Graph coloring problem with Red Deer Algorithm

BELKESSA Linda
Higher National School of computer science
Algiers, Algeria
il_belkessa@esi.dz

KHAMOUM Amel
Higher National School of computer science
Algiers, Algeria
ia_khamoum@esi.dz

LAMDANI Wilem
Higher National School of computer science
Algiers, Algeria
iw_lamdani@esi.dz

LAOUADI Abir
Higher National School of computer science
Algiers, Algeria
ia_laouadi@esi.dz

MABROUKI Mohamed Laid Malik
Higher National School of computer science
Algiers, Algeria
im_mabrouki@esi.dz

SEBAA Souad
Higher National School of computer science
Algiers, Algeria
is_sebaa@esi.dz

Acknowledgement—We would like to earnestly acknowledge the sincere efforts and valuable time given by our class teacher Mr. KECHID Amine and respected teacher Mme. BESSEDIK Malika. Their valuable guidance and feedback has helped me in completing this project.

***Abstract*—This paper aims to solve a famous combinatorial optimization problem, that of graph coloring. We use a novel metaheuristic called the Red Deer Algorithm (RDA), we also explore different ways to improve the latter in order to achieve better results.**

Keywords—graph coloring, red deer, metaheuristics, hybridation, combinatorial optimization

I. Introduction

Since the beginning of graph theory more than 160 years ago, graph coloring has become one of its most crucial components. Although the graph coloring issue appears to have a straightforward formulation, its abstract representation of the real world shows how intricately related various things are. By experimenting with different graph coloring techniques, graph coloring can be used as an effective or even powerful tool to address a wide range of real-world issues.

II. Definitions

A. Definition of graph

A graph G is a pair (V, E), noted as G = (V, E). V is a non-empty set called vertex set, including all vertices. | V | denotes the number of vertices. E is the edge set including all edges, i.e., pairs of vertices. Note that the same pair of vertices can appear more than once in E. If edges have the direction, the graph is called directed graph; otherwise, it is called undirected graph.

B. Definition for graph coloring problem

A proper coloring of a graph is an assignment of colors to the vertices of the graph so that no two adjacent vertices have the same color.

Usually we drop the word "proper" unless other types of coloring are also under discussion. Of course, the "colors" don't have to be actual colors; they can be any distinct labels—integers, for example. If a graph is not connected, each connected component can be colored independently; except where otherwise noted, we assume graphs are connected. We also assume graphs are simple in this section.

If a graph is properly colored, the vertices that are assigned a particular color form an independent set. Given a graph G, it is easy to find a proper coloring: give every vertex a different color. Clearly the interesting quantity is the minimum number of colors required for a coloring. It is also easy to find independent sets: just pick vertices that are mutually non-adjacent. A single vertex set, for example, is independent, and usually finding larger independent sets is easy. The interesting quantity is the maximum size of an independent set.

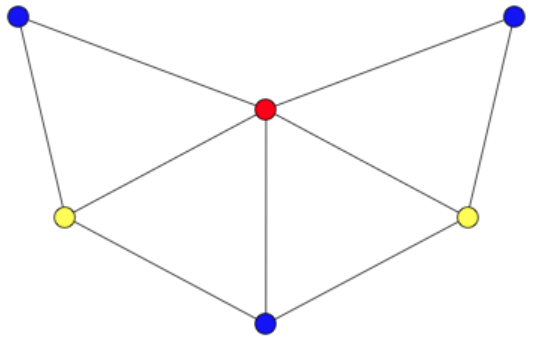

FIG1. A valid graph coloring

C. Chromatic number

The minimum number of colors needed to color a graph G is called its chromatic number. A graph coloring for a graph with 6 vertices. It is impossible to color the graph with 2 colors, so the graph has chromatic number 3.

The RDA starts with an initial random population of size Npop where each individual represents an RD. A number of best RDs among the population is selected as male RD and the rest of them are called hinds.

First of all, the male RD should roar. Based on the power of the roaring process, the male RDs are divided into two groups : commanders and stags. After that, commanders and stags fight against each other in order to own their harem. A harem is a group of females RD, i.e. hinds. Besides, harems

are formed by commanders, this means that only commanders have harems. The number of hinds in harems is directly related to the commanders' abilities in the roaring and fighting process. Consequently, commanders mate with a number of hinds in harems. Note that the other males (i.e., stags) mate with the nearest hind without considering the limitation of the harem. Offsprings are the result of the mating process, and an intermediate population is formed. In order to verify if the stop condition is satisfied, we need to select individuals from this generation that can perform in the next one. Then, we verify the stop condition, if satisfied, we can stop the algorithm.

The main steps of the algorithm are :

- Generating randomly an initial red deer population and selecting males and hinds.
- The roaring process of males : Regarding the solution space, we find the neighbors of the male RD, and if the objective functions of the neighbors are better than the male RD, we replace them with the prior ones.
- Selecting a percent of the best male RDs as Commanders.
- The fighting process between commanders and stags : We let each commander fight with stags randomly. Regarding the solution space, we let a commander and a stag approach each other. So, we obtain two new solutions and replace the commander with a better one.
- Form Harems : A harem is a group of hinds in which a male commander seized them. The number of hinds in harems depends on the power of male commander. We define the power of the commander as its OF. To form the harems, we divide hinds among commanders proportionally by:

$$V_n = v_n - \max_i\{v_i\}$$

where vn refers to the power of the nth commander (i.e., its OF) and Vn is its normalized value. To calculate the normalized power of commanders, the following equation is provided to achieve this goal.

$$P_n = \left| \frac{V_n}{\sum_{i=1}^{N_{Com}} V_i} \right|$$

- Mate commander of a harem with α percent of hinds in his harem.
- Mate commander of a harem with β percent of hinds in his harem.
- Mate stag with the nearest hind.
- Select the next generation.

The stopping condition may be the number of iterations, the quality of the best solution ever found or a time interval.

Generally, the mentioned steps of the RDA are designed in a way to consider the exploitation and exploration phases satisfactorily.

Accordingly, the roaring of male RD is the counterpart of local search in solution space to improve the exploitation properties. Similarly, the fighting between commanders and stags is also considered as local search; however, in this process, we only accept the better-observed solutions. This step mainly considered the exploitation characteristics as well. After that harems are formed and allocated to the commanders according to their power. This step helps the algorithm to do the exploration phase. The flowchart below presents the structure of the RDA. The blue boxes represent the intensification phase, the red ones the diversification and the green one shows the part of the algorithm to escape from the local optimum. The mix of those techniques is the strength of this algorithm.[1]

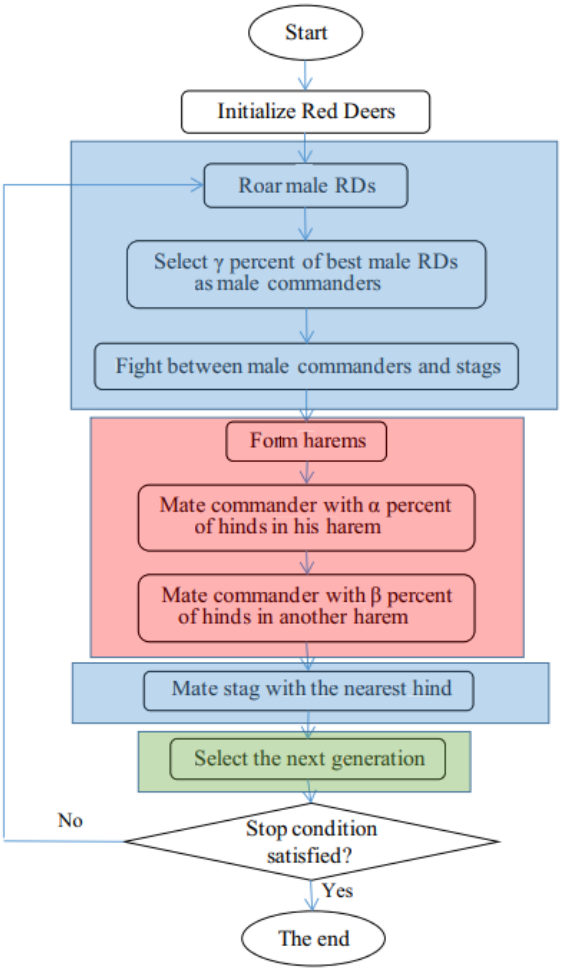

FIG2. General flow of steps of the RDA algorithm

III. GRAPH COLORING ALGORITHMS

Numerous methods have been used thus far to overcome the issue of graph coloring. While there are very few exact algorithms, heuristic methods have been presented. In actuality, many heuristic algorithms use specific exact algorithms. To look for the most advantageous local solution, while the global optimization is handled by their particular heuristic approach. Thus, in these kinds of

heuristic algorithms, both exact methods and heuristic strategies are indispensable.

A. Exact algorithms

As a simple and direct exact method for combinatorial optimization problems, the enumeration method can get exactly the best coloring solution for a graph. Nevertheless,

As it is mentioned before, the graph coloring problem is a NP-complete problem. In other words, the best result cannot be found with the simple enumeration method in a polynomial time for an arbitrary input graph. Searching the best solution with the enumeration method is almost impossible while the number of vertices becomes large enough. In spite of this, studying different enumeration methods can make a worthwhile contribution to some useful exact methods, such as Branch & Bound.

Exact - or complete - methods guarantee a solution optimal to a given optimization problem instance. They are generally based on the tree search and on the enumeration of the solution space. They are used to find at least one optimal solution to a problem. The best-known exact algorithms are the split-and-evaluate method -BranchBound-, dynamic programming, and linear programming. And although they offer the best possible solutions to problems, these methods 11 suffer from what is called the "combinatorial explosion": The number of combinations increases with the dimension of the problem, the execution time is exponential. Thus The efficiency of these algorithms is only promising for instances of small size problems.

B. Heuristic algorithms

The majority of recent research on graph coloring appears to use the heuristic approach. Significant research has been put into creating different heuristic tactics to improve the graph coloring capabilities of algorithms. Heuristic approaches can find a suboptimal answer in a reasonable amount of time even while they cannot find the ideal solution. They therefore dominate a wide range of applications.

The greedy approach is the most fundamental heuristic algorithm. The greedy technique attempts to assign the used colors that are usable in a specific coloring sequence to the uncolored vertices. This algorithm is so effective that it serves as the foundation for numerous additional heuristic techniques. It can always obtain the coloring result in a polynomial amount of time because of its extremely low worst-case computational cost of O(n). The outcome is occasionally so poor, though, that it falls far short of the ideal solution. Therefore, improving the greedy technique is required.

However, the result is sometimes so bad that the gap with the optimal solution is large. Hence the improvement of greedy methods becomes necessary. It is known that the node coloring sequence in the greedy algorithm plays a vital role in enhancing its performance, since at least one of these node coloring sequences leads to the best coloring solution. A significant impetus for greedy methods is to find a good coloring sequence. But traversing all node coloring sequences is almost impossible as the number of nodes increases.

A simple method is to arrange the coloring sequence in the decreasing order of the nodes' degree, which can apparently improve the result of the greedy algorithm. Nevertheless, this method cannot get the best coloring solution, so further improvement of the greedy algorithm is expected.

A famous heuristic method, called DSATUR, is proposed using the saturation degree to decide the node coloring sequence based on a greedy algorithm. The saturation degree of a vertex is the number of different colors in its neighboring vertices which have been colored. The first node to color is the one with the maximal degree. Then, the node coloring sequence is determined in the decreasing order of the nodes' saturation degree. Due to the improvement of the node coloring sequence, DSATUR performs much better than the simple greedy method. It becomes an essential submodule for several heuristic methods.

C. Metaheuristics

The word metaheuristic is composed of two Greek words: meta and heuristic. The word meta is a suffix meaning beyond, i.e. higher level. Metaheuristics are methods generally inspired by nature. Unlike heuristics, they are reusable and can be applied to several problems of a different nature. 12 Mainly dedicated to solving optimization problems. Their goal is to achieve a global optimum while escaping local optima. Metaheuristics include methods that can be divided into two classes:

1. Single-solution metaheuristics: These methods process only one solution at a time, in order to find the optimal solution.

2. Metaheuristics with population of solutions: These methods use a population of solutions at each iteration until the global solution is obtained.

These kinds of methods include tabu search, genetic algorithm, ant colony optimization, simulated annealing, artificial neural networks and so on.

Applying the proposed RDA to some benchmark function, shows its ability in dealing with different types of optimization problems.

By solving multiple benchmark functions and important engineering as well as multi-objective optimization problems, the superiority of the proposed RDA shows in comparison with other well-known and recent meta-heuristics.

The RDA is also tailored to solve a generalized network design problem that aims to design a network with minimal cost while satisfying several practical

constraints. To assess the performance of this new bio-inspired metaheuristic on solving such NP-hard problems, computational experiments were conducted on Benchmark as well as real-world instances. Computational experimentation illustrates the accuracy of the RDA that outperforms all of the existing recent metaheuristics.

In the next step of this article, we are going to discuss the adaptation we made to the Red Deer Algorithm to the Graph Coloring Problem, by describing every step of the algorithm. Then we are going to see the tests and the results of the implementation of RDA. Finally we are going to conclude with a comparison between the RDA and other methods (exact methods and heuristics).

IV. Formulation of the problem

A. *Representation of data*

1) Uncolored graph

The simplest way to represent an undirected graph is to use its adjacency matrix. An adjacency matrix is a binary

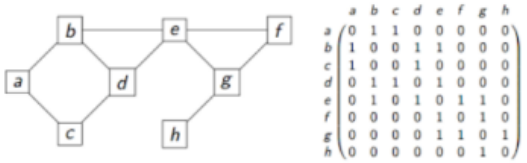

FIG3. adjacency matrix for a given graph

2) Colored graph

The colored graph - i.e. the solution - is represented by a table where each box represents the number of the color chosen for the node whose number is the index of its location in the array. The indexes are taken from the adjacency matrix mentioned above.

B. *Graph coloring applications*

Graph coloring problem has immense industrial and commercial applications, especially in the case of dealing with interference and conflict. Below only some traditional applications are listed while the potential applications of graph coloring are much more than these.

Frequency allocation: In the telecommunication networks, one of the key problems is the frequency allocation because of the limited physical frequency spectrum resource. Iber of frequency resources for communication while the mutual interference is minimized generally, the optimization problem for frequency allocation is to use a minimum number. If each device or communication link corresponds to a vertex, two vertices are connected by an edge if their corresponding frequencies have conflict. Let each frequency resource correlate with a color, then the objective to minimize the number of frequencies used in the network is transferred to color the graph with the minimum number of colors. In this way, the frequency allocation problem is naturally transferred into a graph coloring problem. So the frequency allocation efficiency largely depends on the performance of the used graph coloring algorithm.

Scheduling: For diverse scheduling scenarios, the essence is often to coordinate various collisions or interference. The graph coloring model is pretty good at describing this kind of relationship. Therefore, a lot of scheduling problems are formulated as graph coloring problems, such as time table scheduling, aircraft scheduling, seating plans designing, sports leagues designing and so on.

Register allocation: Register allocation aims to use a large quantity of variables on the Central Processing Unit (CPU) which has only a limited number of registers. Supposing each variable is represented by a vertex, if the variables cannot use the register simultaneously, there will be an edge connecting their corresponding vertices. Thus, a conflict graph is formed well defining the relationship of interference between variables. Then the graph coloring algorithm is used to get the minimum number of registers required.

For the adaptation of this algorithm to our problem, we are going to use the following structures :

- a Red Deer is an array of colors where the index represents the vertex.
- The graph is represented by an adjacency matrix

The diagram below shows each step of the adaptation of RDA to the Graph Coloring Problem.

The algorithm starts the initialization of the chromatic number with the minimum between the value given by a greedy heuristic and the maximum vertex degree + 1. The use of this heuristic at this point is considered as a form of hybridization.

Then, we initialize the population of size **Npop** randomly, each Red Deer represents a solution with conflicts. We call a conflict a situation where two adjacent vertices are colored with the same color.

After generating the population, we proceed to the selection of males and hinds. We choose males as the **Delta%** best Red Deers and the rest of the population will be considered as hinds. We evaluate the Red Deers based on their Objective Function (OF or fitness) which represents for every Red Deer the number of conflicts it contains.
The next step is the roaring process where we try to improve the quality of the males. We generate the neighborhood of a male by modifying randomly the color of the vertices that causes a conflict. We stop the process when we find the first solution that improves the fitness of the male. The intensification phase is done by the roaring process where it constitutes the second form of hybridization : A Local Search with first-improve technique inside an evolutionary algorithm (LTH).

We sort the males and select among them the best **Gamma%** males as commanders and the rest of the males as stags.

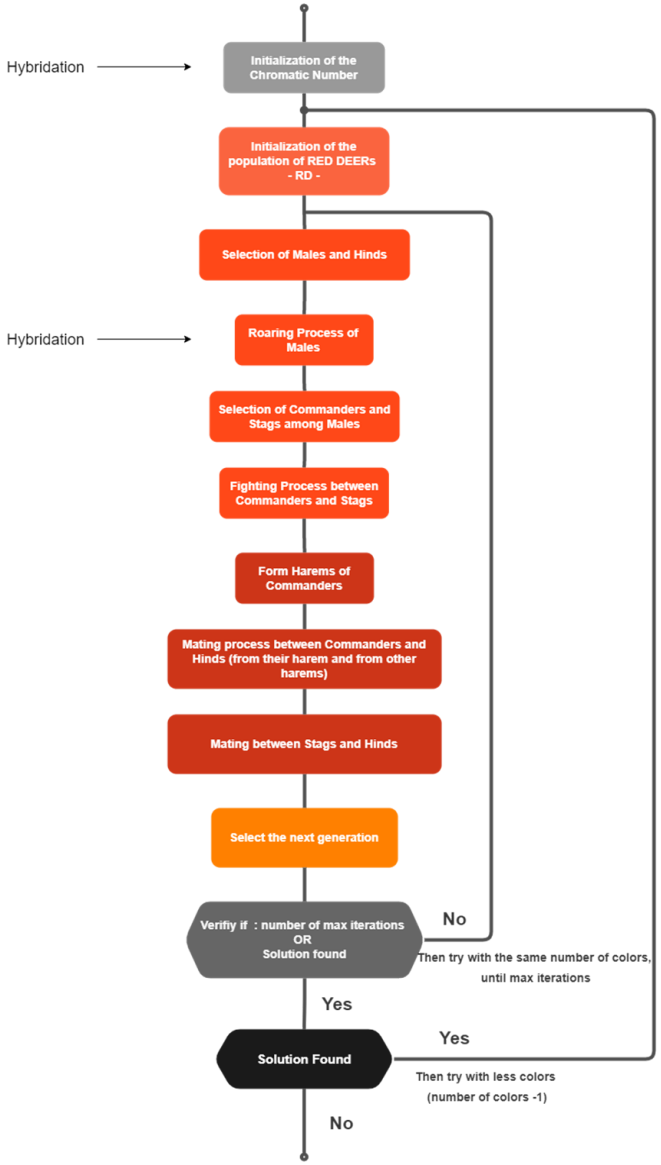

FIG4. flow chart detailing the various steps of the algorithm

We confront the commanders and stags in the fighting process where we choose for each commander a stag and we generate 2 other solutions and replace the vertices that cause a conflict of the commander with those of the stag and vice versa. We choose the best solution from the four (commander, stag and two other solutions) as the new commander. The stag will remain the same as before the fighting process.

We then calculate the power of the commanders based on their fitness among all the commanders. We form the harems of the commanders, and their size (number of hinds in the harem) is proportional to the power of the commander. It's important to notice that only commanders will have harems not stags.

After forming the harems, we start the mating process between Commanders and **Alpha%** of the hinds in their harem and **Beta%** of hinds of other harem. We continue with the mating process with stags and randomly selected hind regardless of its harem. The result of the mating process will be the offspring.

We proceed to the selection phase where we select the Red Deer for the next generation. We sort all the population (males, hinds, and offsprings) and chose a percent of the best solutions and the rest of the worst solutions. We use this technique to accomplish the diversification and escape for a local optimum.

Finally, we chose the best solution among all the population and compare its fitness. If the stopping condition was verified because we found a feasible solution with the current chromatic number, let's name it K. We try with the same population to find if the graph is K-1 Colorable. Otherwise, if we stopped the algorithm due to the max number of iterations, this means we gave enough chance to K as the chromatic number and we didn't find a solution, we can conclude that the graph is K-1 Colorable.

V. TESTS AND RESULTS

After having exposed the technical details concerning the implementation of our solution, we move this part of the article that will test how the algorithm will pass several tests considered to forge even further the quality of the solution by optimizing the parameters of the technique for a given dataset, and most importantly comparing its' efficiency and execution time to other techniques introduced in past works related to the same problem of graph coloring.

A. Evaluation of the algorithm

An algorithm is characterized by its complexity and execution time, in our case we consider furthermore the quality of the provided solution, in other words the number of distinct colors used to fully color the graph.

To objectively and efficiently evaluate the algorithm, we will consider several benchmark datasets that provide a variety of graphs with multiple nodes which number ranges from relatively small to very large, though the most interesting case is that of reasonably large graphs, where some classic techniques fail to provide sufficiently satisfying results, thus we will show how RDA could outperform them in terms of execution time and other considered metrics.

Instances used are in DIMACS standard format. Instances in .col.b are in compressed format (a binary format). Each instance includes the information: (nodes, edges), optimal coloring, source.

Aside from the data, we will also evaluate the considered metrics for multiple configurations and combinations of parameters in order to conclude the best performing (this of course varies accordingly with the dataset used), to do so we'll perform a Grid Search. We will also discuss the results presented in other research papers which consider the generic form of the algorithm without it being applied to a specific problem.

B. *Testing plan*

The steps of the testing phase are as follow :

1) **Performance testing :** Executing the algorithm on different benchmark datasets, the targeted variable is the number of nodes of the input graphs, we want to study here the evolution of the execution time and quality of the output results conditionally to the size of the input.

2) **Parameters tuning and optimization :** Executing the algorithm by taking different parameters configuration and performing a grid search.

3) **Comparative study :** comparing the results of RDA algorithm to other techniques mainly:
   a) The exact approach : Branch and bound
   b) Heuristics : Dsatur, Greedy coloring, Max-Stable
   c) Evolutionary Metaheuristics : GA (Genetic algorithm)
   d) Swarm intelligence metaheuristics : ACO (Ant Colony Optimization)

C. *Characteristics of the machine and execution environment*

- CPU model and architecture : Intel(R) Core(TM) i7-8550U CPU
- Frequency and clock-rate : 1.99 GHz
- RAM capacity : 8,00 Go
- Python interpreter version : python 3.10.1

D. *Execution results*

1) Using the myciel4.col instance

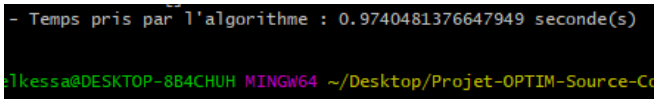

FIG4. execution time taken for myciel4.col instance

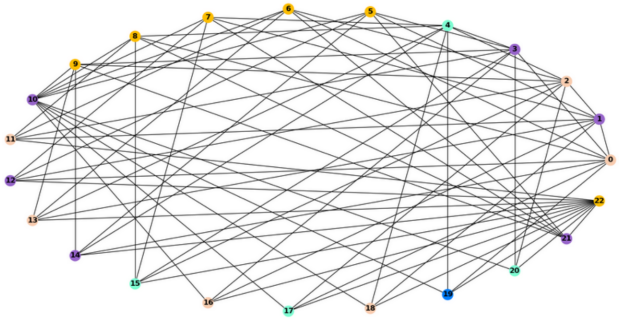

FIG5. k-coloring found for myciel4.col instance

2) Using the queen5_5.col instance

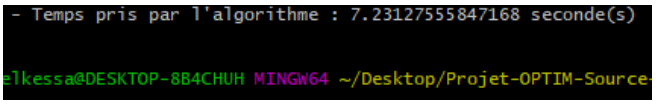

FIG6. execution time taken for queen5_5.col instance

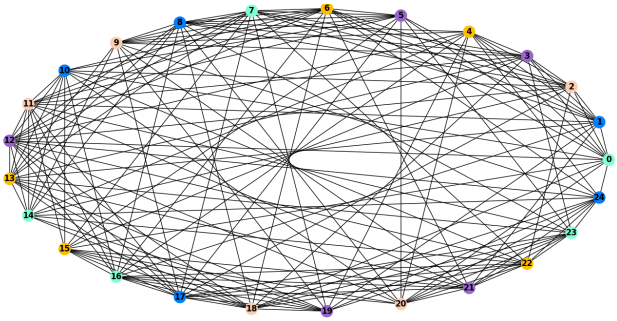

FIG7. k-coloring found for queen5_5.col instance

E. *Parameters tuning*

Grid search [2] is a process that searches exhaustively through a manually specified subset of the hyperparameter space of the targeted algorithm. We've conducted this experiment on the following parameters in order to study their effect on the overall performance and execution time for then choosing the best ones relatively to the given dataset :

- Population size
- Number of maximum iterations
- Number of generations
- Algorithms' parameters : alpha, beta, gamma, delta

We've obtained the following results :

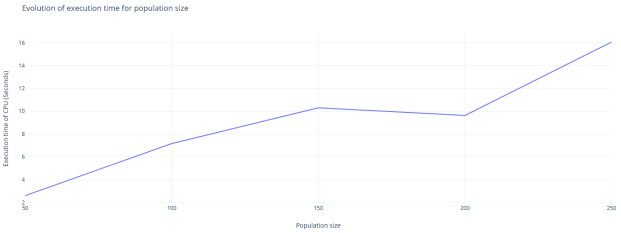

FIG8. Evolution of execution time for population size

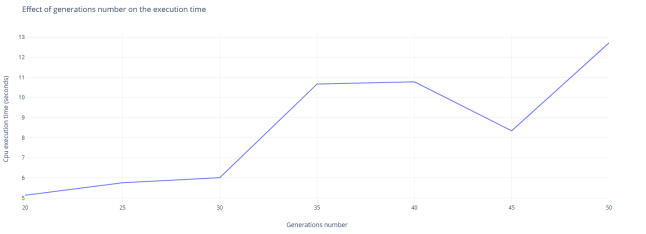

FIG9. Effect of generations number on the execution time

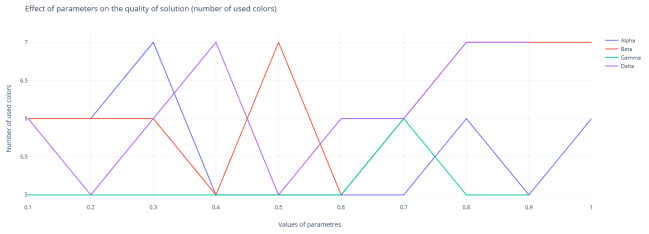

FIG10. Effect of RDA parameters on the number of colors used

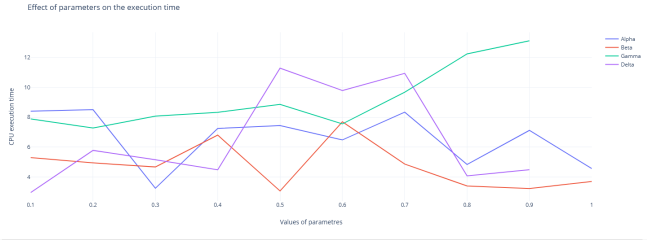

FIG11. Effect of RDA parameters on the execution time

F. *Comparative study*

Algorithms	**Execution time (in seconds)**		
	myciel4	***queen5_5***	***queen7_7***
Exact methods			
Branch & Bound	5.0406s	0.0015s	0.7767s
Heuristics			
Greedy coloring	0.00005s	0.00007s	0.0002s
DSatur	0.0003s	0.0007s	0.0031s
Max stable (with greedy coloring)	0.0017s	0.0023s	0.0078s
Meta-heuristics			
Recherche Tabou (RT)	0.0007s	0.0011s	0.0034s
Algorithme génétique (AG)	34.72391s	90.6011s	183.8560s
New meta-heuristic			
Red Deer Algorithm (RDA)	5.3178s	7.3244s	12.52314s

TABLE I. EXECUTION TIME RESULTS FOR DIFFERENT ALGORITHMS

G. *Discussion of the obtained results*

At first glance RDA seems to behave well in terms of execution time compared to GA which in this case is the closest in terms of complexity and overall idea. It even outperforms it for multiple instances of the GCP benchmark datasets.

This is mainly due to avoiding intensification where it's not needed and favoring diversification instead, we're pointing here to the fact that we went for a random choice of next generation's deers as an effort to bring up further the results which seem to have positively turned out.

REFERENCES

[1] Fathollahi-Fard, A.M., Hajiaghaei-Keshteli, M. & Tavakkoli-Moghaddam, R. Red deer algorithm (RDA): a new nature-inspired meta-heuristic. *Soft Comput* **24,** 14637–14665 (2020). https://doi.org/10.1007/s00500-020-04812-z

[2] Liashchynskyi, Petro & Liashchynskyi, Pavlo. (2019). Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. https://www.researchgate.net/publication/337916821

[3] Jianding Guo. Theoretical research on graph coloring : Application to resource allocation in device-to-device 4G radio system (LTE). Networking and Internet Architecture [cs.NI]. Université Bourgogne Franche-Comté, 2018. English. ⟨NNT : 2018UBFCA007⟩. ⟨tel-01867956⟩