CS4106 - Machine Learning: Methods and Applications, CSIS/UL
Genetic algorithm

Lecturer: Dr Malachy Eaton
Project Coordinator: Davi Monteiro

March 2020

1 Introduction

In this project we are going to look at an open problem in the field of Graph Visualisation. Graph visualisation is categorised as part of a bigger group of research called data visualisation or information visualisation. One of the challenging aspects of visualising a graph is to minimize the number of edge-crossings. In this project we are introducing a Genetic Algorithm that can be used in any iterative Graph Visualisation algorithm in order to reduce the number of edge-crossings [4, 5, 6].

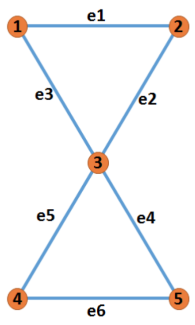
2 Graph

Graphs (sometimes called Networks) are comprised of two sets (the node set and the edge set). $G = (N, E)$ Graphs can be directed, but in this project we target undirected and connected graphs.

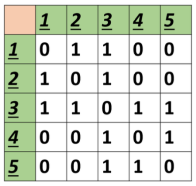
3 How to represent a graph

A graph can be represented using its connectivity information such as:

1. Adjacency matrix: It is a $|N| \times |N|$ matrix in which each cell represents the pairwise connectivity between the corresponding node in column and row, see Figure 1(b).
2. Adjacency list: It is usually an unordered list. Each item in the list represents an edge in the graph and it is comprised of two heads of the edge (two nodes), see Figure 1(c).

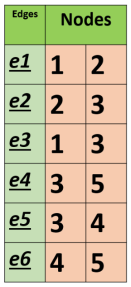


(a)



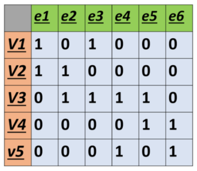
	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	0
3	1	1	0	1	1
4	0	0	1	0	1
5	0	0	1	1	0

(b)



Edges	Nodes	
e1	1	2
e2	2	3
e3	1	3
e4	3	5
e5	3	4
e6	4	5

(c)



	e1	e2	e3	e4	e5	e6
V1	1	0	1	0	0	0
V2	1	1	0	0	0	0
V3	0	1	1	1	1	0
V4	0	0	0	0	1	1
v5	0	0	0	1	0	1

(d)

Figure 1: Different graph representations

3. Incidence matrix: It is a $|N| \times |E|$ matrix in which each cell in the matrix represents the relationship between an edge and a node; E.g. if node 3 is connected to edge 5 then the corresponding cell (3, 5) is equal to 1, see Figure 1(d).

A graph also can be represented using node-link diagram which is called a visual representation of a graph, see Figure 1(a). In this project we are going to read input data related to a graph and use a Genetic Algorithm in order to prepare it for node-link visualisation.

4 Graph drawing using genetic algorithm

In this project we are going to draw any undirected graph in a 2D environment within a circle which is divided by $|N|$ ($|N|$ is the number of nodes). The result of the Genetic Algorithm can then help to start the actual graph drawing using some Force-Directed (or iterative) graph drawing algorithms [2] which they would benefit from an aesthetic initial layout. A detailed discussion about Force-Directed algorithms is beyond the scope of this project; we only focus on how to create a layout that implies the general structure of a graph to some extent. In this project we aim at ending up with an ordering (an ordering is referred to a sequence of node indexes) that can be placed on a closed circle (See Figure 8).

4	17	13	8	10	1	9	15	16	2	3	5	6	11	12	7	14

Figure 2: An ordering of 17 nodes

5 Genetic algorithm

The Genetic Algorithm (GA) belongs to the larger class of Evolutionary Algorithms (EA) which generates solutions to optimisation problems using techniques inspired by natural evolution, such as inheritance, mutation and selection. GAs are mostly used for problems in which their search domain is increasing exponentially as the input increases linearly; e.g. for a problem with size n, n! number of searches are required to find the optimum solution. If n is very large then the time required to look at all the search domain could be a few years, even with the fastest CPUs. A GA generates a number of instances from the search domain and tries to find a solution near to the optimum solution. The generated set of instances (in GA we call it a population) then undergoes three main actions: Crossover, Mutation and Selection.

6 Notation

6.1 Ordering

The candidate solutions for our project are called orderings. An ordering is simply a sequence of numbers as a one-dimensional array in the range $[1 - |N|]$ where $|N|$ is the total number of nodes in a given graph. The size of each ordering must be exactly $|N|$. An ordering is not allowed to have any repeated value or miss any value from the range $[1 - |N|]$. For instance, if there are in total 17 nodes then one possible ordering would look like the array in Figure 2.

6.2 Population/Population size

A genetic algorithm initially starts with a predefined number of different randomly generated orderings (the population). The initial population of orderings is also called the first generation or generation 0. The concept of a generation will be discussed shortly.

6.3 Selection

Selection is a GA technique which tries to smartly select the better orderings from a population or current generation for Crossover and Mutation.

6.4 Crossover

Crossover is a GA technique which randomly picks and removes two different orderings (parents) from the current population and creates two new orderings (children) and inserts them into a new population (next generation). In this project we will be using AP crossover (Alternating-Position Crossover). This will be discussed later.

6.5 Mutation

Mutation is another GA technique that randomly picks and removes one ordering from the current population and applies small change to the ordering and inserts it into the next generation. In this project we will be using Exchange Mutation (EM) which will also be discussed later.

6.6 Reproduction

Reproduction randomly picks and removes one ordering from the current population and inserts it into the next generation.

6.7 Fitness function

The fitness function indicates how good an ordering is. The indicator for the goodness of an ordering is called **the fitness cost**. In this project, the sum of the edges' lengths for a given ordering is considered as a **fitness cost**. Shorter edges would help to reduce the number of edge crossing when the drawing environment is a closed circle.

6.8 Generation

The GA starts with an initial population (generation 0). First of all the orderings are put through the selection process, and then crossover and mutation are applied to the output of the selection process in order to fill in the next generation/population. The whole process repeats until a certain number of generations are created and then from the very last generation the best ordering based on the fitness cost is selected as the final result. The number of required generations is set by the user.

7 Project Evaluation

You are required to write a Java program in order to perform the project. The whole project has 20 marks out of your final exam.

8 How to do the Project!

This project has the following steps:

1. Creating an adjacency matrix from an edge list:
   The first task in this project is to convert the provided input file which is in the form of Edge List into an Adjacency Matrix. See Figures 1(c)

and(b) You can download the input file from Gist[1]. Note: The adjacency matrix is a symmetrical matrix with all zero on the diagonal line.

2. The project has to use JOption Pane and ask End-user to provide some values for a few parameters as follows:

   (a) Population Size (P): A positive integer value
   (b) Number of Generations (G): A positive integer value
   (c) Crossover rate (Cr): A positive integer value in the range [0,100]
   (d) Mutation rate (Mu): A positive integer value in the range [0,100]. Note: The sum of Cr and Mu must not be greater than 100.

   All of the above parameters have to be asked for and the entered values need to be validated according to their conditions. If the value of a parameter fails the validation process, an appropriate error message has to be given and the end-user is asked to re-enter the value.

3. Declare two empty 2D arrays with size $N \times P$. One is to be called Current Population the other one next Population.

4. Creating the first population:

   Once you have read and converted the edge list into an adjacency matrix, you know the size of the graph, N, (the largest value in the edge-list; Note: In Java we start from 0 so the size of the graph would be the largest value plus 1).

   The next step is to randomly generate P number of different orderings to populate the Current Population. You need to print all the orderings.

5. The following steps are repeated G times:

   (a) Selection is a process on Current Population which filters some of the low quality orderings (orderings with high fitness cost) and makes a new copy of the high quality orderings (orderings with low fitness cost). The concept of the fitness function will be discussed shortly. The selection process sorts all the population members from the best towards worst. The population then is divided into 3 different sections. s1, s2 and s3; s1 has the best orderings, s2 has the average orderings, and s3 has the worst. The last section, s3, is then replaced by s1 so then there are two copies of the best orderings and one copy of the average orderings. Figure 3 illustrates the general steps of a selection process. The population at the beginning has a set of orderings with different fitness costs (the colors here are the indicators of fitness cost, the darker red the better ordering with lower fitness cost). All the orderings are sorted from the best towards the worst

[1] `https://shorturl.at/fAFO7`

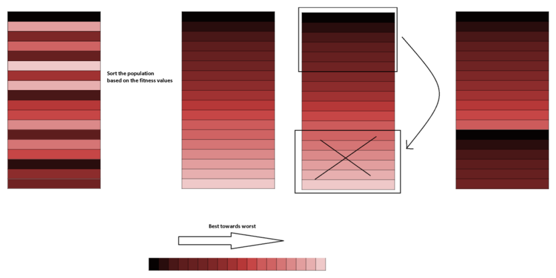


Figure 3: The general steps of a selection process

according to their fitness cost. The population is divided by three and the last section is removed and replaced by the best section.

(b) Repeat the next step (i) until all of the orderings in the Current Population are removed and Next Population is filled in. The Next Population is then called New Generation.

   i. Generate a random number P r in the range of [0, 100]:

      IF $Cr \geq Pr$: (**Crossover**)
      Select and remove two different orderings (Parents) randomly from the Current Population and perform the Cross-Over function on them. The result of the Cross-Over is two new orderings (Children) that are added into the Next Population. The Cross-Over function generates a random number Cp (Cutting-Point) in the range of $[1, |N| - 2]$. The two randomly selected orderings are split from Cp point and the first half of the two orderings are swapped so that there are two new orderings that have to be added into Next Population. Figure 4 shows the process of Crossover. The results of the Cross-Over (Children) might have some duplicated and some miss- ing numbers in which the duplicated ones have to be replaced by the missing ones so that the resultant ordering must not have any duplicated number.
      IF $Cr \leqslant Pr \leqslant (Cr + Mu)$: (**Mutation**)
      Select and remove one ordering randomly from the Current Population and perform Mutation function on it. The result of the Mutation is one ordering that is added into the Next Population. The Mutation function generates two different random numbers M1 and M2 in the range $[0, |N| - 1]$ and swap their positions. The resultant ordering is then added into the Next Population. Figure 5 shows the process of Mutation.
      IF $(Cr = Mu) \leqslant Pr$: (**Reproduction**)
      Select and remove one ordering randomly from the Current Population and add it directly into the Next Population without any change.

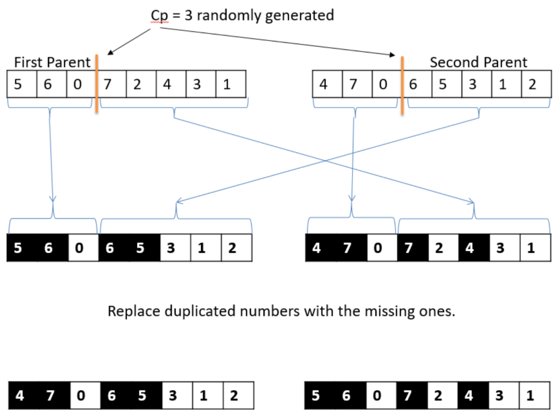


Figure 4: The general steps of a crossover process

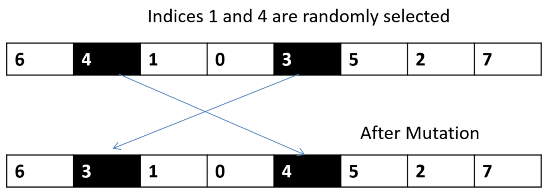


Figure 5: The general steps of a mutation process

8.1 Fitness function

Our fitness function aims at minimising the total lengths of edges in the layout. To calculate the fitness value for any given ordering one needs to simulate the given ordering on a circle with radius 1 starting from the first node to the last node. To calculate the total length of a layout we need to first find the Cartesian coordinates of the two heads of each edge and then use the Pythagoras equation. Since the nodes are placed on the border of a circle with radius 1, then it should be easy to find the coordinates of each node (x, y). To do that we introduce a constant for our graph which is called Chunk and it is calculated as follow:

$$Chunk = \frac{2 \times \pi}{|N|} \tag{1}$$

Figure 6 illustrates an example of how the members of an ordering (3, 0, 1, 6, 4, 5, 2) are placed on the border of a circle. The coordinates of all nodes in that ordering is calculated using Algorithm 1.

Algorithm 1: How to calculate the (x, y) of nodes. See Figure 6

```
for $i = 0; i < |N|; i{+}{+}$ do
    $x.Ordering[i] = cos(i \times chunk)$;
    $y.Ordering[i] = sin(i \times chunk)$;
end
```

Note: You can find all edges by finding the cells equal to 1 in the adjacency matrix. As you can see in Figure 7, different orderings lead into different layouts.

9 What your project has to do

You are expected to implement this project in Java as a group of 4 students.

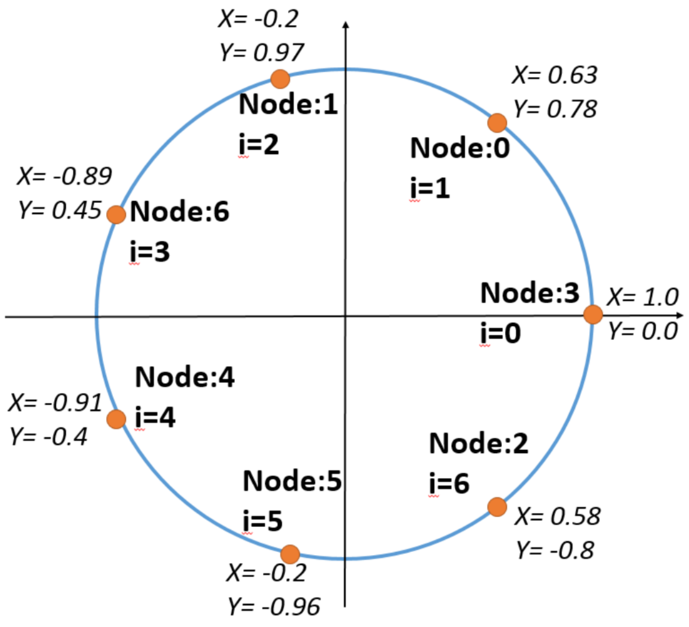
1. After reading the Edge-List, the project has to convert the Edge-List into Adjacency matrix and printed out to the Java console. For instance, for the matrix in Figure 1(b);

2. A new fitness function needs to be implemented using proposed algorithms that can be found in the literature such as [3, 7, 1, 8]. After the implementation, you have to compare the two functions in terms of the number of generations required to find a solution and the time to compute the fitness cost;

3. For each New Generation, you need to find the best ordering using the fitness function. The best ordering then needs to be visualised using the piece of code in Figure 8.

Ordering: {3,0,1,6,4,5,2}

$$\text{Chunk} = \frac{2*\pi}{|N|} = \frac{6.28}{7} = 0.89$$



$$\boxed{\begin{aligned} x &= \cos(i \,.\, chunk) \\ y &= \sin(i \,.\, chunk) \end{aligned}}$$

Figure 6: How to calculate (x, y) for nodes

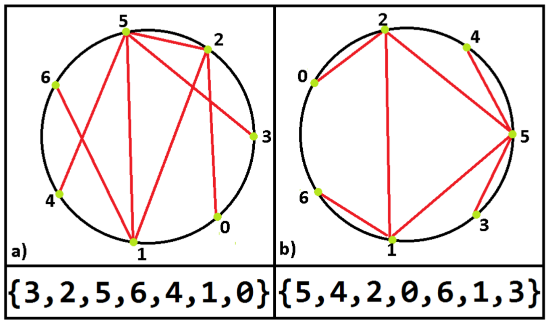
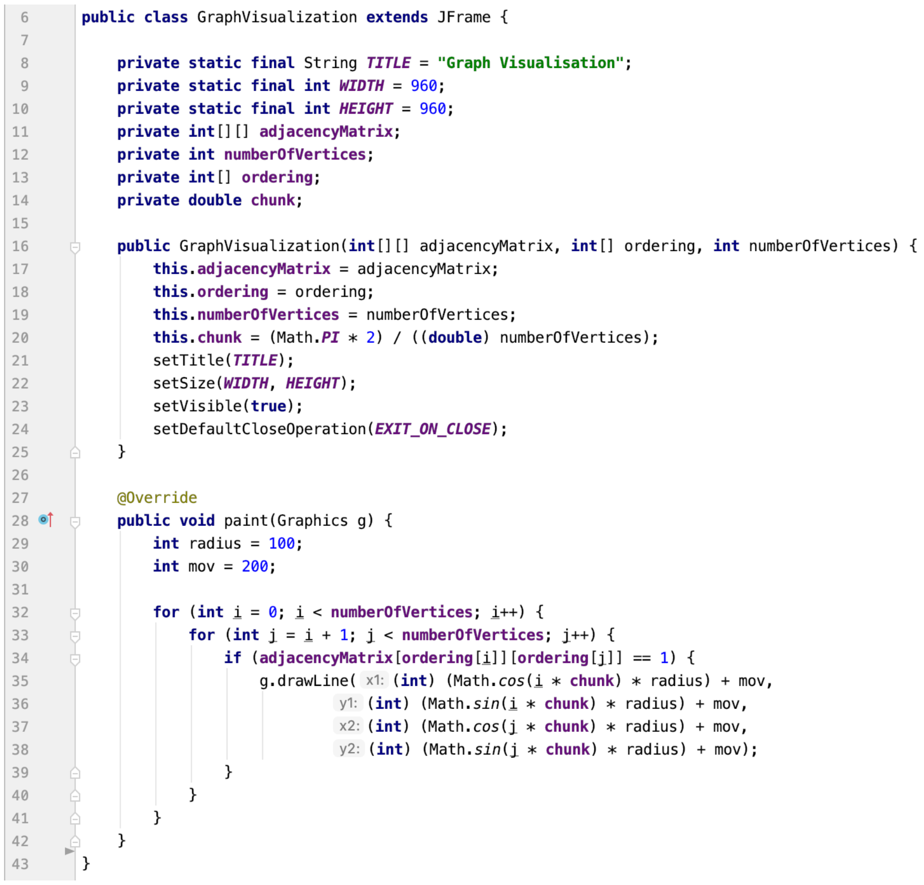


Figure 7: Two different orderings result two different layouts



```java
public class GraphVisualization extends JFrame {

    private static final String TITLE = "Graph Visualisation";
    private static final int WIDTH = 960;
    private static final int HEIGHT = 960;
    private int[][] adjacencyMatrix;
    private int numberOfVertices;
    private int[] ordering;
    private double chunk;

    public GraphVisualization(int[][] adjacencyMatrix, int[] ordering, int numberOfVertices) {
        this.adjacencyMatrix = adjacencyMatrix;
        this.ordering = ordering;
        this.numberOfVertices = numberOfVertices;
        this.chunk = (Math.PI * 2) / ((double) numberOfVertices);
        setTitle(TITLE);
        setSize(WIDTH, HEIGHT);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    @Override
    public void paint(Graphics g) {
        int radius = 100;
        int mov = 200;

        for (int i = 0; i < numberOfVertices; i++) {
            for (int j = i + 1; j < numberOfVertices; j++) {
                if (adjacencyMatrix[ordering[i]][ordering[j]] == 1) {
                    g.drawLine( x1: (int) (Math.cos(i * chunk) * radius) + mov,
                            y1: (int) (Math.sin(i * chunk) * radius) + mov,
                            x2: (int) (Math.cos(j * chunk) * radius) + mov,
                            y2: (int) (Math.sin(j * chunk) * radius) + mov);
                }
            }
        }
    }
}
```

Figure 8: A piece of code to visualise an ordering

10 Submission

You are required to write a Java program to solve the graph visualisation problem using a GA algorithm.

1. The deadline of the submission is (Saturday) 25th of April 2020 (end of week 12).
2. Only one .java file has to be submitted by one of the members of the group.
3. The name of the .java file has to be *is*12345678.*java* where 12345678 is the student ID of the person who submits the project.
4. The first few lines of the .java file has to contain the full name and the student ID of all the member in the group.
5. The submission must be made by email to *davi.monteiro@ul.ie* and the full name and student ID of all members have to be listed in the email. Please CC all the members of the group in the submission email as well.
6. Any submission that fails to run and/or compile will be graded 0.
7. Submissions after the deadline will not be evaluated.
8. The submission is marked out of 20 marks.

References

[1] A. M. S. Barreto and H. J. C. Barbosa. Graph layout using a genetic algorithm. In *Proceedings. Vol.1. Sixth Brazilian Symposium on Neural Networks*, pages 179–184, Nov 2000.

[2] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235 – 282, 1994.

[3] Timo Eloranta and Erkki Mäkinen. Timga-a genetic algorithm for drawing undirected graphs. In *Divulgaciones Matematicas*. Citeseer, 1996.

[4] Farshad Ghassemi Toosi, Nikola S. Nikolov, and Malachy Eaton. Evolving smart initial layouts for force-directed graph drawing. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1397–1398, New York, NY, USA, 2015. Association for Computing Machinery.

[5] Farshad Ghassemi Toosi, Nikola S. Nikolov, and Malachy Eaton. A ga-inspired approach to the reduction of edge crossings in force-directed layouts. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 89–90, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Farshad Ghassemi Toosi, Nikola S. Nikolov, and Malachy Eaton. Simulated annealing as a pre-processing step for force-directed graph drawing. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 997–1000, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Andrea GB Tettamanzi. Drawing graphs with evolutionary algorithms. In *Adaptive Computing in Design and Manufacture*, pages 325–337. Springer, 1998.

[8] Qing-Guo Zhang, Hua-Yong Liu, Wei Zhang, and Ya-Jun Guo. Drawing undirected graphs with genetic algorithms. In Lipo Wang, Ke Chen, and Yew Soon Ong, editors, *Advances in Natural Computation*, pages 28–36, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.