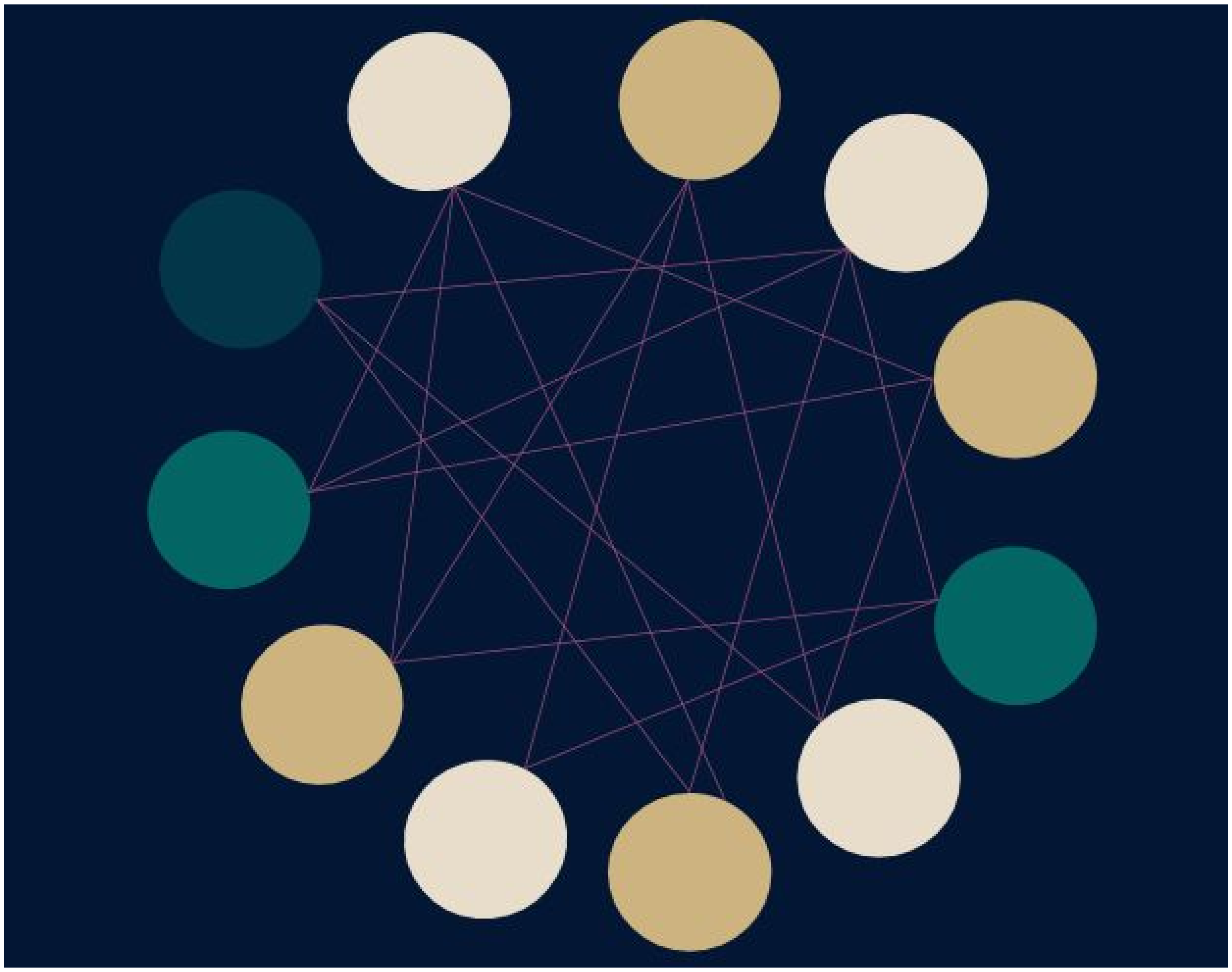
Graph Coloring

Genetic Algorithm, Simulated Annealing, Tabu Search



Juan José Martín Cara

22.12.2017

Heuristic Algorithms for NP-complete Problems

INTRODUCTION

In this report we will discuss the problems we have encounter during the implementation of this practice where we try to solve the Graph coloring problem.

Besides this discussion, we will also compare and contrast the advantages and disadvantages found for every method used to solve the problem and which one has the best performance in our tests.

THE GRAPH COLORING PROBLEM

The Graph coloring is a NP-Complete problem and a special case of the graph labeling problem. To simply describe it we can say that is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color, this process is called vertex coloring.

ALGORITHMS

1. **Genetic Algorithm**
2. **Simulated Annealing**
3. **Tabu Search**

THE GRAPHS

The graphs used in this problem where Graphs based on the Mycielski transformation. These graphs are difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size.

These graph are from Michael Trick (myciel3, myciel4 and myciel5) from the website

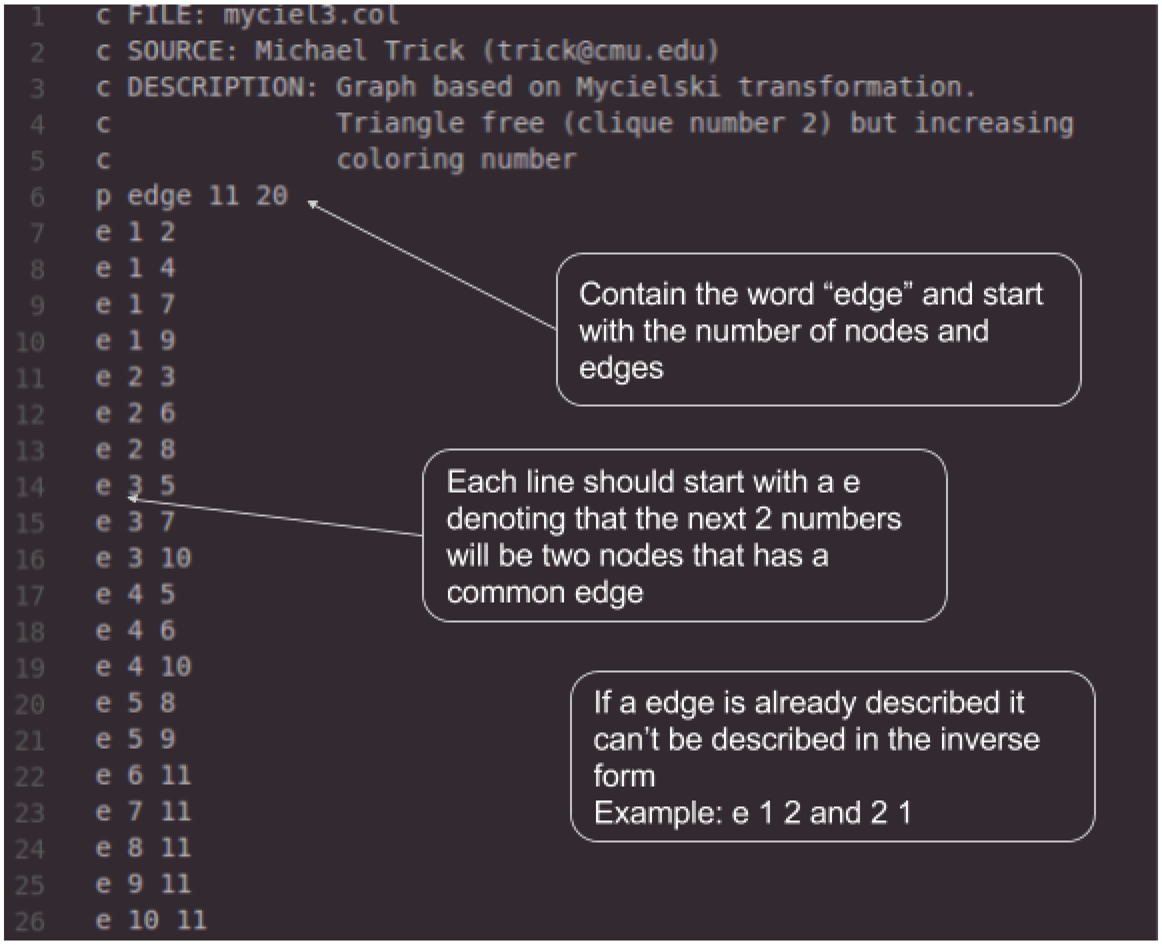
http://mat.gsia.cmu.edu/COLOR/instances.html#XXMYC

IMPLEMENTATION

To implement the Graph coloring problem we made some assumptions that need to be mentioned in this report.

Of course, the fact that the name of the problem contains the word “coloring” doesn’t mean that we need to use colors, this word comes from the original problem where colors were used on a map. For our implementation the system uses unsigned integers that will represents colours. It is not important which color are those numbers, although it is important the number of colours that are required to colour the graph.

There are a few assumptions worth to mention that we did in the process of load a new graph, those are:



```
c FILE: myciel3.col
c SOURCE: Michael Trick (trick@cmu.edu)
c DESCRIPTION: Graph based on Mycielski transformation.
c              Triangle free (clique number 2) but increasing
c              coloring number
p edge 11 20
e 1 2
e 1 4
e 1 7
e 1 9
e 2 3
e 2 6
e 2 8
e 3 5
e 3 7
e 3 10
e 4 5
e 4 6
e 4 10
e 5 8
e 5 9
e 6 11
e 7 11
e 8 11
e 9 11
e 10 11
```

Contain the word “edge” and start with the number of nodes and edges

Each line should start with a e denoting that the next 2 numbers will be two nodes that has a common edge

If a edge is already described it can’t be described in the inverse form
Example: e 1 2 and 2 1

As we will see later, there is a class called Individual that it is used as individuals in the genetic algorithm implementation, but due to the good fitness of that class we used it on other algorithms as a solution or temporary solution.

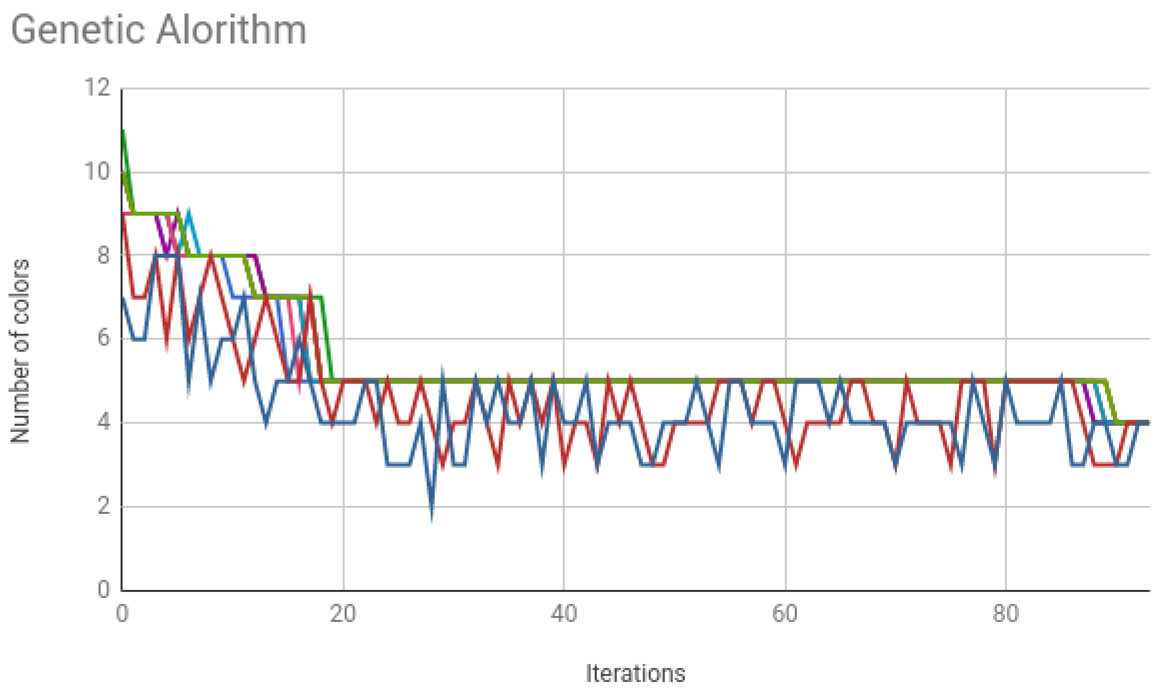
Also, the fitness function used in the three methods is the same but with some variations, basically its assign a number that will be the score or fitness of an individual. If the individual has less collisions, its score will be higher.

Once the individual does not have any collisions in the solution, the fitness will increase with the difference between a solution with all nodes painted with different colors and the current one, meaning that if we used less colors, our fitness value will be higher.

GENETIC ALGORITHM

In the Genetic algorithm the system starts with a population that it is created randomly, then the best individuals are chosen, with this best individuals we create a new population applying some operations.

Here he have a graph about the iteration in the Genetic Algorithm



This correspond to the Myciel3, as we can see all individuals are initialized with a high number of colors and move a lot, but there is a part where all individuals stabilize, that is because it has found a solution that it is really close to the best solution, so it is more difficult to improve at that point than previously.

Once the system find the best solutions (at the end of the graph) it will stabilize even more because all individuals will converge to that individual.

The program will never stop because it does not know if it has found the best solution or just another normal solution, so we need to provide a stopping criteria.

The stop condition in this case is that the program will finish its execution if eight out of the ten individuals find the solution. We use eight and not all of them because the mutation process will always impact on some of the individuals.

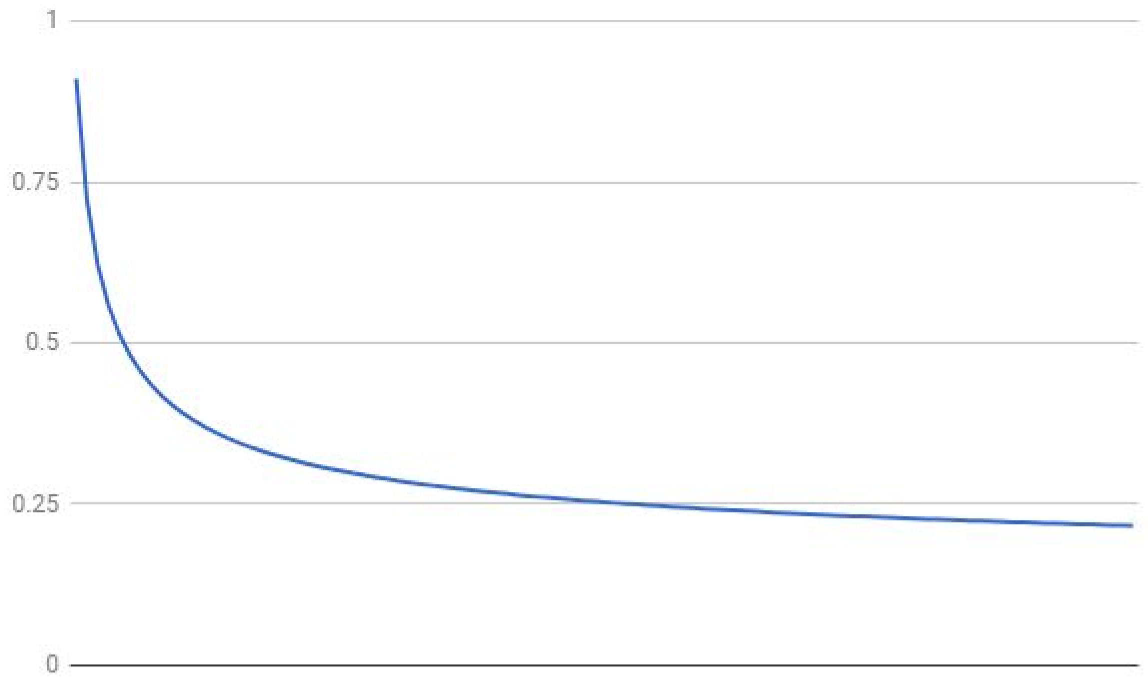
The mutation operation is one of the three operations used in this program to find new populations:

- Best Individuals: The 40% of the new population is found with this operation, to compute it, the system just calculate the fitness of all the individuals from the previous population and choose the 40% that has the best fitness
- Crossover: This operation computes the 40% also of the best individuals, to do that just take two individuals from the “best individuals” set and merge their genes to create a new individuals
- Mutation: The 20% left of the individuals is computed with this method, to do that the mutation function chose a random individual and randomly change some of its genes

SIMULATED ANNEALING

The simulated annealing basics are simple, we start with an initial random configuration, then we select a neighbor from our neighborhood. In order to do this, we have to evaluate the fitness of all the individuals in the neighborhood and decide if we select a specific candidate. Once the quality of the solution is enough we can stop or if we reach a stopping criteria.

However this method can get stuck in a local minimum, that’s why we use a variable called temperatura that will control if we accept or not a worst solution.



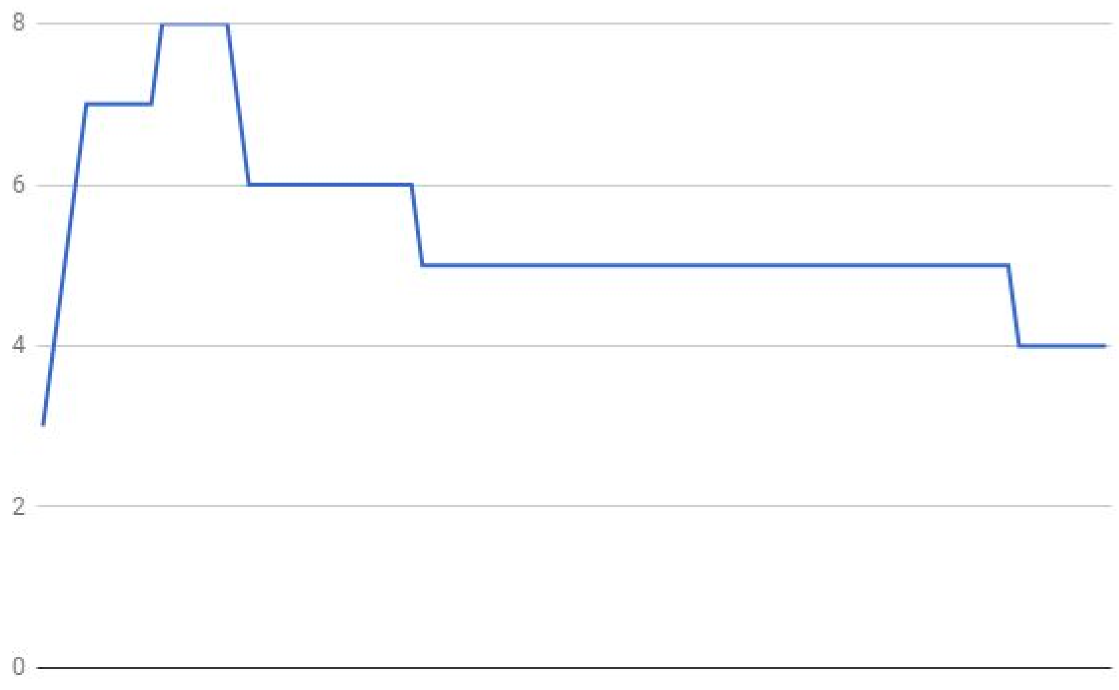
This is the graph for the temperature used in our system, as we can see it follows this ecuation:

temperature = temperature0 * (1/(log(k+2)));

Where:

- temperature: is the current temperature
- Temperature0: is the initial temperature
- K: is a value calculated each iteration

As we can see the temperature starts with a high value, meaning that at the beginning the system has a big chance of choosing a worst solution of the neighborhood, but by the end of the running, the temperature will be so low that it is almost impossible that the system move to a worst solution.



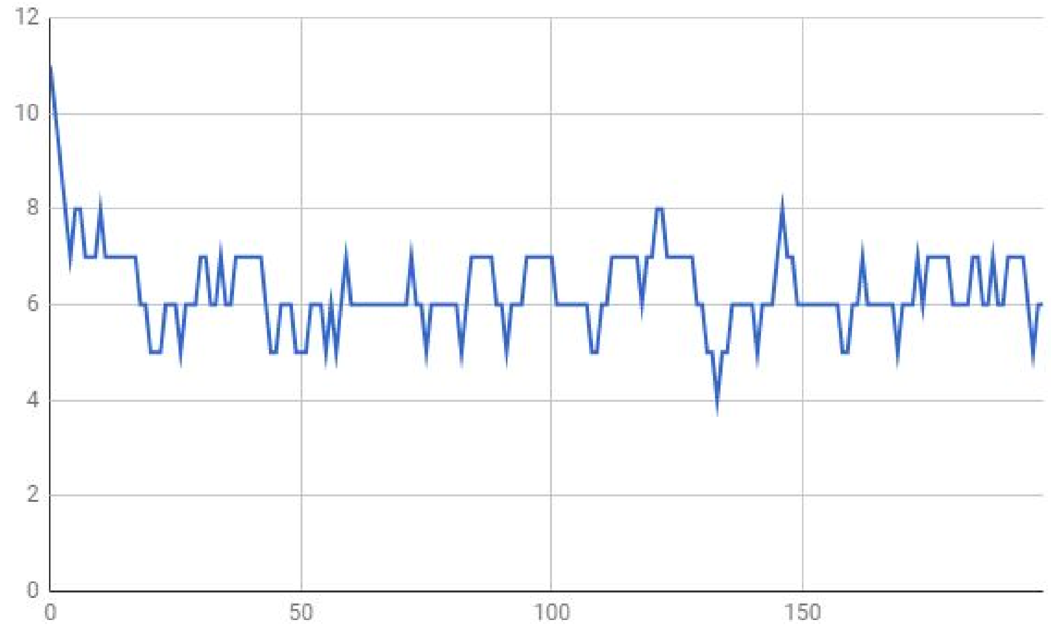
This graph shows the numbers of colours that has the current solution in each iteration, as we can see, at the beginning the system is not stable and that is why we move to worse solutions but by the end the system is stable and it is difficult to go to a worst solution. That is why we can see a straight line in the middle of the graph by that point the system will only choose a better solution.

TABU SEARCH

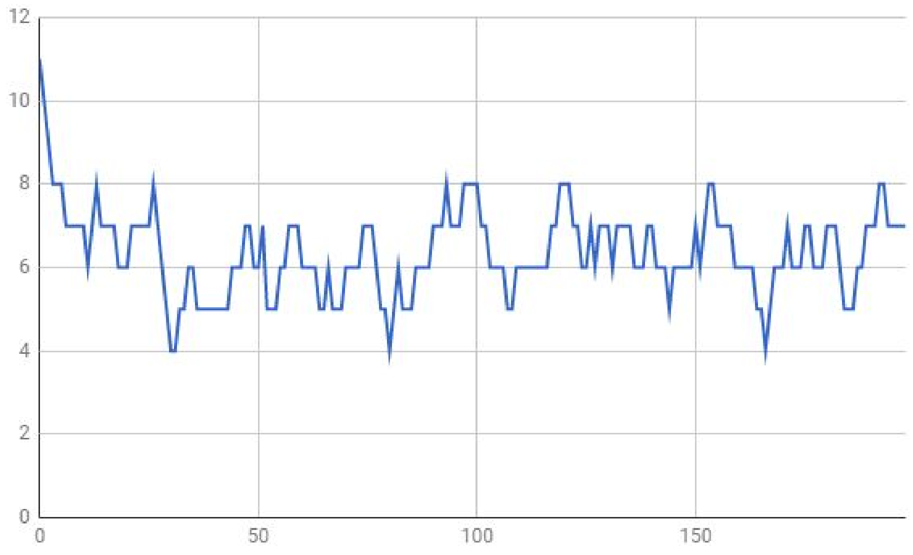
The tabu search is a method really similar to the simulated annealing, we have to create a neighborhood and find the best neighbor to move from the current state, the principal difference is that in this method we have a memory that will not let the system repeat solution that were choose recently.

To do this we use a short-term memory and a long-term memory in the short-term memory we store the states that were chosen recently and assure that we choose other solutions from the neighborhood that were not visited.

The long-term memory store significant solutions such as solutions that were the best of its neighborhood or solutions that were important in the resolution of the program.



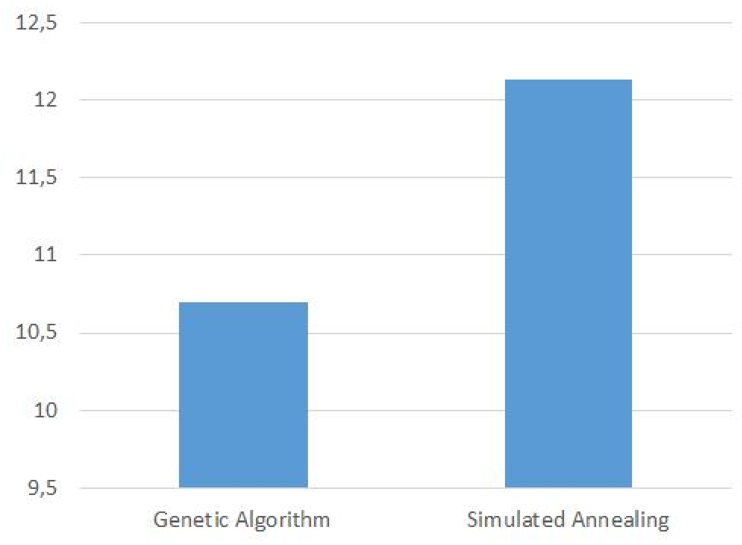
The problem with this method is that it is a little bit more based on chance so, we can not predict when it is going to find a solution. In the previous graph we can see that it found the solution around the iteration 130 but as we can see in the next graph



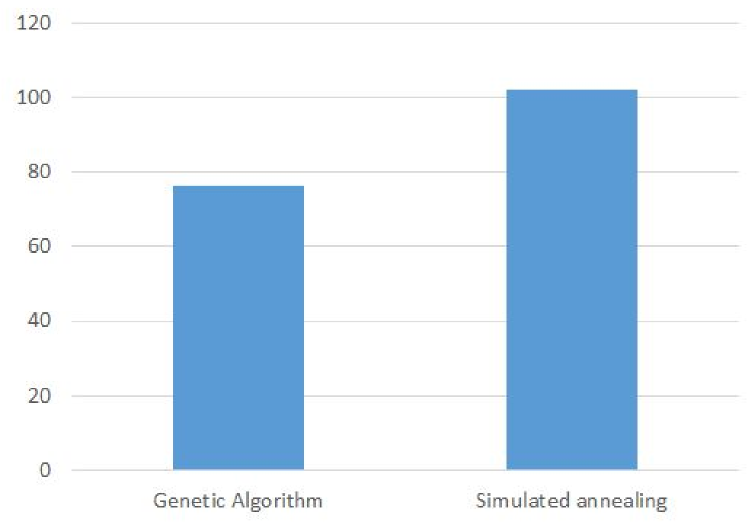
the solution is found in less than 50 iterations. That is why we run the algorithm several times and found a average number of iteration where the solution is always found and use that to measure the number of iterations needed to solve the problem.

COMPARATIVE

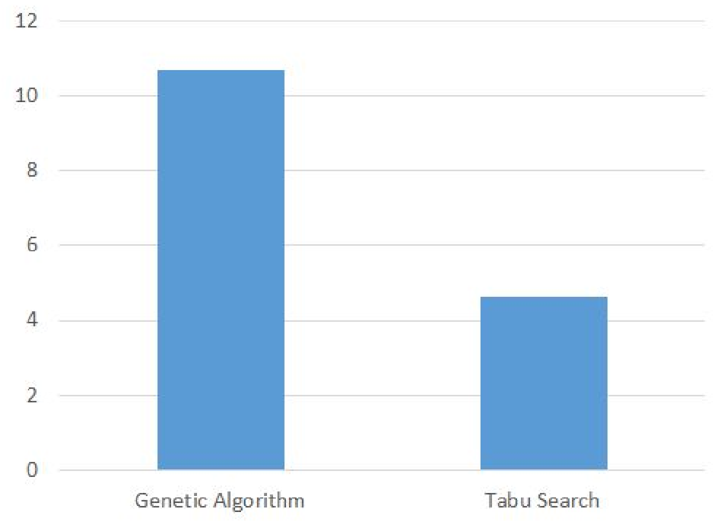
Here we will try to compare the result found with the various algorithm in the same graph



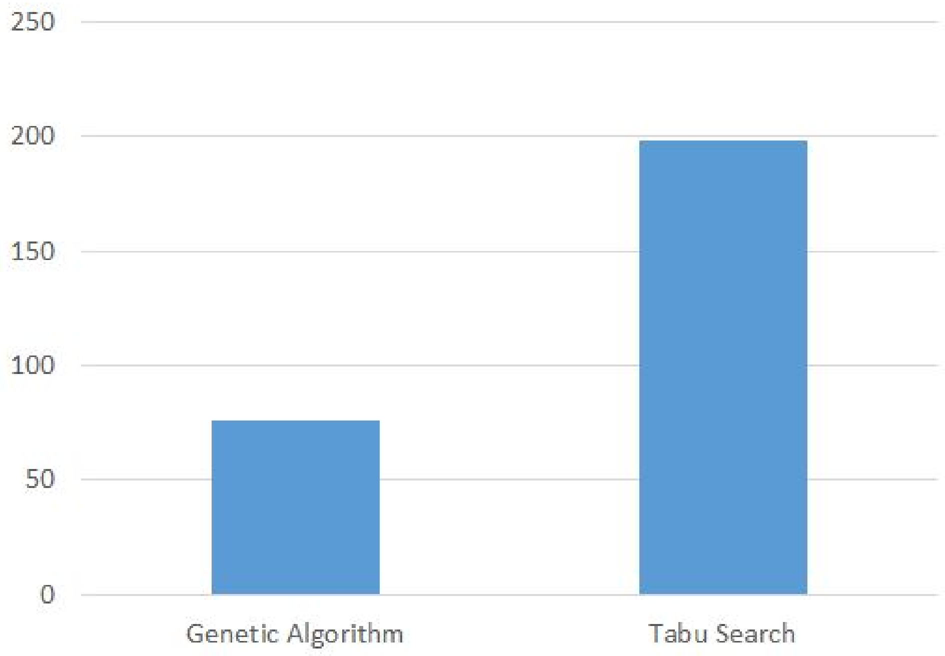
This graph correspond to the time needed by the genetic and simulated algorithm, as we can see the genetic algorithm is much faster in this case, because the graph used is not really big but in cases where we need to use a lot of data this algorithm is even slower than the simulated annealing.



This graph show us the number of iterations that were used with each algorithm as we can see the genetic algorithm find the solution with a smaller number of iteration due to the population used, in the simulated annealing algorithm we just have one state at a given time but in the genetic algorithm we use multiple individuals.



This graph shows the time consumed by each algorithm as we can see the genetic algorithm takes more time than the tabu search but as we can see in the next graph it takes less iterations. That is because the genetic algorithm has to deal with more data than the Tabu search compromising the time consumed



REFERENCES

Genetic algorithm

https://github.com/mvpcom/ColoringGraph/tree/master/ColoringGraph

https://github.com/zare3/Graph-Coloring-Using-Genetic-Algorithms

Computing and Informatics, Vol. 24, 2005, 123–147 “EFFICIENT GRAPH COLORING WITH PARALLEL GENETIC ALGORITHMS” by Zbigniew Kokosinski, Krzysztof Kwarciany, Marcin Kolodziej

“Genetic Algorithm Applied to the Graph Coloring Problem” by Musa M. Hindi and Roman V. Yampolskiy

Simulated Annealing

https://github.com/fakufaku/graphcoloring_mcmc

Slides from the course

Tabu search

Computing 39, 345- 351 (1987) “Using Tabu Search Techniques for Graph Coloring” by A. Hertz and D. de Werra, Lausanne

Slides from the couse