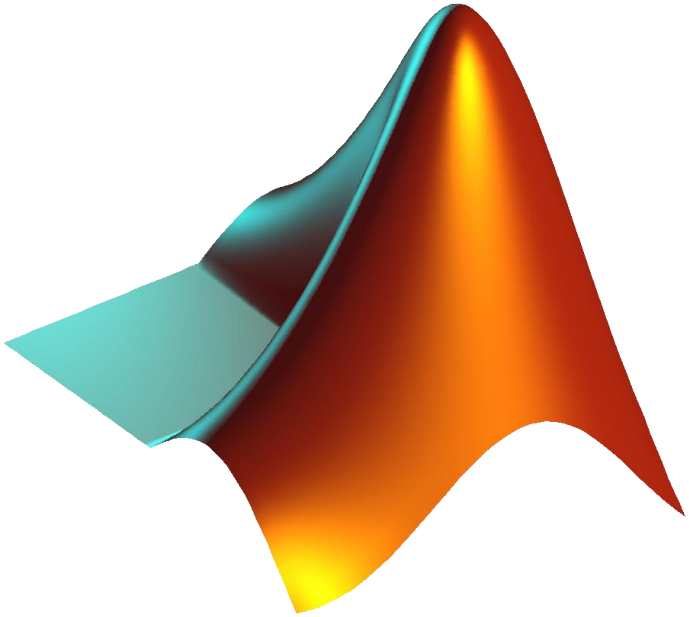
Evolutionary Programming and Genetic Algorithms

-- project --



HARABA Laura Gabriela

coordinating teacher

COCIANU Cătălina Lucia

Project content

1. Problem statement
2. Representation method in the solution space and fitness function
3. The general structure of the genetic algorithm
4. Explaining each component included in the utilized genetic algorithm for solving the problem
5. MATLAB sources that were used for solving the problem

1. Problem statement - **Solve the problem of maps coloring in a evolutive way.**

Abstract

The following project presents the results of an experimental investigation on solving the maps coloring problems. I chose as a starting point the courses from Evolutionary Programming and Genetic Algorithms discipline for solving this problem. A simulating problem is presented together with the model and the input data and the main goal is to determine the correct output data.

The idea of evolutionary calculation is solving the problems of experiment-error type. In other words, in a given environment, all individuals from a certain population enter the competition to survive and to reproduce themselves across generations. The ability of the individuals to achieve these goals in the environments in which they are living is corelated with their chances of surviving and multiplication and this is what determines the evolution of the population in time.

The principle on which I solve the problem is the following: for a given population of individuals, the influence of the environment determines a process of natural selection (being induced by the adaptability of each individual to the environment), which has as an effect the global growth of the population quality, expressed through the fitness function. In this process I included also the main elements of an evolutionary system: variation operators (recombination and mutation) which assure the necessary diversity for creating individuals with new characteristics and selection which force the growth of the individuals' quality from a population.

2. Representation method in the solution space and fitness function - ***About the problem***

I started to search for a way of searching this problem by reading the Evolutionary Programming and Genetic Algorithms book, reading interesting articles on the internet and watching an interesting YouTube tutorial. So, the statement is that every map can be colored using four colors, such that two neighboring countries are different colors for not creating confusions like the below one:

In other words, I am saying that there are no maps that need more than five colors, only if you want to differentiate each region on the map. I initially thought that it would be more complicated to color a map because you will need more colors, but apparently all maps can be done with just four colors (maximum) or even better.

This problem was unsolved for a long time, for 125 years. Finally, in the 1970s, K. Appel and W. Haken found a method to solve the problem, even if it was a little controversial at that time. Apparently, there was a guy who was trying to color in the countries of Britain. He suspected he could do it using four colors and he mentioned it to his brother, and his brother was a math student, and his brother mentioned it to his lecturer who was a man called Augustus de Morgan, the famous mathematician, and, well, it was de Morgen who spread the problem around. In order to prove it, we wither need to show that every map can be done with four colors or better, or we just need to find one example that needs five colors. If you try the method with the map of the world, you will only need four colors, the same happens with the United states map or even with the Romania map. Of course, it depends also on the complexity of the map.

The solution of map coloring can be represented in a chromosomal genetic structure, so that the "c" number of colors will be expressed through "c" alleles.

For example, red, green, blue … etc. are represented through 1,2,3, …, c, where $c \in \mathbb{N}$. Each region will be represented by a gene. For example, a feasible solution for coloring all 28 countries from the European Union, including Romania, can be represented through a chromosome with 28 genes.

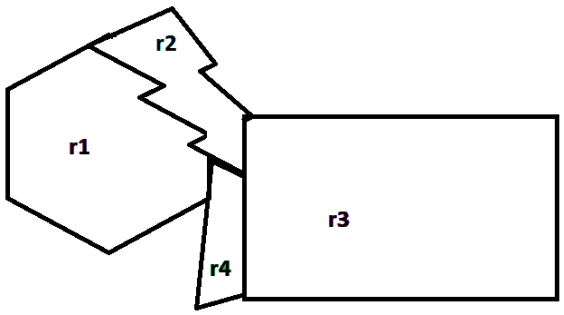
The **A** chromosome will represent the coloring of a certain **n** number of regions and it is expressed like this:

$$A = (c_1, c_2, c_3, \ldots, c_n),\ c_i \in \{1, 2, \ldots, c\},\ i = \overline{1, n}$$

Information about the adjacency of the regions can be kept in a **n x n** matrix, knowing as the adjacency matrix M:

$$M = \begin{pmatrix} m_{11} & \ldots & m_{1n} \\ \ldots & \ldots & \ldots \\ m_{n1} & \ldots & m_{nn} \end{pmatrix},\ m_{ij} = \begin{cases} 1, if\ it\ is\ adjacent\ j \\ 0, otherwise \end{cases}$$, where i,j = $\overline{1, n}$ cu $i \neq j$

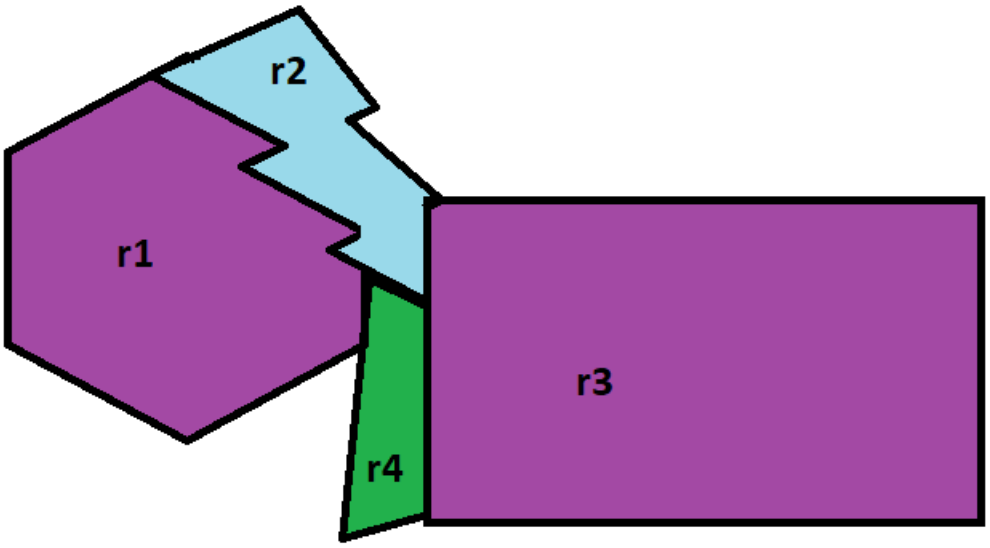
An example is represented below figure, where r1 region is adjacent for the r3 region, meaning that they have a common border.



By having the example from the previous figure, for n=4 (the number of regions) we have the next adjacency matrix m.

	r1	r2	r3	r4
r1	0	1	0	0
r2	1	0	1	1
r3	0	1	0	1
r4	0	1	1	0

A feasible solution is represented as A = (1, 2, 1, 3) and it is colored below:



3. The general structure of the genetic algorithm

Each time we found values that are equal to 1, it means that we find a way to color the map. The process is made of:

1. Start with a **"c"** number of color and **"n"** countries for generating the initial population pop = (c_1,c_2,c_3, …, c_n), each country being associated with a color.
2. Consider the objective function (fitness function) to take into consideration the number of conflicts that we have to face when we try to color the map (to chose the right color). With every generation we try to reduce the number of conflicts to 0 – which will be the valid solution.

   The fitness function corresponds to the following sum of conflicts:

   $f = \sum_{i=1}^{n} \sum_{j=i+1}^{n} m_{ij}$, if $i \neq j$, $i,j = \overline{1,n}$, $m_{ij} = 1$ and $c_i = c_j$

3. The following steps construct the algorithm for map coloring:
   a. Randomly generate the initial population with a number of **"dim"** individuals, where "**dim"** is the dimension of the population
   b. for k = 0 : number of generation
      i. select the parents
      ii. apply the recombination algorithm with the recombination probability
      iii. apply the mutation algorithm with the mutation probability

4. Explaining the components with MATLAB sources

I. **The objective function** has as scope the minimization of the conflicts between the regions with the same color and it is computes as:

```
function[sum]=objectiveFunction(m,n,x)
sum=0;
for i=1:n
   for j=i+1:n
      if m(i,j)==1 && x(i)==x(j)
         sum=sum+1;
      end
   end
end
end
```

II. At every moment in time the population is formed by "dim" individuals. **The initial population** is randomly generated as:

```
function[pop]=initialPop(m,dim,t,c)

% a population of dim dimension represented by c
% dim = the number of chromosomes
% t is the number of genes in a chromosome (no. of columns)

pop=zeros(dim,t);
for i=1:dim
   for j=1:t
      pop(i,j)=unidrnd(c);
   end
   x=pop(i,:);
   pop(i,t+1)=objectiveFunction(m,t,x);
end
end
```

For each individual, on the last position, is the value of the fitness function.

III. **Parent selection** is done through the following procedure: by **dim** times are chosen pairs of chromosomes from the current population and the best one is selected. This selection is done by comparing the fitness function values for every individual.

```
function[parents]=parentSelection(pop)

#the tournament selection will be applied

[dim,n]=size(pop);
parents=zeros(2,n);
for i=1:dim
   p1=unidrnd(dim);
   p2=unidrnd(dim);
   while(p1==p2)
      p2=unidrnd(dim);
   end
   if(pop(p1,n) <= pop(p2,n))
      parents(i,:)=pop(p1,:);
   else
      parent(i,:)=pop(p2,:);
   end
end
end
```

IV. **Recombination** is realized for each pair of individuals that are on random positions and that were not chosen before in the current process of recombination – x and y from the current population – and c1 and c2 offspring are obtained through the uniform recombination.

Each gene is evaluated independently and if the value of r is lower than the value of the probability of recombination, then the first descendent inherits the value of the gene from the second parent and the second descendent from the first parent. Of course, on the last position from the new individual the fitness function is computed.

```
function [c1,c2]=uniformRecomb(x,y,m,recombProb)
c1=x;
c2=y;
[~,n]=size(x);
for i=1:n-1
   r=unifrnd(0,1);
   if r<recombProb
      c1(i)=y(i);
      c2(i)=x(i);
   end
end
c1(n)=objectiveFunction(m,n-1,c1);
c2(n)=objectiveFunction(m,n-1,c2);
end
```

```
function [newPop] = recombination(pop,m,recombinationProb)

newPop=pop;
[dim,~]=size(pop);
list=[-1 -1];
for i=1:dim/2
   poz=unidrnd(dim, [1 2]);
   i=min(poz);
   j=max(poz);
   if((i<j) && ~ismember([i j],list,'rows'))
      list=[list;[i j]];
      disp('parents:');
      disp(pop(i,:));
      disp(pop(j,:));
      [c1,c2]=uniformRecomb(pop(i,:),pop(j,:),m,recombinationProb);
      disp('Descendants: ');
      disp(c1);
      disp(c2);
      newPop(2*1-1,:)=c1;
      newPop(2*1,:)=c2;
   end
end
end
```

If a pair of parents is not selected, it is kept for the next generation.

V. **Mutation operator** is applied with the mutation probability of an individual and it resets all alleles of a gene randomly.

```
function [newPop]=mutation(pop,m,c,mutationProb)
[dim,n]=size(pop);
for i=1:dim
   x=pop(i,:);
   disp('The mutation was applied for: ');
   disp(x);
   y=randomlyResetMutation(x,mutationProb,m,c);
   disp('The new individual: ');
   disp(y);
   newPop(i,:)=y;
end
end
```

```
function [y]=randomlyResetMutation(x,mutationProb,m,c)
y=x;
n=length(x);
for i=1:n-1
   r=unifrnd(0,1);
   if(r<pm)
      y(i)=unidrnd(c);
   end
end
y(n)=objectiveFunction(m,n-1,y);
end
```

VI. **The algorithm** is presented below. It will stop when the number of generations (iterations) is exceeded.

```
function[]=maps(dim,countriesNo,coloursNo,k,recombProbability,mutationProb)
m=importdata('matrix.txt');
disp('The adiacent matrix of the map is:');
disp(m);
pop=initialPop(m,dim,countriesNo,coloursNo);
disp('The initial population for map coloring is: ');
disp(pop);
n=countriesNo+1;
generations=zeros(dim,n);
objective=zeros(1,k);
for i=1:k
   [newPop]=recombination(pop,m,coloursNo,mutationProb,recombProbability);
   objective(i)=min(pop(:,n));
   generations=[generations;pop(:,1:n)];
end
generations=[generations(dim+1:dim*k,1:n)];
disp('Generations: ');
disp(generations);
disp(' ');
disp('Fitness function: ');
disp(objective);
[g,~]=size(generations);
minimum=min(generations(:,n));
disp('The individuals with the best value of the fitness function: ');
for i=1:g
   if generations(i,n)==minimum
      disp(['From generation: ' num2str(round(i/dim))]);
      disp(generations(i,1:n));
   end;
end
ax=1:k;
plot(ax,objective, '*-*');
%hold on;
%plot(ax,objective);
%stairs(ax,objective);
%plot(ax,objective, '--');
hold on;
end;
```

Testing the problem by calling the *maps.m* script:

```
maps(4,4,3,10,0.7,0.8);
The adiacent matrix of the map is:
   0   1   0   0
   1   0   1   1
   0   1   0   1
   0   1   1   0
The initial population for map coloring is:
   1   1   2   1   2
   3   1   3   2   0
   2   3   2   2   1
   3   1   1   3   1
parents:
   1   1   2   1   2
   3   1   3   2   0
Descendants:
   3   1   3   2   0
   1   1   2   1   2
parents:
   3   1   3   2   0
   2   3   2   2   1
Descendants:
   2   3   2   2   1
   3   1   3   2   0
parents:
   1   1   2   1   2
   2   3   2   2   1
Descendants:
   2   3   2   2   1
   1   1   2   1   2
parents:
   3   1   3   2   0
   2   3   2   2   1
Descendants:
   2   3   2   2   1
   3   1   3   2   0
parents:
   1   1   2   1   2
   3   1   3   2   0
Descendants:
   3   1   3   2   0
```

```
   1   1   2   1   2
parents:
   1   1   2   1   2
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   1   1   2   1   2
parents:
   2   3   2   2   1
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   2   3   2   2   1
parents:
   1   1   2   1   2
   3   1   3   2   0
Descendants:
   3   1   3   2   0
   1   1   2   1   2
parents:
   1   1   2   1   2
   2   3   2   2   1
Descendants:
   2   3   2   2   1
   1   1   2   1   2
parents:
   1   1   2   1   2
   3   1   3   2   0
Descendants:
   3   1   3   2   0
   1   1   2   1   2
parents:
   3   1   3   2   0
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   3   1   3   2   0
parents:
   1   1   2   1   2
   2   3   2   2   1
Descendants:
```

```
   2   3   2   2   1
   1   1   2   1   2
parents:
   1   1   2   1   2
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   1   1   2   1   2
parents:
   3   1   3   2   0
   2   3   2   2   1
Descendants:
   2   3   2   2   1
   3   1   3   2   0
parents:
   3   1   3   2   0
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   3   1   3   2   0
parents:
   2   3   2   2   1
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   2   3   2   2   1
parents:
   3   1   3   2   0
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   3   1   3   2   0
parents:
   3   1   3   2   0
   3   1   1   3   1
Descendants:
   3   1   1   3   1
   3   1   3   2   0
parents:
   2   3   2   2   1
   3   1   1   3   1
```

```
Descendants:
   3   1   1   3   1
   2   3   2   2   1

Fitness function:
   0   0   0   0   0   0   0   0   0   0
The individuals with the best value of the fitness function:
From generation: 0
   1   1   2   1   2
From generation: 1
   1   1   2   1   2
From generation: 2
   1   1   2   1   2
From generation: 3
   1   1   2   1   2
From generation: 4
   1   1   2   1   2
From generation: 5
   1   1   2   1   2
From generation: 6
   1   1   2   1   2
From generation: 7
   1   1   2   1   2
From generation: 8
   1   1   2   1   2
```