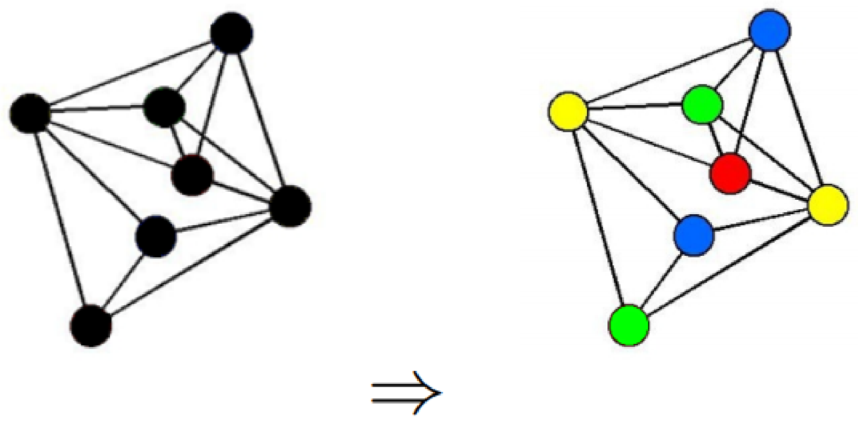
Homework 3 - Finding the minimum number of colors used in a graph coloring problem using a genetic algorithm in comparison with an heuristic algorithm

Elisa Giurgea, Rameder Carol - A2

December 2019

1 Introduction

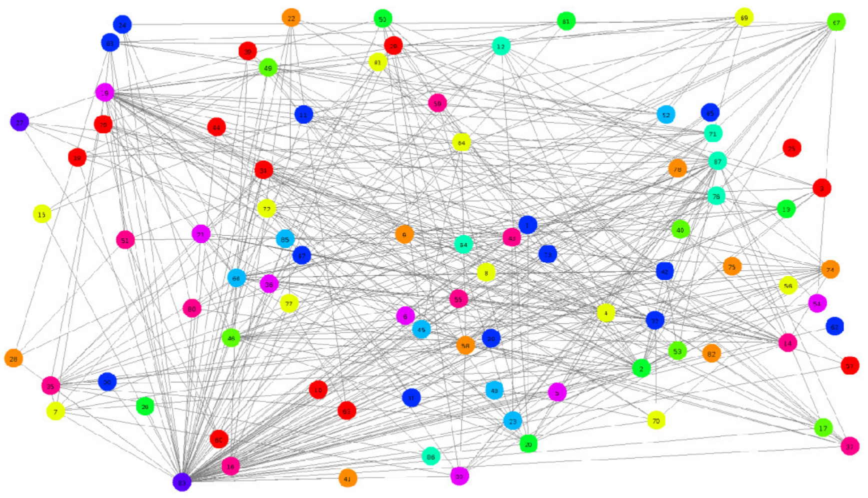
The graph coloring problem consists of assigning colors to certain elements of a graph taking into consideration specific requirements. One widely known graph coloring problem is vertex coloring: given a number of colors, find a way of coloring the nodes of the graph such that no two adjacent nodes have the same assigned color.



For this project, the studied problem is finding the minimum number of colors for coloring a graph that respects the previously stated constraint: no two adjacent nodes with the same assingned color.

Finding the minimum value using implemented algorithms can be quite a challenging task, as the problem itself is NP-Complete, meaning that there is no discovered way for its algorithm to be solved in a polynomial time, but there are certain algorithm which can find the right solution in some cases: heuristic and genetic algorithms. In this case study, we will underline the main differences between how a genetic algorithm works for solving this problem and how a classic nondeterministic algorithm works for finding the minium number of colors.

The graphs that were used for testing the algorithms and analyzing the results can be found in the DIMACS dataset for graph clique numbers.



2 Methods

2.1 The Genethic Algorithm

First of all, what is a genetic algorithm based on? A genetic algorithm is a heuristic search that is inspired by the theory of natural evolution. On a generated population of individuals, ”chromosomes”, a selection algorithm is applied so that the fittest individuals are chosen for reproduction in order to produce offspring of the next generation. In a genetic algorithm, the set of genes of an individual is represented using a binary values to encode the genes in a chromosome.

2.1.1 Pseudocode

```
begin
t := 0
generate P(t)
evaluate P(t)

while (not stoping condition) do begin
          t := t + 1
          select P(t) from P(t-1)
          recombine P(t) //crossover between chosen parents and mutation their child
          evaluate P(t)
end
```

2.1.2 The Roullete Wheel Selection Method

The Roulette Wheel selection is a method which says that all the individuals in the population are placed on the roulette wheel according to their fitness value with each individual being assigned a segment of the roulette wheel proportional to its fitness score. Then, the virtual roulette wheel is spinned and the individual corresponding to the segment on which roulette wheel stops is elected.

This particluar method of implementation clearly favors the high fitness individuals and it can even miss the best individuals of a population, so there is no certainty that good individuals will be added to the next generation.

2.1.3 Pseudocode

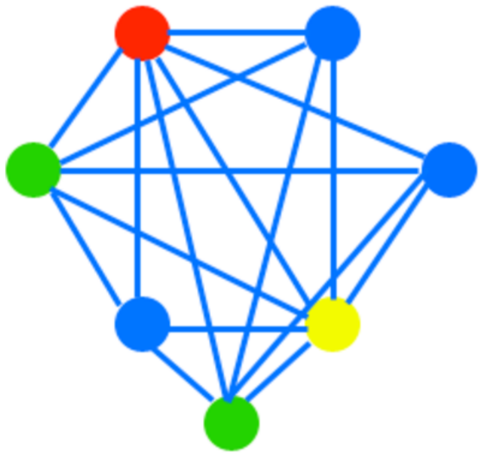
```
EVALUATE P
    for i:=0 to POP_SIZE
        eval[i]=f(P(i))
TOTAL FITNESS
    for i:=0 to POP_SIZE
        T+=eval[i]
INDIVIDUAL SELECTION
    for i:=0 to POP_SIZE
        p[i]=eval[i]/T
PROBABILITY FOR CUMULATED SELECTION
    q[0] = 0
    for i:=0 to POP_SIZE
        q[i+1]=q[i]+p[i]
SELECTION
    for i:=0 to POP_SIZE
        generate random r in (0,1]
        select for survival j
            for which q[j]<r<=q[j+1]
```

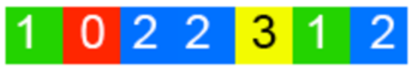
2.1.4 Population

A population is a set of individuals, each individual being a possible solution to be analyzed for the given problem. An individual, as said before, is represented as a set of genes.

2.1.5 Representation

Each chromosome is represented as an array, where each value from a position in the array is a gene which has the value of the color with which the node with the index equal to the one that the position has.





2.1.6 Fitness

The fitness function determines how fit an individual is and based on that it gives the individual a score, which will later be used to determine the probability that a certain individual is selected.

For this algorithm, the fitness will be calculated like this: if k, which represents the number of nodes in the graph which aren't colored correctly in 0, is 0, meaning that the solution is correct, the fitness value is bigger, thus favoring correct solutions in favor to incorrect ones, but even for the incorrect ones (as it is an NP-complete problem which doesn't guarantee that a correct result will be found in a thread of execution), the solutions which have a smaller number of bad nodes are favored.

$$fitness = \begin{cases} \frac{1}{maximum\_color\_used * k} & k >= 1 \\ \frac{1}{maximum\_color\_used * 1} & k = 0 \end{cases}$$

By using this fitness function, the fittest individuals will be the most likely to survive in the Roullete Wheel selection.

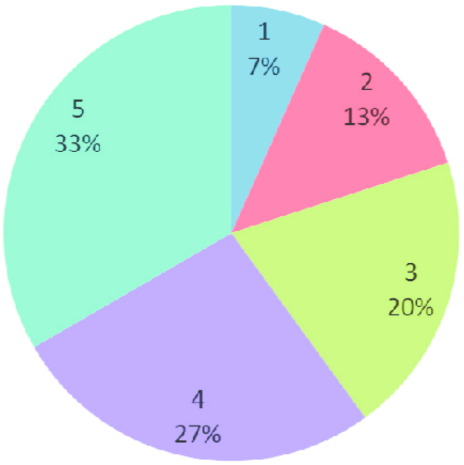
2.1.7 Selection

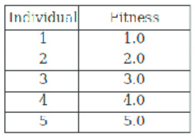
The idea of the selection step is to select a pair of individuals based on their fitness. For the Roulette Wheel selection method, a field of probabilities will be created: the individuals with a higher fitness value (the ones which are more likely to be closer to a correct solution) will also have a higher probability to be selected to be in the next generation.

Each chomozome of the population will be assigned a probability using the formula:

$$\frac{fitness(i)}{\sum_{j=1}^{nodes} fitness(j)}$$

After creating the initial array of probabilities we will generate a cumulated probability for each chromosome, thus selecting the i chromosome if its cumulated probability is bigger than the one for the $i + 1$ chromosome.





Individual	Fitness
1	1.0
2	2.0
3	3.0
4	4.0
5	5.0

2.1.8 Crossover

For each pair of parents to be selected for reproduction, a crossover point is chosen randomly within the genes. The genes from a parent up that position merge with the genes from the other parent from that position on to create a new individual to be added to the population.

2.1.9 Mutation

A mutation is a change in the genes of an individual and it occurs to maintain diversity within the population and prevent premature convergence.

In this case, the mutation is done less destructively in order to have better results. Before doing the actual the mutation. we visit all the neighbours of the node and memorize the colors that cannot be used because they are already assigned to a direct neighbour. The node is then assigned a random color which wasn't used for any neighbour.

This way the given solution will be highly improved.

2.1.10 Results

The genethic algorithm was tested 30 times for each given data graph in order to provide a more accurate understanding of the results.

Tested graph	Best Value	Worst Value	Time(seconds)	Actual solution
anna.col	16	21	23.919607	11
david.col	14	19	23.481177	11
homer.col	16	21	21.406298	13
huck.col	12	17	22.079593	11
jean.col	11	14	21.560792	10
games120.col	12	16	23.851527	9
miles1000.col	45	52	22.820344	42
miles1500.col	77	83	22.016013	73
miles250.col	12	23	24.055900	8
miles500.col	23	26	22.016013	20
miles750.col	34	37	21.784309	31
queen10_10.col	18	20	25.115941	10
queen11_11.col	20	23	27.553711	11
queen12_12.col	21	25	29.857117	12
queen5_5.col	8	9	21.350732	5
queen6_6.col	10	16	23.851527	6
queen7_7.col	9	12	22.158261	6
queen8_12.col	18	20	24.752088	11
myciel3.col	4	5	20.461944	4
myciel4.col	5	8	20.694113	5
myciel5.col	9	10	21.428677	6

Table 1: Roulette Wheel

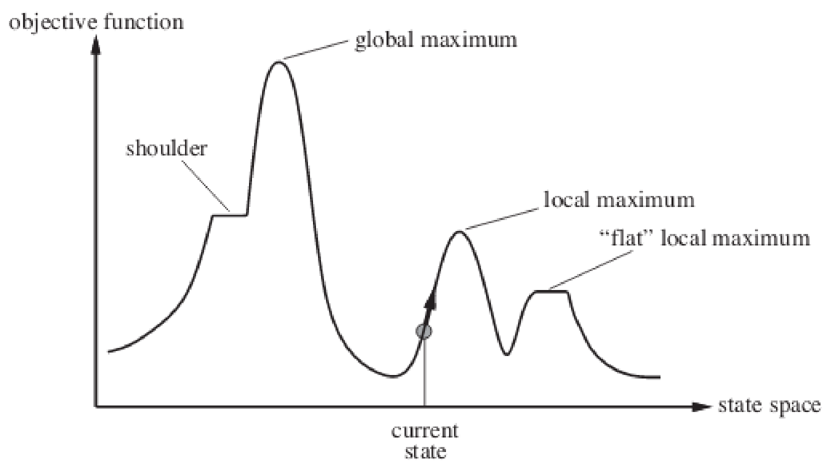
2.2 Hill Climbing Algorithm

Hill Climbing algorithm is a local search algorithm.It initally takes an arbitrary value and starting there it compares its neighbours to see which has a higher value, then it moves in the direction of increasing neighbour values to find the peak of the hill or best solution to the problem, which is the global optima.

It ends when it reaches a point were no surrounding values are better than the current best. It has a classic Greedy approach: it constantly searches for the solution which can optimize the result.

This algorithm finds optimal solutions for convex problems – for other problems it will find only local optima, which are not necessarily the best possible solutions. A local optima is surrounded by values that are worse than it, which creates an attraction basin for that point. Thus, the local optima in cause is the best possible value in that certain "neighbourhood".

Hill Climbing First Improvement is based on the idea that the Hill Climbing algorithm searches in a neighbourhood the first solution better than the current one. Meanwhile, Hill Climbing Best Improvement looks for the first solution better that the overall best one up until that point.



2.2.1 Hill Climbing algorithm description

The Hill Climbing algorithm used to find the optimal solution for the given examples starts with a random solution for the graph, generated as in the following pseudocode. This solution is improved by browsing its neighbourhood until we find the best neighbour, and if this is more optimal than the first solution, the values are stored in the solution itself and the algorithm repeats and searches again for the best neighbour. At the end, following these steps, we will find the minimal local solution, which will be taken into consideration for the minimal selection over all the 30 iterations that will be proceeded.

This way, the solution found with the heuristic algorithm is considerable close to the real solution of the problem. The better the neighbourhood is searched and the more iterations, the better the chances are to find a closer solution in comparison to the real one.

In our case, the solution is represented by a vector with n elements, one for every node in the given graph. The value stored on position *i* in our vector is the color associated with the *i* node .

The neighbourhood is serched by changing every position in our solution with one unit up and down, so there will be 2*n neighbours . These are checked so that they respect the conditions of the graph coloring problem and evaluated for the number of colors used with two implemented functions: "esteSol()" a boolean function which returns 1, if the neighbour is a solution, or 0 otherwise, and "eval" which counts how many colors are used.

2.2.2 Pseudocode

```
ITERARE HC
begin
    t := 0
    initialize best
        repeat
            local := FALSE
BEGIN HC
            select a candidate solution sol at random
            evaluate vc
            repeat
```

```
                bestn := Improve(Neighborhood(sol))
                if eval(bestn) is better than eval(sol)
                    then sol := bestn
                    else local := TRUE
END HC
            until local
            t := t + 1
            if sol is better than best
                then best := sol
        until t = MAX
end

GENERATE RANDOM SOLUTION
for i in nodes
check_Available_Colors(i)
sol[i]=choose_Random_Available_Color()
```

2.2.3 Results

Tested graph	Best Value	Worst Value	Time(seconds)	Actual solution
anna.col	40	43	0.001 - 0.998	11
david.col	27	32	0.001 - 0.998	11
homer.col	180	195	0.001 - 0.998	13
huck.col	24	28	0.001 - 0.998	11
jean.col	26	36	0.001 - 0.998	10
games120.col	12	16	0.001 - 0.998	9
miles1000.col	45	52	0.001 - 0.998	42
miles1500.col	77	83	0.001 - 0.998	73
miles250.col	12	23	0.001 - 0.998	8
miles500.col	23	26	0.001 - 0.998	20
miles750.col	34	37	0.001 - 0.998	31
queen10_10.col	18	20	0.001 - 0.998	10
queen11_11.col	20	23	0.001 - 0.998	11
queen12_12.col	21	25	0.001 - 0.998	12
queen5_5.col	9	11	0.001 - 0.998	5
queen6_6.col	12	14	0.001 - 0.998	6
queen7_7.col	14	18	0.001 - 0.998	6
queen8_12.col	38	50	0.001 - 0.998	11
myciel3.col	4	7	0.001 - 0.998	4
myciel4.col	7	9	0.001 - 0.998	5
myciel5.col	20	20	0.001 - 0.998	6

Table 2: Hill Climbing

3 Conclusion

As it can be seen from the obtained results, the Roullete Wheel genetic algorithm returns fairly good data, but it fails from time to time, so it raises the question as to what is the reason behind it. Similar to its name, it acts as a roulette wheel, thus favoring for selection the individual with a higher fitness score, as it is proportional to the probability that it is assigned to each individual. The bigger the fitness score, the larger is the portion of the hypothetical wheel that the individual occupies. This algorithm can also miss the best individuals of a population. In the case of the graph coloring problem, because of the modified mutation and selection method, the result are satisfying.

In this case, the Hill Climbing algorithm provides data that fits one of the requirements: no two adjacent nodes have the same color, but is rarely goes close to the minimum value of colors which is requested. A possible reason for this is the number of iterations and the way the neighbours are searched. It goes close to the answer in some cases, but it can be improved.

When it comes to speed, the genetic algorithm's performance is way below the hill climbing approach, because of the more throughout way of verification.

Two highly different algorithms, the Roullete Wheel and Hill Climbing showcase successfully the diferences between non-deterministic and deterministic solutions in solving an NP-complete problem.

4 Bibliography

https://drops.dagstuhl.de/opus/volltexte/2018/8455/pdf/OASIcs-ICLP-2017-5.pdf
http://rhydlewis.eu/papers/groupPaper.pdf
http://ceur-ws.org/Vol-841/submission$_1$0.*pdf*
https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
https://profs.info.uaic.ro/ pmihaela/GA
https://www.al-roomi.org/component/tags/tag/1st-de-jong-s-function