

GENETIC ALGORITHM FOR GRAPH COLORING

Everaldo Pereira Gomes

Federal University of Fronteira Sul
Rodovia SC 484 - Km 02, Fronteira Sul, 89815-899, Chapecó-SC, Brazil
everaldo.gomes@estudante.uffs.edu.br

Felipe Grando

Federal University of Fronteira Sul
Rodovia SC 484 - Km 02, Fronteira Sul, 89815-899, Chapecó-SC, Brazil
felipegrando@uffs.edu.br

Andrei Braga

Federal University of Fronteira Sul
Rodovia SC 484 - Km 02, Fronteira Sul, 89815-899, Chapecó-SC, Brazil
andrei.braga@uffs.edu.br

Samuel Feitosa

Federal University of Fronteira Sul
Rodovia SC 484 - Km 02, Fronteira Sul, 89815-899, Chapecó-SC, Brazil
fsamuel.feitosa@uffs.edu.br

ABSTRACT

Finding solutions to problems in which there is a vast search space can be a task that consumes a lot of time and other computational resources. Graph coloring is an example of a complex problem that has many real applications. For this research, it was developed and tested a genetic algorithm that in some cases, could be efficient in solving the optimization problem associated with graph coloring, having better results than other genetic algorithms in the literature. To identify the most suitable parameters for the genetic algorithm, this research proposed a methodology where different combinations of parameters were implemented and tested, and the efficiency results of several benchmarks were empirically collected. By identifying the most suitable configuration for the graph coloring problem, we hope to contribute to the most varied research fields and in our daily life problems.

KEYWORDS. Heuristic search. Genetic algorithms. Graph coloring.

Paper topics. Heuristic search; Graph.

1. Introduction

Using graph theory to solve problems has brought innovative solutions to various applications [West, 2001]. Over time, more data could be managed to be used in a variety of problems, such as allocation of registers used in compilers, frequency assignment, and scheduling. Real problems like these can be translated into instances of classic graph coloring which is a combinatorial and NP-Complete problem [Chartrand and Zhang, 2009; Bondy and Murty, 1976; Rahman, 2017; Sivanandam and Deepa, 2008]. Since for these kinds of problems, we do not have an efficient solution for all instances, and searching for solutions can take a long time depending on the search space, therefore, many strategies and algorithms were proposed to solve graph coloring instances efficiently but there are still areas for further research, improvements, and breakthroughs using different techniques, using a heuristic search that severs to find a solution more quickly.

A set of techniques that gained popularity for achieving promising results in many combinatorial problems is commonly known as genetic algorithms, a meta-heuristic capable of solving optimization and search problems inspired by the evolution theory [Sivanandam and Deepa, 2008].

In the literature, we came to know that most of the papers used their genetic algorithm with some other techniques in order to increase the chances of getting better results and performance. Examples of these techniques are SPCGX, Tabu search, conflict genes removal (CGR), ANTCOL, Max-Min ant system (MMGC), where some had more than one crossover, and mutation methods [Marappan and Gopalakrishnan, 2013; Fleurent and Ferland, 1996; Assi et al., 2018; Mumford, 2006; Sethumadhavan and Marappan, 2013; Salari and Eshghi, 2005].

With this research, we intended to find out whether a genetic algorithm could find or not good solutions without using any specific techniques, by only using different parameters in all the algorithm's phases, until find one parameter that could give reasonable results for the graph coloring problem. These results are being compared with three other papers [Douiri and Elbernoussi, 2015; Abbasian and Mouhoub, 2011; Hindi et al., 2012] because they used the same instances as we did to perform the tests, therefore, having a way to compare our algorithm. To get these results, we tried to determine what is the best way to initialize the population, including its size, what would be the best way to perform crossover and mutation, as well as the mutation rate, which heuristic type would be a good choice to implement, which instances would be used to test the algorithm, and by doing this, decide which stop criterion would be used, what would be a fair way to compare the results in order to decide which configuration performed better for coloring all the vertices with a minimum number of colors without conflicts.

This paper is organized as follows. First, it is presented a brief introduction of important topics to understand the general idea of this paper. Section 2, covers some state-of-art works, which cover some algorithms and techniques to solve graph coloring problems. Section 3 shows the methodology, where is described all the steps taken to find our results. Section 4 is about the empirical tests, which show the configurations and parameters used to test the proposed algorithm. In Section 5 we analyze and compare the algorithm, pointing out important aspects. Section 6 talks about the results and Section 7 it is given our conclusion and ideas for future works.

1.1. Heuristic search

Heuristic search is an evolution method to solve a problem using the reference of the previous result to find better solutions, without using a lot of time and computational resources. It contains some predefined rules, which are repeated in every new solution and it is necessary to have one or more stopping criteria to indicate that the desirable solution was found. By having this characteristic, the heuristic search can be used in several fields, and there are several algorithms

used to do such a search, for example, A* search, Tabu search, and best first search [Edelkamp and Schrödl, 2012]. Another algorithm is called genetic algorithm which is widely used.

Genetic algorithms came up with the proposal to imitate human evolution, where the strongest individuals prevail, and these individuals will generate stronger children and build an increasingly promising population. The first implementation of this idea was in 1975. This type of algorithm can be used to solve many problems, such as scheduling, finance, vehicle routing, crew rosters in aviation, and graph coloring [Ertel, 2017; Sivanandam and Deepa, 2008; Edelkamp and Schrödl, 2012].

A genetic algorithm is composed of a series of functions that, when combined can reach the expected result, and depending on the size of the search space, the result can be obtained quickly or not. Each phase produces an output for the next one, the initial population is evaluated in the object function, which will give the necessary information for the selection function to choose the individuals, these individuals will generate offspring in the crossover and some of them will be mutated, which will be part of the population. The following figure illustrates these functions.

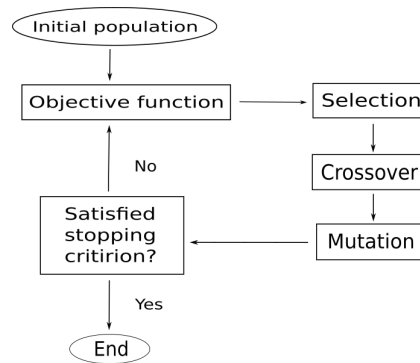


Figure 1: Steps in a genetic algorithm

By using these functions, we can solve many types of problems, including those related to graph coloring.

1.2. Graph

A graph is a set of points called vertices, which are connected by lines called edges. There are some types of graphs, such as complete, directed, undirected, and weighted graphs, which are used to represent situations in our lives [West, 2001]. For graphs, there are several techniques that can be used to solve problems, such as finding the shortest path between two vertices we can use the well-known Dijkstra algorithm. Other problems such as scheduling require other techniques, such as graph coloring.

In this kind of problem, we have to define constraints between situations or objects, which cannot share the same resource (e.g. space, time, objects). For example, in a warehouse where there are several chemical products, and for some reason the same products cannot be close to each other [Bondy and Murty, 1976; West, 2001]. How to store all these products, can graph coloring help to solve this problem?

Figure 2 shows a graph with vertices painted using the minimum colors required to not cause any conflicts. Figure 3 illustrates a graph with conflicts, which has the optimal amount of colors, but the colors are disorganized causing conflicts on adjacent vertices.

There are several benchmarks available. The most used are those that come from DIMACS collection such as anna, david, miles250, games120, huck. These instances are used for

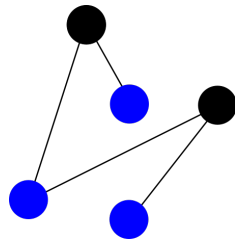


Figure 2: Graph without conflicts

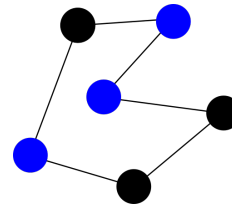


Figure 3: Graph with conflicts

testing algorithms that try to solve graph coloring problems.

2. Related work

Here are shown some of the state-of-the-art papers used as a base for the present paper, which were used to be part of our project, by providing results to be compared with our algorithm.

2.1. Genetic Algorithm Applied to the Graph Coloring Problem

Hindi et al. [2012] proposed an algorithm that made use of two methods for selecting individuals. The crossover was performed using a random cut off point to generate a child, and two more methods for mutating, which depend on the current color of the evaluated vertex. It has a probability of 0.7%, and if this new value at that vertex violates some restriction on graph coloring, another value is chosen at random, and the initial population was randomly generated. A progressive increase in the search for an optimal global solution was obtained, and not stuck in a local solution.

2.2. An efficient hierarchical parallel genetic algorithm for the graph coloring problem

Abbasian and Mouhoub [2011] used an extension of the genetic algorithm called the parallel genetic algorithm but did not make use of any specific technique, just using different parameters. The algorithm was compared with another algorithm called DSATUR, where it was shown that the proposed algorithm used fewer colors. The operations in the algorithm were randomly executed. The crossover was performed using a random cutoff point to generate a child. The mutation keeps running while the mutated gene causes conflicts. It was shown that the algorithm had better results in finding the number of colors used, but in other test cases, they obtained equal results.

2.3. Solving the graph coloring problem via hybrid genetic algorithms

In the approach by Douiri and Elberoussi [2015], the local search heuristic called DBG was used. The proposed algorithm was compared with other algorithms from other authors, and for some instances, the average execution time to find the solution was improved. The initial population is composed of the number of colors, these numbers are generated until a number of iterations is reached or until there is no conflict between adjacent vertices. In the crossover, one cut-off point was chosen randomly with a probability of 0.7%. In the mutation, the invalid colors (i.e. conflicts between vertices) are randomly chosen to be replaced by another one, with a probability of 0.15% of values ranging from 250 to 2000. The algorithm showed to be better for some instances, and the average execution time to find the solution was improved.

3. Methodology

In this section, we present the steps taken to build, test, and analyze the results obtained by our genetic algorithm approach. First, we present all the tested configurations for all phases in the genetic algorithm (i.e. initial population, objective function, crossover, and mutation) and the intuition behind some of the choices made in each of these phases. It is also presented some additional functions used in an attempt to make the algorithm work more efficiently as well as, the environment used to run the tests. The initial parameters were chosen based on the state-of-the-art papers, and by observing how they behaved in our algorithm, we were able to direct the values that best fit each of the functions.

3.1. Genetic Algorithm

We tested different implementations and parameters in order to define which was the best strategy to obtain the best results for this specific type of problem. These parameters are tested in all phases of the genetic algorithm. Eighty different configurations were tested in total. The source code was written in C++ [Gomes.E.P, 2023]. As an attempt to improve the performance, was used a compiler directive library called OpenMP for parallelization.

3.1.1. Initial population

Two values for the initial population's size which were 100 and 300 randomly generated. These ranges were chosen due to the fact that a lower number of individuals is generally bad because it is important to have a diverse group of solutions, and a larger number increases the time since all the following phases will take longer to be done and also increases memory usage.

3.1.2. Objective function

The objective function will evaluate the number of colors and the number of conflicts for each individual. The population is also sorted by the best solutions (i.e. less number of colors). By doing this, we can separate the best and worst solutions, where the worst parents will be removed later.

3.1.3. Selection

The first 50% of the total population is chosen, which happens to be the best solution for the population. This is important because parents with less number of colors may generate offspring with few numbers of colors too, but this can increase the number of conflicts, where this is treated late with a function to resolve conflicts. The selected individuals are paired to generate offspring with the crossover operators.

3.1.4. Crossover

We configured and tested two strategies of crossover operations, which takes pairs of individuals (I1 and I2) to generate two new offspring. In the first one, each parent solution will pass half of its genes to each one of the offspring. The first half of the first offspring is filled with genes from the individual I1 randomly chosen, and the second half of the first offspring is filled with genes from the individual I2 randomly chosen. The second offspring follows the same principle, as illustrated in Figure 4.

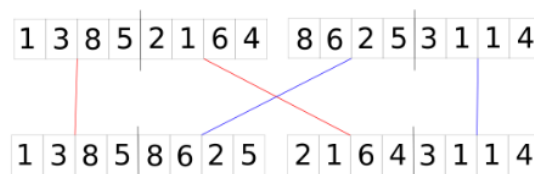


Figure 4: Crossover type A

In the second type of crossover, the chosen individuals are divided into two halves, then producing 4 chunks (I1A, I1B, I2A, I2B). Each one of the new offspring is composed of any combination of these possibilities randomly chosen, where each offspring will have its first half from the I1 and second half from I2 or vice versa. By doing this, will have half of the total population number, where the last 50% of the population's individuals is replaced by the generated offspring, where this last 50% are the worst parents that were sorted previously. One possible example is illustrated in Figure 6.

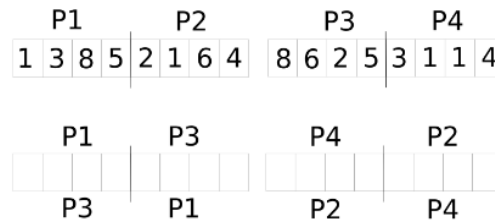


Figure 5: Crossover type B

3.1.5. Mutation

The mutation is performed in some offspring, the quantity goes from 10% up to 100% of the total population size. There are two types of mutation. The first type is responsible for taking the gene that repeats the most and replacing it with the gene that has fewer occurrences, while the second type picks any color randomly chosen to be put in the place of the old color. The gene chosen to be changed regardless of the mutations is randomly chosen.

3.1.6. Stop criterion

Two stopping criteria are considered. The first one is 80 seconds as the maximum execution time, and the second one is when the known optimal solution is found for each instance. The reason for using that maximum time cap of 80 seconds is based on the fact that the results of the related work were also computed in about 80 seconds.

3.1.7. Resolve conflicts

A function is used to check if a conflict exists in the graph and resolves it as follows. The vertex that is causing conflict will be replaced by a new color, starting from the initial color until this new color resolves the conflict in all neighbor vertices, and after changing, all the N-connected vertices are checked as well, it is executed in every individual that passed in the objective function.

3.1.8. Color Reduction

It replaces the first occurrence of the highest color number with a color randomly chosen, which is less than the highest color number, and keeps doing this only if this new color does not cause a new conflict, if there is no way to avoid the conflict, the new color will simply be the highest color number - 1. All these operations are only done in the parent's individuals (i.e. the first 50% of the population size). The other half is composed of the offspring, which was not evaluated yet, therefore it may contain conflicts.

4. Empirical tests

Fifteen instances from DIMACS collection were selected as a benchmark, which were the most used instances in the state-of-art works, which are the most used. Ten independent samples for each instance were executed for each combination of parameters. There were 80 configurations of parameters in total. These configurations are composed of the initial population size, the crossover and mutation used, and the mutation rate. In Figures 6 and 7 show all the configurations and their parameters.

The code was written in C++ and the first type of crossovers is being parallelized using the library OpenMP. A computer with Ubuntu 20.04 LTS 64-bit operational system, with an Intel Core i5 @ 3.20 GHz x4 processor, and 8GB of RAM, was used to run the experiments. The produced results (e.g., number of colors found, time spent, the configuration used, the average of colors found, and its standard deviation) were saved in a file in order to be compared and further analyzed.

Configurations																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Population of 100 individuals																			
Crossover A																			
Mutation A										Mutation B									
Mutation rate																			
0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%	0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%

Configurations																			
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Population of 100 individuals																			
Crossover B																			
Mutation A										Mutation B									
Mutation rate																			
0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%	0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%

Figure 6: Configuration 1 to 40.

Configurations																			
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Population of 300 individuals																			
Crossover A																			
Mutation A										Mutation B									
Mutation rate																			
0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%	0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%

Configurations																			
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Population of 300 individuals																			
Crossover B																			
Mutation A										Mutation B									
Mutation rate																			
0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%	0.1%	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%	0.8%	0.9%	1%

Figure 7: Configuration 41 to 80.

5. Analysis and comparison

To make a reasonable comparison, the average time is given in seconds since the authors' results use seconds as well.

We observed that our algorithm could find the best number of colors quickly for the majority of instances, taking around 20 seconds. In some instances, our algorithm could not find a solution, therefore, used all the available time in an attempt to find a solution, but without success.

6. Results

Using all the steps and decisions shown in our methodology, our algorithm could find the best solutions for 6 out of 15 instances in at least one of the 10 executions. All these instances were small graphs, which ranged from 11 up to 138 vertices.

The best configuration found was configuration number 10. It had 20 colors on average, in comparison with all other configurations. The other configurations had values higher than configuration 10, ranging from 21 up to 25 colors on average. Figure 8 shows the color per configuration of all instances.

Figure 8 shows the average of colors per configuration, where on the left side (i.e. first 40 configurations) we can see small values, and on the right side (i.e. the last 40 configurations) we can see values increasing using the same configurations but only changing the population size.

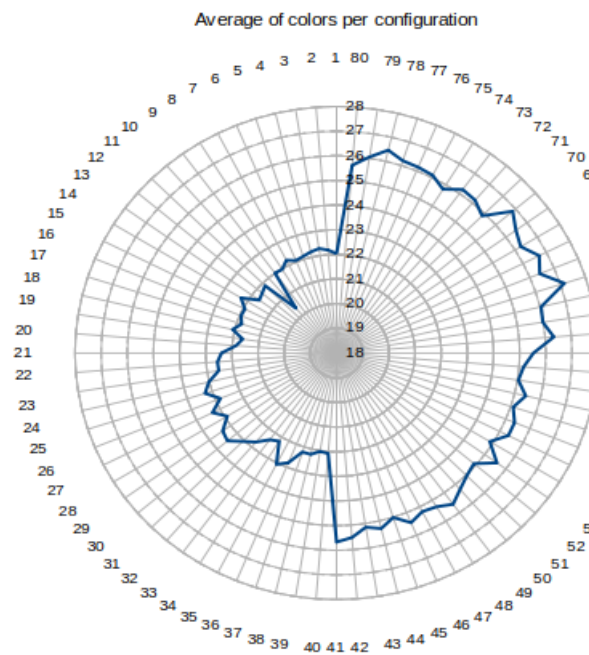


Figure 8: Average of colors per configuration

Looking at Figure 8, was evident that the first 40 configurations were the best. Those configurations shared common parameters. The only different parameter was the crossover. The first half received crossover type A and the second half received crossover type B. These parameters can be seen in Figures 6 and 7. Figure 9 shows the difference between these crossovers, where the red line is the crossover type A and the blue line is the crossover type B. Labels are the configuration used for each crossover, the bottom label for crossover type A and the top label for crossover type B. Figure 10 shows the comparison of the mutation types, which do not have much difference between their results. where the red line is the mutation type A and the blue line is the mutation type B. Labels are the configuration used for each mutation, the bottom label for mutation type A, and the top label for mutation type B.

Figure 11 is the instance in which our algorithm found the optimal color number, but it took longer than the other authors' solution. Figure 12 is the instance in which our algorithm found the optimal color number as well, but it was faster than any other authors' solution. These results can be seen in Table 2.

Table 1 shows the best 4 configurations (i.e. average number of colors found) ordered in ascending order and shows the parameters used.

In order to measure the obtained results by our algorithm, the results are being compared

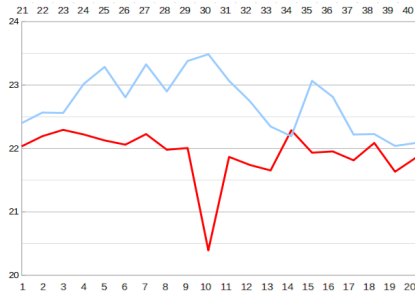


Figure 9: Crossover type A represented by the red color and crossover type B represented by the blue color.

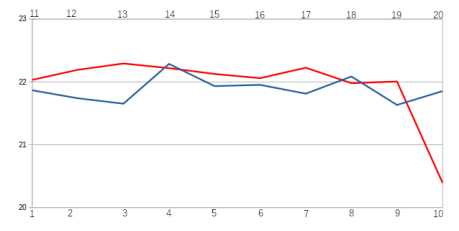


Figure 10: Mutation type A represented by the red color, mutation type B represented by the blue color.

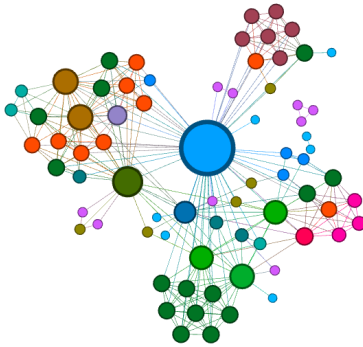


Figure 11: huck instance

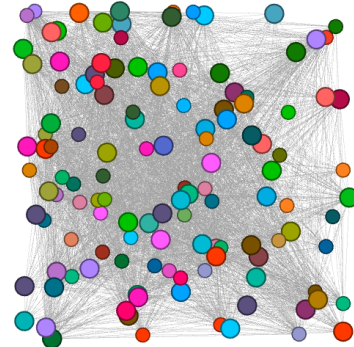


Figure 12: miles1000 instance

Configuration	Population size	Crossover	Mutation	Mutation rate	Avg of colors
10	100	Type A	Type A	1%	20.39
19	100	Type A	Type B	0.9%	21.63
13	100	Type A	Type B	0.3%	21.65
12	100	Type A	Type B	0.2%	21.74

Table 1: Parameters of the best configurations

with 3 results from other papers [Douiri and Elberoussi, 2015; Abbasian and Mouhoub, 2011; Hindi et al., 2012]. Table 2 shows the results of the best configuration (i.e. configuration 10).

In Table 2, we have a *graph* column indicating the graph's name. Columns $|V|$ and $|E|$ indicate the number of vertices and edges in the graph respectively. Column $X(G)$ indicates the best-known number of colors for that graph. Column $M.K$ is the minimum number of colors found in 1 of the 10 executions. Column K is the average number of colors of all 10 executions. Column SD is the standard deviation of colors among all the execution. Column T is the average time in seconds taken by our algorithm. Columns $A1$, $A2$, and $A3$ are the time obtained by the other authors [Douiri and Elberoussi, 2015; Abbasian and Mouhoub, 2011; Hindi et al., 2012] respectively, which their algorithm could find the best-known color number to all instances listed in Table 1. Columns with

a dash mean that the author did not use that instance.

Graph	V	E	X(G)	M.K	K	SD	T	A1	A2	A3
anna	138	493	11	11	11	0	5.5223	11.63	2.43	0.058
david	87	406	11	11	11	0.3	2.6484	10.89	1.57	0.019
fpsol2.i.1	496	8691	65	139	144.7	11.94	80	82.03	77.42	22.656
games120	120	638	9	11	13.1	2.74	80	10.77	2.38	0.027
homer	561	1629	13	28	30.9	10.12	80	-	46.74	0.760
huck	74	301	11	11	11.7	1.30	40.1876	11.1	1.76	0.015
jean	80	254	10	12	12.8	1.66	80	9.92	1.31	0.015
miles250	128	387	8	11	14.1	2.97	80	4.39	1.88	0.076
miles1000	128	3216	42	42	42	2.33	1.1427	-	6.04	48.559
myciel3	11	20	4	4	4	0.49	0.0235	0.01	0.12	0.003
myciel4	23	81	5	5	5.1	0.60	8.2273	0.89	0.44	0.006
queen5.5	25	160	5	6	6.7	0.40	80	1.2	0.65	-
queen6.6	36	290	7	8	9.2	0.87	80	1.32	0.84	-
queen7.7	49	476	7	8	8.9	1.66	80	6.91	1.13	6.27
queen8.8	64	728	9	10	10.7	1.96	80	9.87	1.59	-

Table 2: Results of the best configuration (10)

We can say that it was better to use 100 individuals than 300 because by using 300 individuals we had similar results but the algorithm took longer. For bigger graphs such as fpsol2.i.1, configurations with a population composed of 300 individuals had worse results than those configurations that used 100 individuals.

In our algorithm, was seen that the major factor that made some solutions to be better than others was the population size. Using 100 individuals, made it possible for all the other algorithm's phases to have a more important role, not only depending on generated random values initially. The second more important factor was the crossover used, which was the crossover of type A that led to more accurate results. The mutation type that had slightly better results was the mutation of type B, although this mutation type in combination with all other parameters had a worse result, making the mutation of type A be the best parameter. The last factor was the mutation rate, which actually did not have major effects.

In comparison to the state-of-the-art, our approach did not use any particular technique. Was used a very small and higher mutation rate, not only a fixed value, and the number of the initial population varied as well since we did not use a range from 150 up to 250 individuals. About the crossover and mutation, most of them used one particular method, not having multiple functions and making a decision as to which one should be used, as was done in this paper. They also used CPUs like Intel Core i7 860 @ 2.8GHz where the algorithm was written using Java.

7. Conclusion

In graph coloring, we have to take into account the time the algorithm takes to find a good solution and minimize the number of colors it uses. In this process, we use lots of techniques, and we end up with a huge and more complicated algorithm, then, having more chances to make a mistake. It would be better if we could find these solutions in a reasonable time with a simple algorithm solution. In this paper, we presented a solution using a genetic algorithm in an attempt to

improve the solution's time and complexity for problems related to graph coloring, without using any additional and more complex techniques.

Although for most of the instances, the algorithm could not find the best solution, it got very close for small instances. From the analyzed results, our solution has the potential to find the best solutions in the beginning. The most important key of our solution was the initial population which can be seen in Figure 8, that did not take too many individuals to have a reasonable result. The crossover type A showed that can be a good starting point as well as mutation type B as shown in Figures 9 and 10. Because of this, the stop criterion showed that it was perfectly fine for small instances and could even be much less, and it is on the right way for larger instances. As we saw in Table 2, our algorithm was better than some authors and in other cases was better than all of the authors. All of these are related to those 15 particular instances used in our tests, We can not ensure that the best configuration in our tests would be the best configuration using other groups of instances. At the same time, these results can assist in selecting what would be the best starting point to test other instances.

We had some evidence that a better way to find solutions for bigger graphs is required. Relying only on random values to perform all the phases in a genetic algorithm was not very efficient, in order to obtain better results, something more sophisticated is required.

For future works, it would be a good start to try to figure out deeply what the common aspects among the graphs that made our algorithm perform better for some instances and worse for others and which part of our algorithm has more and/or less impact in each instance. Thereafter try to find some key points in the proposed algorithm to increase its performance that can make all the tested instances in this paper have a significant improvement, and test it using new instances as well.

References

- Abbasian, R. and Mouhoub, M. (2011). An efficient hierarchical parallel genetic algorithm for graph coloring problem. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, p. 521–528.
- Assi, M., Halawi, B., and Haraty, R. A. (2018). Genetic algorithm analysis using the graph coloring method for solving the university timetable problem. *Procedia Computer Science*, 126:899–906.
- Bondy, J. and Murty, U. (1976). *Graph theory with applications*. The Macmillan PressLtd.
- Chartrand, G. and Zhang, P. (2009). *Chromatic graph theory*. CRC Press.
- Douiri, S. M. and Elbernoussi, S. (2015). Solving the graph coloring problem via hybrid genetic algorithms. *Journal of King Saud University-Engineering Sciences*, 27(1):114–118.
- Edelkamp, S. and Schrödl, S. (2012). *Heuristic Search, Theory and Applications*. Elsevier.
- Ertel, W. (2017). *Introduction to Artificial Intelligence, 2nd edition*. Springer.
- Fleurent, C. and Ferland, J. A. (1996). Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63(3):437–461.
- Gomes, E. P. (2023). Web page. http://github.com/Everaldo-Gomes/Graph_coloring.

- Hindi, M. M., Yampolskiy, and V, R. (2012). Genetic algorithm applied to the graph coloring problem. In *Proc. 23rd Midwest Artificial Intelligence and Cognitive Science Conf*, p. 61–66.
- Marappan, R. and Gopalakrishnan, S. (2013). A new genetic algorithm for graph coloring. In *2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation*, p. 49–54.
- Mumford, C. L. (2006). New order-based crossovers for the graph coloring problem. In *Parallel problem solving from nature-PPSN IX*, p. 880–889. Springer.
- Rahman, M. S. (2017). *Basic graph theory*. Springer.
- Salari, E. and Eshghi, K. (2005). An aco algorithm for graph coloring problem. In *2005 ICSC Congress on computational intelligence methods and applications*, p. 5–pp. IEEE.
- Sethumadhavan, G. and Marappan, R. (2013). A genetic algorithm for graph coloring using single parent conflict gene crossover and mutation with conflict gene removal procedure. In *2013 IEEE International Conference on Computational Intelligence and Computing Research*, p. 1–6. IEEE.
- Sivanandam, S. N. and Deepa, S. N. (2008). *Introduction to Genetic Algorithm*. Springer.
- West, D. B. (2001). *Introduction to Graph Theory, 2nd edition*. Pearson Education.