4 7
Fitness of best unit: 218,44
```

Figure 20. An example of a generated result file

8. Analysis of the operation of the programme

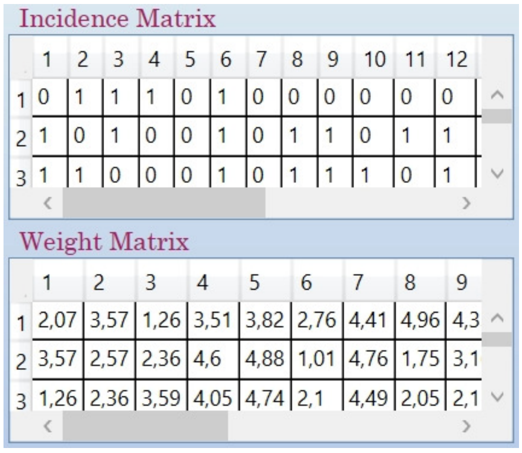
In order to start solving the problem of the two-criteria graph tripartition in an evolutionary approach, first the parameters for the procedure of generating a random, basic output graph must be defined. Additionally, the settings for the genetic algorithm must be adjusted, the graphical presentation procedure must be selected, and the stop condition of the algorithm must be defined in the form of the number of iterations the algorithm must perform. In the test scenario, which will be used to present the way the application is run and the correct interpretation of the results returned by the application, the following parameters determining the randomly generated graph and evolutionary algorithm coefficients will be used, which are presented in Figure 21.



Number of Vertices: 21
Probability of Edges: 0.7
Range of Weights: 1 – 5
Algorithms: Circular
Iterations: 0
Limit: 20
Plausibility of mutation: 0.5
Plausibility of crossover: 0.3
Popsize: 100
Sleep: 2

Figure 21. Application work rates for the test scenario

After setting all the necessary parameters (changing the default settings), you can proceed to generate the base graph by pressing the "Generate graph" button once. After this operation, the panel presenting incident matrices and weights, as well as the panel storing an interactive visual version of the drawn graph will be filled with relevant data. At this stage, all the application control factors have been defined, and also a randomly generated graph was created and presented in two forms. The individual incident matrices and weights for the test scenario can be analysed in Figure 22, while the presentation of the basically generated graph is shown in Figure 23.



Incidence Matrix

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	0	1	0	0	0	0	0	0
2	1	0	1	0	0	1	0	1	1	0	1	1
3	1	1	0	0	0	1	0	1	1	1	0	1

Weight Matrix

	1	2	3	4	5	6	7	8	9
1	2,07	3,57	1,26	3,51	3,82	2,76	4,41	4,96	4,3
2	3,57	2,57	2,36	4,6	4,88	1,01	4,76	1,75	3,1
3	1,26	2,36	3,59	4,05	4,74	2,1	4,49	2,05	2,1

Figure 22. Incident and weights matrix for the test scenario

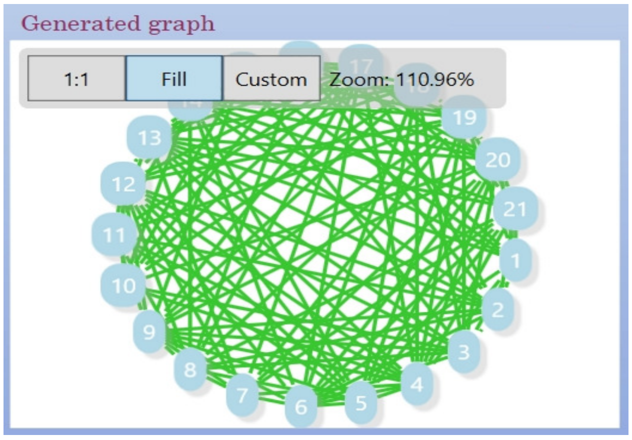


Figure 23. Base generated random graph presented in graphic - interactive form

After all the above steps have been taken, you can proceed to run the appropriate evolutionary algorithm, solving the problem of the graph tripartition in two-criteria approach. This operation is done by pressing the button marked on the user panel as "Start". In the next steps you should observe the changes that take place in subsequent iterations. During breaks between successive iterations of evolutionary algorithm panels will be refreshed:
- visualization in the form of interactive graph tripartition for the best individual from a given iteration,
- a window containing a detailed distribution of the assignment of individual vertices of the base-generated graph to groups defined within the tripartition,
- minimum value fluctuation charts and the average function of adaptation of individuals (fitness) over successive iterations,
- graphs of the Pareto compromise set with an annotation of the coordinates (two criteria - the number of edges connecting vertices from two different groups and the sum of their weights) for the best adapted individual in a given iteration.

Once the execution of the genetic algorithm for the test scenario has been completed in the next step, a review can be made of how the mean and minimum (corresponding to the best individual in the iteration) of the adaptation function changed through all the iterations that have been done. These data are contained in the fluctuation charts in Figure 24.

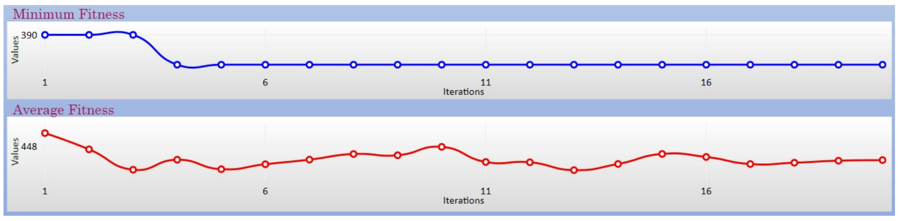
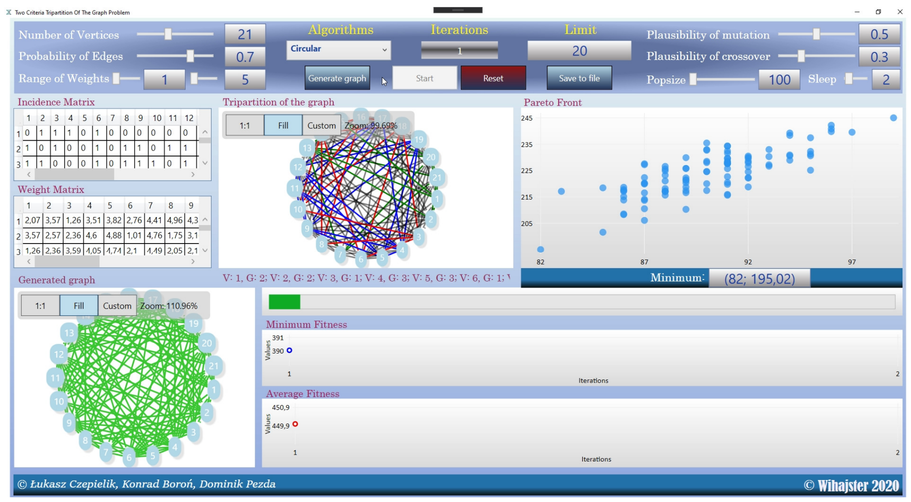


Figure 24. Fluctuation charts of adaptation function values over the iteration area - minimum value (best individual) and average

By performing a thorough analysis of the graphs from Figure 24, it can be seen that the value of the obtained solution to the graph tripartition problem in two-criteria terms using a genetic algorithm, it was systematically improved by finding in subsequent iterations an increasingly better individual (better adapted) from individuals in previous generations of the evolutionary algorithm. The observed fact proves that the implemented algorithm works correctly. The whole application window in the first and last iteration of the genetic algorithm together with all the component panels that were successively analysed in the test scenario are shown in Figures 25 and 26.



Two Criteria Tripartition Of The Graph Problem

Number of Vertices 21
Probability of Edges 0.7
Range of Weights 1 5
Algorithms: Circular
Iterations: 1
Limit: 20
Plausibility of mutation 0.5
Plausibility of crossover 0.3
Popsize 100
Sleep 2

Incidence Matrix

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	0	1	0	0	0	0	0	0
2	1	0	1	0	0	1	0	1	1	0	1	1
3	1	1	0	0	0	1	0	1	1	1	0	1

Weight Matrix

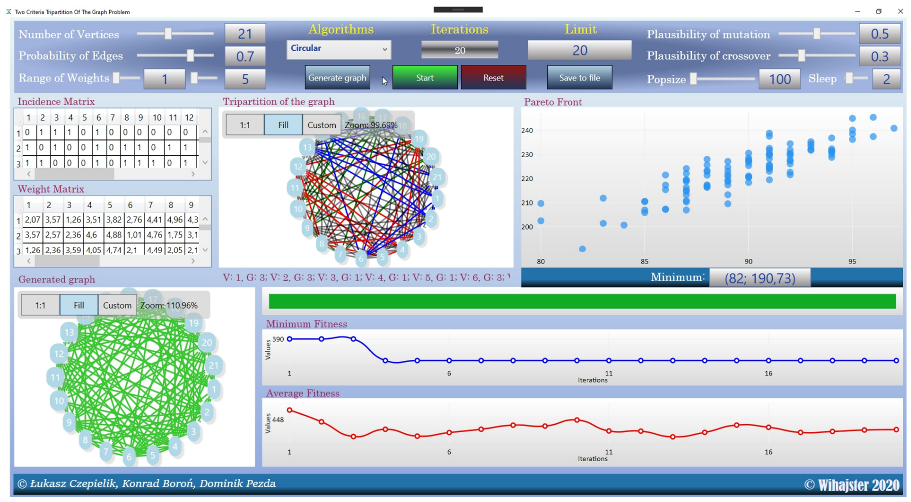
	1	2	3	4	5	6	7	8	9
1	2,07	3,57	1,26	3,51	3,82	2,76	4,41	4,96	4,3
2	3,57	2,57	2,36	4,6	4,88	1,01	4,76	1,75	3,1
3	1,26	2,36	3,59	4,05	4,74	2,1	4,49	2,05	2,1

V: 1, G: 2; V: 2, G: 2; V: 3, G: 1; V: 4, G: 3; V: 5, G: 3; V: 6, G: 1;

Minimum: (82; 195,02)

© Łukasz Czepielik, Konrad Boroń, Dominik Pezda © Wihajster 2020

Figure 25. Full application window - first iteration of the test scenario



Two Criteria Tripartition Of The Graph Problem

Number of Vertices 21
Probability of Edges 0.7
Range of Weights 1 5
Algorithms: Circular
Iterations: 20
Limit: 20
Plausibility of mutation 0.5
Plausibility of crossover 0.3
Popsize 100
Sleep 2

Incidence Matrix

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	0	1	0	0	0	0	0	0
2	1	0	1	0	0	1	0	1	1	0	1	1
3	1	1	0	0	0	1	0	1	1	1	0	1

Weight Matrix

	1	2	3	4	5	6	7	8	9
1	2,07	3,57	1,26	3,51	3,82	2,76	4,41	4,96	4,3
2	3,57	2,57	2,36	4,6	4,88	1,01	4,76	1,75	3,1
3	1,26	2,36	3,59	4,05	4,74	2,1	4,49	2,05	2,1

V: 1, G: 3; V: 2, G: 3; V: 3, G: 1; V: 4, G: 1; V: 5, G: 1; V: 6, G: 3;

Minimum: (82; 190,73)

© Łukasz Czepielik, Konrad Boroń, Dominik Pezda © Wihajster 2020

Figure 26. Full application window - last iteration of the test scenario

After the program is finished, you can take a closer look at the last population of the genetic algorithm, which is presented in the Pareto Front graph, which is shown in Figure 27.

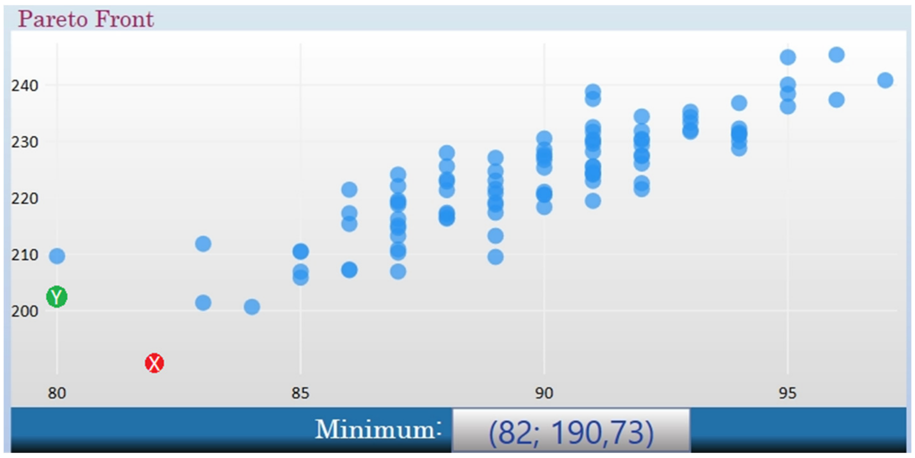


Figure 27. Pareto Front chart with marked best individuals

After a more detailed analysis of the chart presenting the evaluation of the adaptation of individuals, it can be seen that there are two points on the Pareto Front that represent the individuals in the population that are best adapted. For a better explanation, the individuals on the Pareto Front have been replaced with "X" and "Y" signs. The algorithm has indicated the best adapted individual for both criteria, that is, the X individual, but the Y individual is better adapted for the criterion of the number of edges. In order to better illustrate the dominance of X and Y in relation to other individuals in the current population, the cones of dominance can be plotted on the graph. These cones of domination are shown in Figure 28.

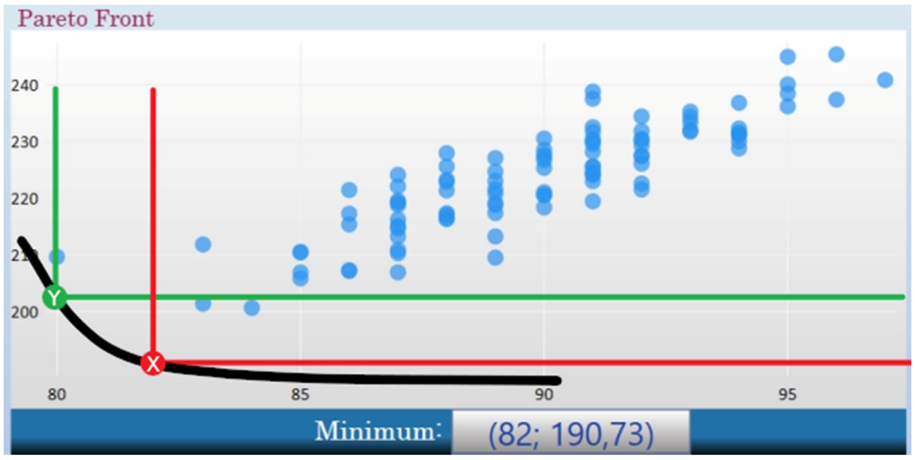


Figure 28. Pareto Front chart with marked dominance cones

After proper determination of the cones of domination for the points that are on the Pareto Front, it is possible to obtain exact information which individuals from the population are dominated by specific points from the Pareto Front in regards to a specific criterion. According to the information contained in Figure 28 individual X dominates all the remaining in terms of the criterion of the sum of edge weights, while individual Y dominates all the remaining in terms of the number of edges. In such a situation the algorithm must have properly implemented mechanisms that will indicate which criterion is more important to select the best individual, or must use some arbitrary selection methodology. In the observed example, the sum of edge weights was considered as the leading criterion, so that the individual X was selected as the best adapted from the whole population.

9. Final remarks

The described programme was created as part of the “Graphs and networks in computer science” classes. The designed solution allows to effectively solve the problem of the tripartition of the graph in a two-criteria approach using a genetic algorithm. Due to the possibility of tuning the parameters determining the behaviour of the algorithm and due to the possibility of accurate observation of the processes of solving the problem of tripartition, the presented program can be a didactic basis for the above mentioned student classes. The discussed issues come from the area of mathematics, programming and graph theory.

REFERENCES

1. DEB K. *Multi-objective optimization using evolutionary algorithms*, John Wiley & Sons, Chichester, New York, 2001.
2. MICHALEWICZ Z. *Genetic algorithms + data structures = evolution programs* Springer–Verlag , Berlin, 1992.HODLER A.: Graph Algorithms, O’Reilly Media Inc., USA, 2019.
3. ARABAS J. *Wykłady z algorytmów ewolucyjnych*, WN-T, Warszawa, 2001.
4. ŻURADA J., BARSKI M., JEDRUCH W. *Sztuczne sieci neuronowe*, PWN, Warszawa, 1996.
5. WOJCIECHOWSKI J., PIEŃKOSZ K. *Grafy i Sieci*, PWN, Warszawa, 2013.
6. WILSON R. *Introduction to Graph Theory*, Prentice Hall, 2010.
7. HODLER A. *Graph Algorithms*, O’Reilly Media Inc., USA, 2019.
8. WEST D.B. *Introduction to graph theory*, Pearson, 2000, Prentice Hall of India, 2007.
9. WOJNAROWSKI J., ZAWIŚLAK S. *Evolutionary Algorithm for graph partitioning* (in Polish) in the book ”Polioptymalizacja i Komputerowe Wspomaganie Projektowania”, Editors: W. Tarnowski, T. Kiczkowiak, WN-T, Warsaw 2002.
10. ZAWIŚLAK S., PAGACZ A. *EA for Graph Theory Problems: Review of Data Structures and Operators,* Workshop on Graphs, Krynica, 2003.

11. WOJNAROWSKI J., ZAWIŚLAK S., KOZIK S., & FREJ G. (2003). *K-partitioning of graph by means of evolutionary algorithm.* Badania Operacyjne i Decyzje, (3), 91-107.
12. ZAWIŚLAK S., & FREJ G. (2003). *An influence of parameters of the evolutionary algorithm applied for the graph k-partitioning problem.* Studia Informatica, 24(4), 165-187.