**Graph Coloring Problem using a Genetic Algorithm**

**INTRODUCTION**

Informally, a graph is just a collection of points (called vertices) some pairs of which are connected by lines (called edges). Graph coloring is one of the early areas of graph theory. Its origins may be traced back to 1852 when Augustus de Morgan in a letter to his friend William Hamilton asked if it is possible to color the regions of any map with four colors so that neighboring regions get different colors. This is the famous four color problem. The problem was first posed by Francis Guthrie, who observed that when coloring the counties of an administrative map of England only four colors were necessary in order to ensure that neighboring counties were given different colors.

**APPLICATIONS OF GRAPH COLORING**

The applications of graph coloring problem can be found in several areas. Such areas are aircraft scheduling, scheduling committee meetings, bi-processor scheduling, frequency assignment, Time-table design, parent teacher conference scheduling, open shop problem, Sudoku, pattern matching and many more. The second reason why the graph coloring problem is important is that it is computationally hard problem and a number of heuristic approaches have been developed to produce optimal or near optimal solutions in an acceptable amount of time.

**PROBLEM FORMULATION**

An undirected graph is denoted by G=(V,E). Coloring a graph G using k-colors means to partition V into k subsets such that no two adjacent nodes can be in the same subset. The problem of graph coloring is to find the k colors for the graph G such that the number of colors k is as small as possible. The smallest value found for k represents the chromatic number X(G) of graph G. This is an NP-hard problem. If two adjacent vertices v1 and v2 have the same color, then the vertices v1 and v2 are called conflicting vertices, the edge {v1,v2} is called a conflicting edge and the color is called a conflicting color. If no edge is found to be conflicting in a graph G then it is a steady sets and the k-coloring is valid. The goal of graph-coloring problem is to find the minimum value of k in a way that valid k-colors exists for the graph G. The graph coloring problem is very important for two reasons. The first reason it has several practical application areas, where to color an undirected graph with minimum possible color has direct effect on the efficiency of the problem to be solved. Graph coloring problem is actually derived from the well known four color problem; the same has now been diverted to mathematical research. Due to various practical application areas many problems can be now formulated as graph coloring problem.

Mathematical formulation of Graph Coloring Problem:

Suppose, k represents number of colors, say k = {1, 2, 3...} and S(G, k) is the number of possible solutions for coloring the graph G with k colors, Then, χ(G) = minimize (k: S(G, k) > 0) Where, χ(G) is a chromatic number.

**PROPOSED GENETIC ALGORITHM APPROACH**

Genetic Algorithm has a general structure. But the structure can be varied depending upon the problem. Algorithm for implementing the problem of Graph Coloring using Genetic Algorithm :

1. Generate initial random population of chromosomes based on number of vertices and number of colors.

2. Calculate fitness value of each chromosome in initial population using fitness function (two methods).

3. Run Genetic Algorithm for specified number of generations.

4. Apply a newly proposed hybrid Selection and Reproduction on current population.

5. Apply 4 –point or Single- point Cross-over Operator on selected parents.

6. Apply bit-inversion or color change Mutation Operator on randomly selected parents chromosome.

7. if the fitness value become optimal and number of color less than the previous number :Go back to step 1 or go back to step 2, continue the process the number of generations are over.

**Chromosome Encoding**

Encoding of chromosomes is the first question to be addressed while starting to solve a problem with GA. Encoding depends on the problem definition. Figure 2 shows the colored graph with 8 vertices and 3 colors. As shown in the figure the main goal behind graph coloring problem is to find the optimal solution in which no two adjacent vertices will have the same color. Figure also shows the chromosome encoding of the graph where, 1 represents red color, 2 represents green color and 3 represents blue color.



Figure 2 Chromosome encoding of a graph

As specified in [1] the adjacency matrix which specifies adjacencies between vertices is formed of dimensions vertices \* vertices, where 0 represents no edge between two vertices and 1 represents an edge between two vertices.

**Generate population**

Here we used three stratige to generate the initial population :

1- we randomly generate all chromosomes with chromtic [from 0 to the number of color]

#Generate random individuals for population

**def** individual**(**number\_of\_genes**,** number\_of\_color**):**

individual **=** **[]**

**for** i **in** **range(**number\_of\_genes**):**

individual**.**append**(**random**.**randint**(**0**,** number\_of\_color**-**1**))**

**return** individual

#Generate random population

**def** population\_1**(**number\_of\_individuals**,**number\_of\_genes**,** number\_of\_color**):**

**return** **[**individual**(**number\_of\_genes**,** number\_of\_color**)** **for** x **in** **range(**number\_of\_individuals**)]**

2- we randomly generate the first half of population with chromtic [from 0 to the half number of color] and the second half of population with chromtic [from the half number of color to the end] :

**def** population\_2**(**number\_of\_individuals**,**number\_of\_genes**,** number\_of\_color**):**

n **=** number\_of\_color**-**1

num **=** 1

pop **=** **[]**

**while** **(**num **<=** number\_of\_individuals**):**

individual **=** **[]**

**if** num **<=** number\_of\_individuals**//**2**:**

**for** i **in** **range(**number\_of\_genes**):**

individual**.**append**(**random**.**randint**(**0**,** n**//**2**))**

pop**.**append**(**individual**)**

**else:**

**for** i **in** **range(**number\_of\_genes**):**

individual**.**append**(**random**.**randint**(**n**//**2**,** n**))**

pop**.**append**(**individual**)**

num **+=** 1

**return** pop

3- we randomly generate quarter of population with chromtic [from 0 to the half number of color] and the another quarter of population with chromtic [from the half number of color to the end] , and randomly the second half of population:

**def** population\_3**(**number\_of\_individuals**,**number\_of\_genes**,** number\_of\_color**):**

n **=** number\_of\_color**-**1

num **=** 1

pop **=** **[]**

**while** **(**num **<=** number\_of\_individuals**):**

individual **=** **[]**

**if** num **<=** number\_of\_individuals**//**4**:**

**for** i **in** **range(**number\_of\_genes**):**

individual**.**append**(**random**.**randint**(**0**,** n**//**2**))**

pop**.**append**(**individual**)**

**elif** number\_of\_individuals**//**4 **<**num **and** num **<**number\_of\_individuals**//**2**:**

**for** i **in** **range(**number\_of\_genes**):**

individual**.**append**(**random**.**randint**((**n**//**2**)-**1**,** n**))**

pop**.**append**(**individual**)**

**else:**

**for** i **in** **range(**number\_of\_genes**):**

individual**.**append**(**random**.**randint**(**0**,** n**))**

pop**.**append**(**individual**)**

num **+=** 1

**return** pop

**Fitness Function**

In this problem the aim is to color all the vertices of the graph with minimum number of colors. So, the fitness function is to minimize the number of colors.

Here we used two stratiges :

1- The fitness function is designed which calculates number of faults.

Here, the fitness function is fitness function = to minimize #faults

Now, as explained earlier adjacency matrix can be used to find if two adjacent vertices have the same color. If two vertices have a same color then the fault will be increased by 1.

**def** fitness\_calculation\_1**(**individual**,** adjacency\_matrix**):**

fitness\_fault **=** 0

**for** i **in** **range(len(**individual**)-**1**):**

**for** j **in** **range(len(**individual**)):**

**if** individual**[**i**]** **==** individual**[**j**]** **and** adjacency\_matrix**[**j**][**i**]** **==**1**:**

fitness\_fault **+=** 1

fitness\_value **=** fitness\_fault

**return** fitness\_value

2- The fitness function is designed which calculates number of faults and number of colors in chromosome

Here, the fitness function is fitness function = to minimize #faults + colors

**def** fitness\_calculation\_2**(**individual**,** adjacency\_matrix**):**

my\_dict **=** **{**i**:**individual**.**count**(**i**)** **for** i **in** individual**}**

fitness\_color **=** **len(**my\_dict**)**

fitness\_fault **=** 0

**for** i **in** **range(len(**individual**)-**1**):**

**for** j **in** **range(len(**individual**)):**

**if** individual**[**i**]** **==** individual**[**j**]** **and** adjacency\_matrix**[**j**][**i**]==**1**:**

fitness\_fault **+=** 1

fitness\_value **=** 10 **\*** fitness\_fault **+** fitness\_color

**return** fitness\_value

**Selection and Reproduction**

The population size of the initial population and all the successive populations is same. Selection and reproduction are the operators of GA that are applied on the old population for cross-over to generate new population.

The new population will be generated by using following hybrid selection method which includes Elitism and random selection:

1. Elitism: It is the method of selecting the first few best chromosomes for the new population i.e. it first duplicates the best 1/4th chromosomes from the current population.

The fitness values of all chromosomes in the current population are calculated and saved in the fitness list.

Then the fitness list is sorted in ascending order of fitness values. As it saves best 1/4th chromosomes of the population size the performance of GA increases remarkably.

2. The rest of the population is selected at random in order to maintain randomness and a chance to produce a very fit individual from two unfit parents. Thus the new population ready for cross-over is obtained.

**def** selection**(**sorted\_population**):**

num **=** **len(**sorted\_population**)**

n **=** **round(**num **/** 4**)**

k **=** num **-** n

# Elitism Selection

elite\_genertion **=** sorted\_population**[**0**:** n**]**

# Random Selection

random\_genertion **=** random**.**choices**(**sorted\_population**,** k**=**k**)**

selected\_genertion **=** elite\_genertion **+** random\_genertion

**return** selected\_genertion

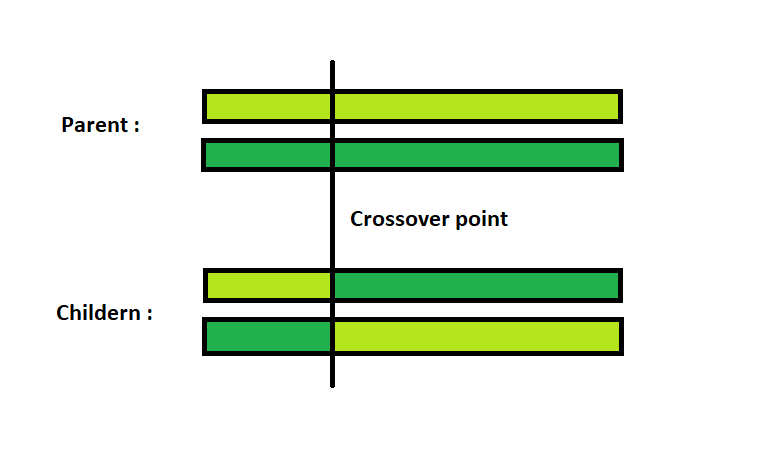
**Crossover**

The crossover operator used in this work randomly picks points and it interchanges the values before and

after that point between two chromosomes to create new offsprings.

In this work 1-point and 4-point crossover has been applied.

**Single point crossover (SPC):**



**def** crossover\_2**(**parent\_list**,** number\_of\_genes**):**

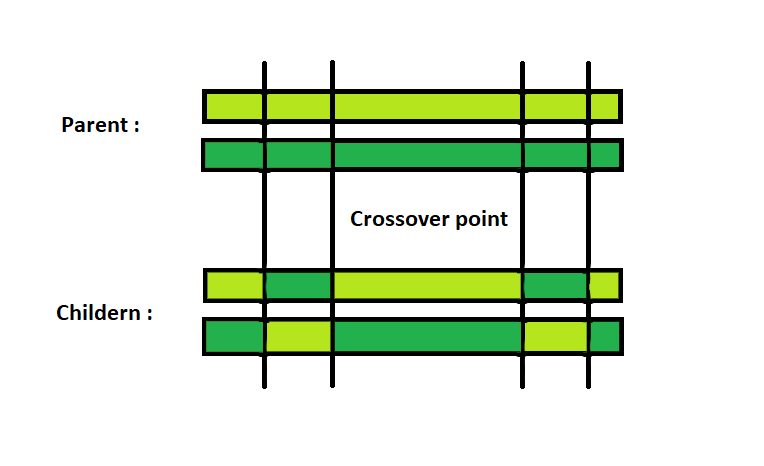
**for** x **in** **range(**0**,** **len(**parent\_list**),** 2**):**

p **=** random**.**randint**(**2**,** number\_of\_genes**)**

parent\_list**[**x**][**0**:**p**],**parent\_list**[**x**+**1**][**0**:**p**]=**parent\_list**[**x**+**1**][**0**:**p**],**parent\_list**[**x**][**0**:**p**]**

**return** parent\_list

**4-point crossover(FPC):**



**def** crossover\_point**(**number\_of\_genes**):**

p1 **=** random**.**randint**(**0**,round(**number\_of\_genes**/**4**))**

p2 **=** random**.**randint**(round(**number\_of\_genes**/**4**),round(**number\_of\_genes**/**2**))**

p3 **=** random**.**randint**(round(**number\_of\_genes**/**2**),round(**number\_of\_genes**\***3**/**4**))**

p4 **=** random**.**randint**(round(**number\_of\_genes**\***3**/**4**),**number\_of\_genes**-**1**)**

point\_list**=** **[**p1**,**p2**,**p3**,**p4**]**

**return** point\_list

**def** crossover\_1**(**parent\_list**):**

**for** x **in** **range(**0**,len(**parent\_list**),**2**):**

#Generate random four point for every parent to make offspring between them

point\_list **=** crossover\_point**(len(**parent\_list**[**0**]))**

parent\_list**[**x**][**point\_list**[**0**]:**point\_list**[**1**]],**parent\_list**[**x**+**1**][**point\_list**[**0**]:**point\_list**[**1**]]** **=** parent\_list**[**x**+**1**][**point\_list**[**0**]:**point\_list**[**1**]],** parent\_list**[**x**][**point\_list**[**0**]:**point\_list**[**1**]]**

parent\_list**[**x**][**point\_list**[**2**]:**point\_list**[**3**]],**parent\_list**[**x**+**1**][**point\_list**[**2**]:**point\_list**[**3**]]** **=** parent\_list**[**x**+**1**][**point\_list**[**2**]:**point\_list**[**3**]],** parent\_list**[**x**][**point\_list**[**2**]:**point\_list**[**3**]]**

**return** parent\_list

**Mutation**

It occurs to some random chromosomes in population , the number of them equal to mutation rate \* number of individuals in the population.

**def** mutation**(**mutation\_method**,** adjacency\_matrix**,** number\_of\_color**,** child\_list**,** mutation\_rate**):**

num **=** **round(len(**child\_list**)\***mutation\_rate**)**

idx\_list **=** **range(len(**child\_list**))**

idx\_individuals **=** random**.**sample**(**idx\_list**,** num**)**

**for** x **in** idx\_individuals**:**

**if** mutation\_method **==** '1'**:**

mutation\_individual\_1**(**child\_list**[**x**])**

**else:**

mutation\_individual\_2**(**child\_list**[**x**],**adjacency\_matrix**,**number\_of\_color**)**

**return** child\_list

**First method:**

During mutation it changes two vertex colors randomly in the same chromosome

**def** mutation\_individual\_1**(**individual**):**

idx **=** **range(len(**individual**))**

i1**,** i2 **=** random**.**sample**(**idx**,** 2**)**

individual**[**i1**],** individual**[**i2**]** **=** individual**[**i2**],** individual**[**i1**]**

**return** individual

**Second methond:**

Search for every fault in the chromosome and change the color

**def** mutation\_individual\_2**(**individual**,**adjacency\_matrix**,**number\_of\_color**):**

**for** i **in** **range(len(**individual**)-**1**):**

**for** j **in** **range(len(**individual**)):**

**if** individual**[**i**]** **==** individual**[**j**]** **and** adjacency\_matrix**[**j**][**i**]** **==**1**:**

**while** **True:**

new\_color **=** random**.**randint**(**0**,** number\_of\_color**-**1**)**

**if** individual**[**i**]** **!=** new\_color**:**

individual**[**i**]** **=** new\_color

**break**

**return** individual

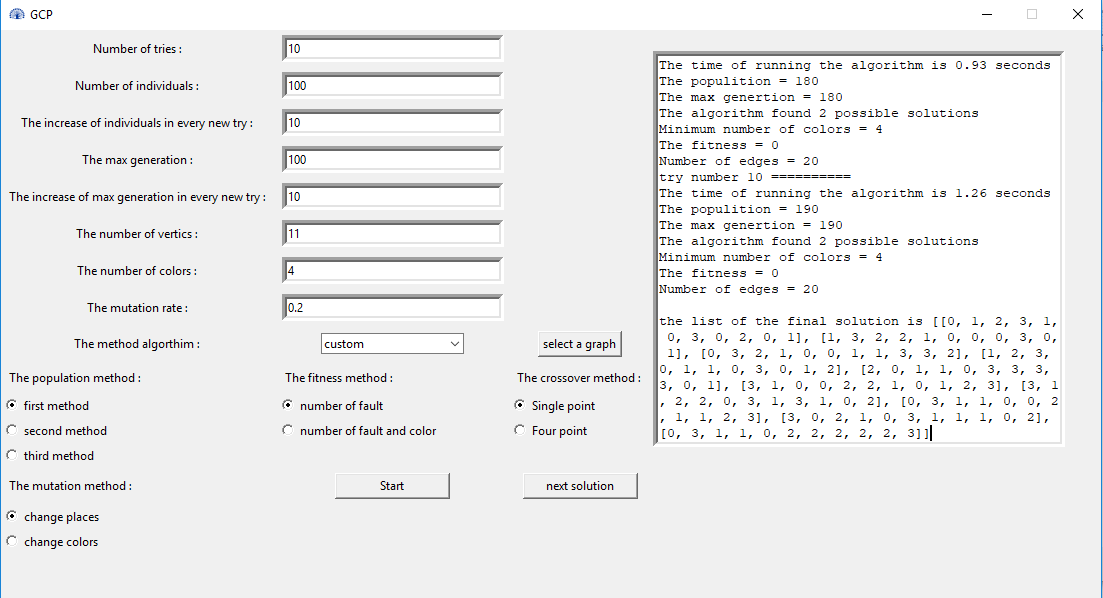
**RESULTS**

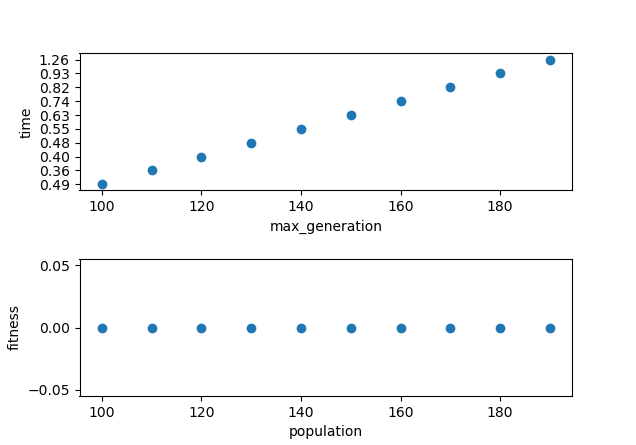
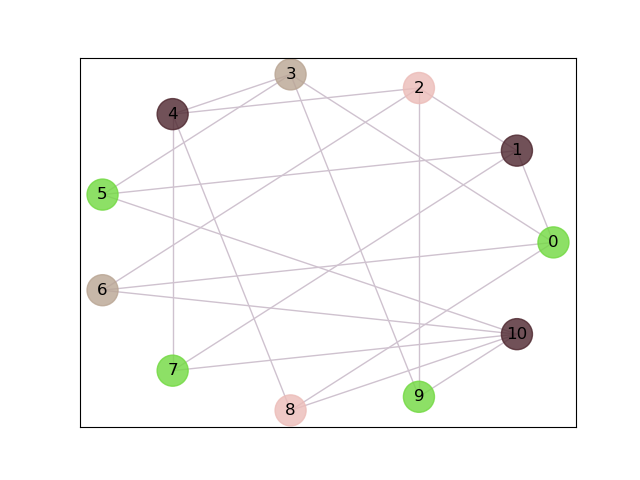
We chose two files to try the methods myciel3.col and myciel5.col

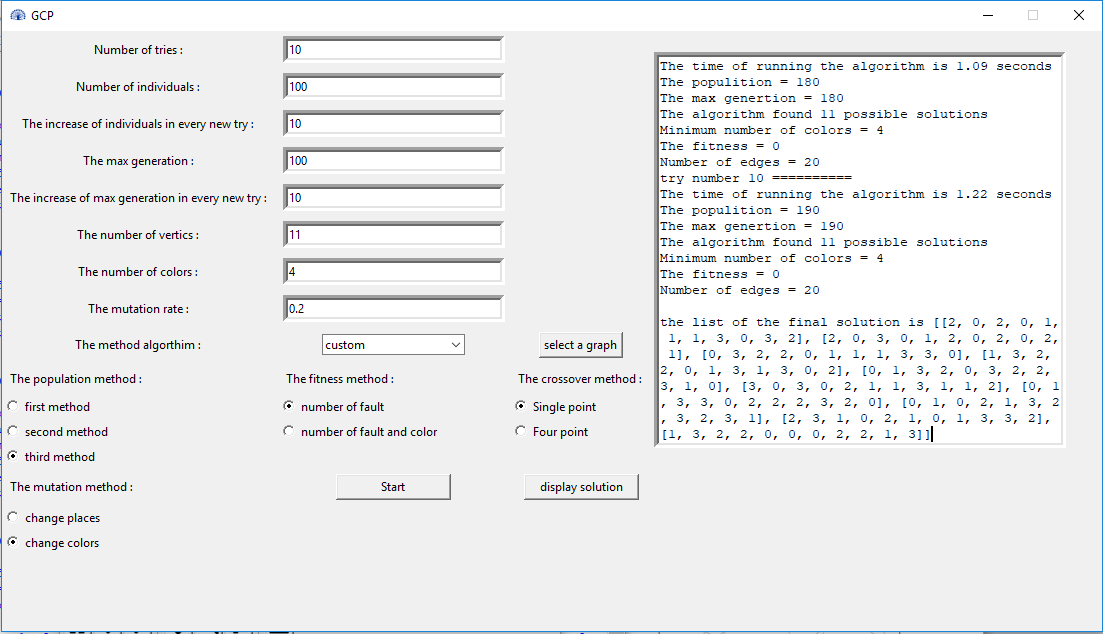
We conducted ten experiments with different methods in every experiment we incraesed the number of individuals in the poputalion and the number of max generation

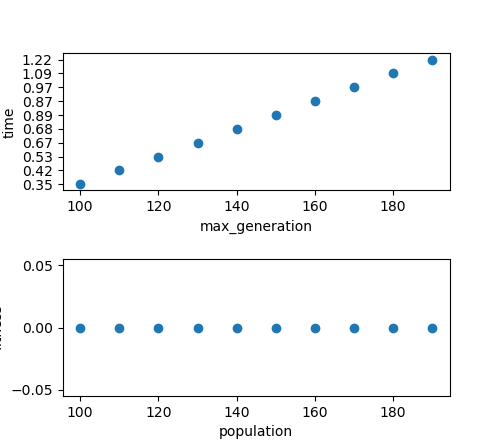
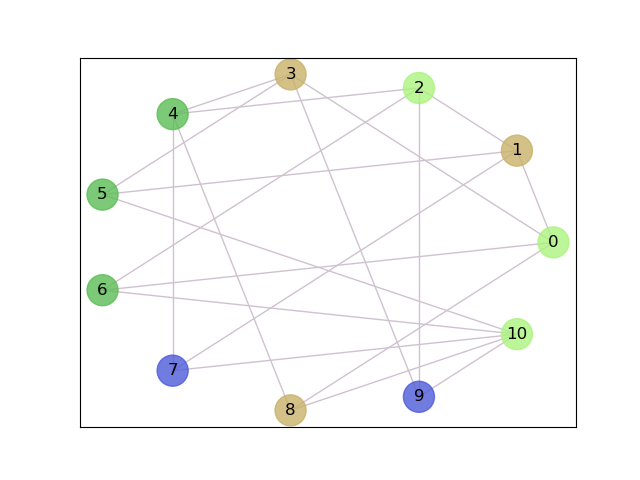
Some of methods for myciel3.col (V=11,E=20,K=4)

1- first method of population, fault method of fitness, SPC, exchange plsces of two colors method of mutation:





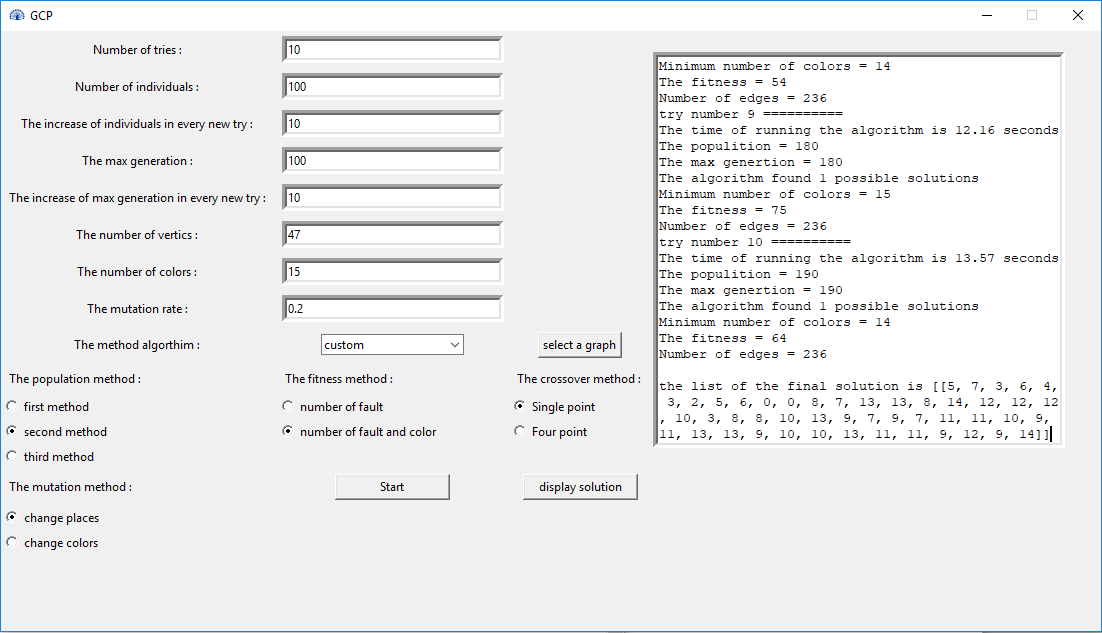
2- third method of population, fault method of fitness, SPC, change colors method of mutation:

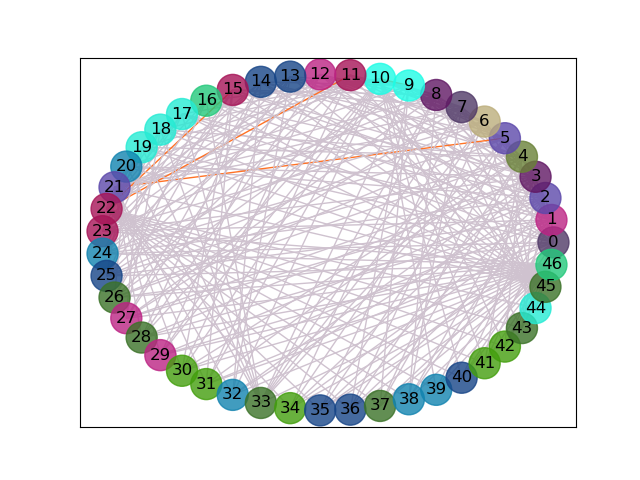
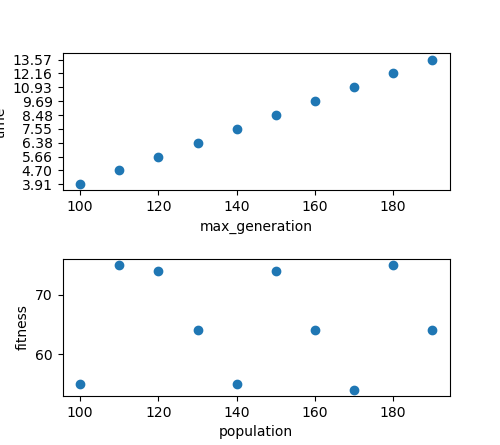


The algorthim can find the optimal solution with any methods we use, the difference is the running time.

Some of methods for myciel5.col (V=47,E=236,K=6)

**IF we start with number of color = 15:**

1- second method of population, fault and color method of fitness, SPC, exchange places of two colors method of mutation 



It’s a very bad result, the algorthim can’t find any optimal solution even with 15 color and the best one is Minimum number of colors = 14

The fitness = 54 🡪 4 fault with 14 color

2- first method of population, fault method of fitness, FPC, change colors method of mutation:

we get a very good result ,the algorthim found a solution but it is not the optimal solution

max\_generation,populition,number\_of\_solutions,number\_of\_color,fitness,time

100,100,14,11,0,6.15

110,110,15,12,0,6.46

**120,120,5,8,0,10.09**

130,130,6,9,0,8.60

140,140,17,11,0,10.27

150,150,13,9,0,15.45

160,160,17,9,0,11.24

170,170,26,9,0,16.14

180,180,14,11,0,14.11

**190,190,3,8,0,18.49**

3- first method of population, fault method of fitness, SPC, change colors method of mutation:

**100,100,50,8,0,6.42**

110,110,16,11,0,6.40

**120,120,7,8,0,9.07**

130,130,8,9,0,11.57

140,140,13,10,0,10.60

150,150,20,10,0,13.08

**160,160,4,8,0,12.45**

170,170,13,10,0,12.87

180,180,11,9,0,17.59

190,190,21,11,0,19.45

**If we start with number of color = 6**

3- first method of population, fault method of fitness, FPC, change colors method of mutation

100,100,1,6,2,4.36

**110,110,1,6,0,5.11**

120,120,1,6,1,6.23

**130,130,12,6,0,7.20**

140,140,1,6,3,8.36

150,150,1,6,1,9.33

160,160,1,6,1,10.80

170,170,1,6,1,12.16

**180,180,1,6,0,13.66**

**190,190,1,6,0,15.15**

4- first method of population, fault method of fitness, SPC, change colors method of mutation

100,100,1,6,1,4.37

110,110,1,6,1,5.67

**120,120,2,6,0,6.17**

130,130,1,6,2,7.10

140,140,1,6,2,8.38

150,150,1,6,3,9.85

160,160,1,6,1,10.58

170,170,1,6,1,12.24

**180,180,2,6,0,13.79**

**190,190,1,6,0,15.46**

After these results, I decied to try another big graph anna.col (V=138,E=986,K=11)

IF START WITH 13 COLORS

1- first method of population, fault method of fitness, FPC, change colors method of mutation

100,100,5,12,0,39.17

110,110,13,12,0,47.18

120,120,6,12,0,56.56

130,130,18,12,0,66.37

140,140,6,12,0,80.18

150,150,9,12,0,88.48

160,160,21,12,0,100.88

170,170,19,12,0,113.02

180,180,9,12,0,127.40

190,190,29,12,0,148.58

2- first method of population, fault method of fitness, SPC, change colors method of mutation

100,100,36,13,0,39.28

110,110,11,13,0,47.39

120,120,15,13,0,56.70

130,130,19,13,0,66.76

140,140,14,13,0,79.98

150,150,23,13,0,88.15

160,160,16,13,0,101.12

170,170,11,13,0,113.06

180,180,20,13,0,128.66

190,190,18,13,0,141.64

**IF START WITH 11 COLORS**

1- first method of population, fault method of fitness, FPC, change colors method of mutation

max\_generation,populition,number\_of\_solutions,number\_of\_color,fitness,time

10,10,1,11,16,0.33

20,20,1,11,0,1.42

30,30,1,11,0,3.46

40,40,1,11,0,6.82

**50,50,3,11,0,9.16**

60,60,1,11,0,13.53

70,70,7,11,0,18.77

80,80,2,11,0,24.44

90,90,4,11,0,30.42

100,100,3,11,0,38.98

110,110,8,11,0,46.45

120,120,2,11,0,56.10

130,130,7,11,0,65.79

140,140,3,11,0,77.45

150,150,8,11,0,88.18

160,160,2,11,0,99.89

170,170,7,11,0,114.88

180,180,6,11,0,128.77

190,190,8,11,0,145.33

2- first method of population, fault method of fitness, SPC, change colors method of mutation

10,10,1,11,8,0.32

20,20,1,11,0,1.40

30,30,2,11,0,3.21

40,40,1,11,0,5.78

**50,50,5,11,0,8.95**

60,60,1,11,0,13.47

70,70,1,11,0,18.63

80,80,2,11,0,24.56

90,90,3,11,0,31.09

100,100,3,11,0,38.64

110,110,2,11,0,46.93

120,120,3,11,0,56.80

130,130,5,11,0,65.67

140,140,10,11,0,77.36

150,150,8,11,0,89.76

160,160,10,11,0,102.32

170,170,8,11,0,116.97

180,180,10,11,0,127.42

190,190,21,11,0,141.16

So we can get multi solution with 50 individual and 50 generation, even we get result with 20 and 20.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name file | V | E | K | Num\_ind | Num\_gen | Our result | Time (s) |
| myciel3.col | 11 | 20 | 4 | 10 | 10 | 4 | 0.001 |
| myciel4.col | 23 | 71 | 5 | 10 | 10 | 5 | 0.01 |
| queen5\_5.col | 25 | 160 | 5 | 60 | 60 | 5 | 0.7 |
| **queen6\_6.col** | 36 | 290 | 7 | 90 | 90 | 8 | 2.22 |
| myciel5.col | 47 | 236 | 6 | 100 | 100 | 6 | 4.44 |
| **queen7\_7.col** | 49 | 476 | 7 | 110 | 110 | 10 | 5.97 |
| **queen8\_8.col** | 64 | 728 | 9 | 160 | 160 | 12 | 19.29 |
| huck.col | 74 | 301 | 11 | 10 | 10 | 11 | 0.11 |
| jean.col | 80 | 254 | 10 | 20 | 20 | 10 | 0.48 |
| david.col | 87 | 406 | 11 | 20 | 20 | 11 | 0.6 |
| games120.col | 120 | 638 | 9 | 30 | 30 | 9 | 2.54 |
| miles250.col | 128 | 387 | 8 | 50 | 50 | 8 | 7.83 |
| miles1000.col | 128 | 3216 | 42 | 80 | 80 | 47 | 19.92 |
| anna.col | 138 | 493 | 11 | 20 | 20 | 11 | 1.87 |
| homer.col | 561 | 1629 | 13 | 50 | 50 | 13 | 157.43 |

**References**

[1] Musa M. Hindi and Roman V. Yampolskiy, Genetic Algorithm Applied to the Graph Coloring Problem, Proceedings of the Twentythird Midwest Artificial Intelligence and Cognitive Science Conference , April 2012,University of Cincinnati Cincinnati, Ohio.

[2] Sandeep R. Vasant1, Bipin Thanki2, Dr Avani R. Vasant3, Dr N.N.Jani4, Optimization of Graph Coloring Problem using Hybrid Selection: A Genetic Algorithm Approach, January 2014, researchgate.