Graph Coloring - SOTA and Benchmark Instances

Apetrii Radu, Baranceanu Vlad, Brezuleanu Alex

IAO 1

1 Graph Coloring Problem

The problem of graph coloring refers to assigning colors/numbers to vertices in such a way that no two adjacent vertices have the same color, while minimizing the total number of colors used. This dates back to the *19-th* century and was later classified as a *NP-complete* problem.

There are multiple techniques that have been used throughout the time, many of them providing really good results. In the next chapter, we plan on getting into details on some of these methods.

2 SOTA Approaches

Considering that the Graph Coloring problem is such a well-known one, there have been numerous attempts at solving it, each method consisting of specific ideas, mechanisms, resources. Knowing this, we have decided to tackle only the methods that can be divided/grouped into three main categories: *heuristic* approaches, *meta-heuristic* approaches, *hybrid* approaches. In terms of meaning, they consist of the following:

- *heuristic*: a practical rule or strategy used to make decisions during the coloring process without guaranteeing optimality. It is mainly based on intuitive or empirical principles rather that rigorous mathematical proof.
- *meta-heuristic*: a higher-level strategy used to efficiently explore the solution space, aiming to find near-optimal solutions without guaranteeing optimality.
- *hybrid*: a strategy that combines multiple techniques, often incorporating both heuristic and meta-heuristic methods, to explore their strengths and overcome their limitations.

2.1 Heuristic approaches

2.1.1 DSatur

A greedy algorithm mainly used for its simplicity and effectiveness in producing reasonably good solutions for the graph coloring problem. It focuses on computing saturation degrees for nodes, which translates into computing the number of different colors used by their neighbors. In short terms, this algorithm runs until all the nodes have been colored, each step consisting of selecting the node with the highest saturation degree that has not been colored yet, color it with the smallest possible color that is not used by its neighbours, and update the saturation degree of the neighbours. [1]

DSatur has been observed to perform well on a wide range of graph instances, including both sparse and dense graphs. It tends to excel when there are vertices with high degrees or high degrees of connectivity, as it prioritizes coloring these vertices early in the process, potentially reducing the overall number of colors needed.

However, DSatur may encounter difficulties with certain types of graphs, such as highly irregular or structured graphs, where the distribution of vertex degrees is uneven or where there are many vertices with similar saturation degrees. In such cases, DSatur may not be able to exploit the available structure effectively, leading to sub-optimal solutions.

2.1.2 Recursive largest first (RLF)

A strategy that intends to build upon DSatur by adding one more phase at each step. Basically, after coloring the selected node, there is a recursive refinement step added into the process. This recursive step considers for each color class, the vertices in non-decreasing order of degree, and then it tries to recolor the current vertex with the color that minimizes conflicts. If conflicts cannot be reduced, revert the coloring back to the previous state. [2]

RLF produces good solutions on tests that have a relatively small number of colors. By iteratively refining the coloring through a recursive process and considering the degree of vertices in the refinement step, RLF aims to reduce the number of conflicts and improve the overall quality of the coloring.

On the other hand, RLF may encounter difficulties with certain types of graphs, such as highly regular or structured graphs, where the recursive refinement step may not yield substantial improvements.

2.2 Meta-heuristic approaches

2.2.1 Genetic Algorithms

A strategy used due to its ability to efficiently explore the solution space and find good-quality solutions. It relies on three key processes during each generation: [3]

- *Selection*: Select individuals from the population for reproduction based on their fitness value. Individuals with higher fitness values are more likely to be selected.
- *Mutation*: Process applied in order to introduce small changes and maintain diversity in the population. This might involve randomly changing the color of a vertex, swapping colors between vertices, or other modifications.
- *Crossover*: Method used for generating offsprings. This involves exchanging genetic material (colors assigned to vertices) between parent solutions to produce new candidate solutions.

Genetic algorithms are capable of finding solutions with relatively few colors compared to the greedy algorithms, particularly for large and complex problem instances. Not only that, but this approach is well-suited for solving large-scale instances, as it can effectively explore the solution space.

However, genetic algorithms can be computationally expensive. Also, if the instance presents characteristics of slow converging, the quality of the solutions obtained by GAs may not always be optimal.

2.2.2 Ant Colony Optimization

An algorithm designed to simulate the behavior of ants, ACO builds solutions iteratively by assigning colors to vertices based on probabilistic decision rules influenced by pheromone trails. In terms of the process, this approach consists of two key components: [4]

- *Ant*: An ant represents a solution that is constructed in an iterative mode by choosing colors in a probabilistic fashion. An ant begins its journey from a start node and goes along the graph until all the nodes have been colored.
- *Colony*: After all the ants of the current generation have finished their movement phase, the algorithm updates the pheromone levels from the edges based on the quality of the solutions found.

Ant Colony Optimization is able to find good solutions that use a few number of colors. Also, it is generally scalable and can handle large-scale instances of the graph coloring problem. However, this method may require a large number of iterations to converge to those good solutions.

2.3 Hybrid approaches

2.3.1 Genetic Algorithm with Simulated Annealing

Method that combines the strength of Genetic Algorithms and Simulated Annealing. It involves a phase for each algorithm, and can be described as such: [5]

- *Genetic Algorithm Phase*: Keep the ideas of this approach, meaning that there will be a Selection (based on fitness value), a Mutation, and a Crossover phase. This will keep the population evolving.
- *Simulated Annealing Phase*: After the termination condition has been met for the first phase, the Simulated Annealing kicks in. This way, it explores the solution space and further refines the candidate solutions obtained from the genetic algorithm part. It also applies moves that may worsen the solution with a certain probability determined by the current temperature. This allows the algorithm to escape local optima and explore new regions of the solution space.

2.3.2 Ant Colony Optimization with Tabu Search

As in the previous method, this approach consists of two phases: running the Ant Colony algorithm, and then optimizing the results using a Tabu Search: [6]

- *Ant Colony*: This phase is exactly as the one described in the meta-heuristic category, meaning that throughout the generations, each ant forms a solution, and then the algorithm updates the pheromone levels based on the goodness of the solution found.
- *Tabu Search*: This phase happens after the termination criteria is met for the first one. The method then performs local search operations such as swaps or perturbations to explore the neighborhood of the current solution, while also maintaining a tabu list to prevent revisiting previously explored solutions and encourage diversification in the search process.

3 Benchmark instances

In terms of instances that have been used over the years in order to test the quality of a solution, a few popular ones can be found here. [7] We will remark some of them that have unique features:

- *fpsol2.i.1* - Graph that is based on register allocation for variables in real code and contains 496 vertices, 11654 edges. The optimum solution for this instance is 65 colors.
- *inithx.i.1* - Graph which serves the same purpose as the instance before, containing 864 vertices and 18707 edges, and having an optimal solution of 54 colors.
- *latin_square_10* - Graph instance with 900 vertices and 307350 edges, constructed by following the pattern of the latin square.
- *le450_5c* - Leighton graph with 450 vertices, 9803 edges, and an optimal solution of 5 colors.

- *mulsol.i.5* - Graph with 186 vertices, 3973 edges, and an optimal solution of 31 colors.
- *school1_nsh* - Class scheduling graph with 352 vertices, 14612 edges.
- *homer* - Book graph containing 561 vertices, 1629 edges and an optimal solution of 13 colors.
- *miles750* - Miles graph containing 128 vertices, 2113 edges and an optimal solution of 31 colors.
- *queen16_16* - Queen graph containing 256 vertices and 12640 edges.
- *myciel7* - Graph based on the *Mycielski* transformation that contains 191 vertices, 2360 edges and an optimal solution of 8 colors.

4 Our approaches

We have opted for two methods in our task of solving the Graph Coloring problem: a Particle Swarm Optimization approach, and an Ant Colony Optimization one.

4.1 Particle Swarm Optimization approach

The PSO algorithm is a metaheuristic that is used as an optimization method that mimics the natural movement of a flock of birds. The particles move in the search space according to a certain speed so that they arrive at the best possible solution. The fitness function calculates the number of nodes that have the same color as their neighbors.

The main steps of the algorithm are:

- Generation of the opulation
- Fitness value calculation
- Speed update
  - updating the best fitness value if the current fitness is better than the previously saved one
  - calculating the speed by the formula:

$$V_i^{t+1} = w \cdot V_i^t + c_1 \cdot r_1 \cdot (P_{best}(i)^t - P_i^t) + c_2 \cdot r_2 \cdot (P_{bestglobal}^t - P_i^t)$$

    - $w \cdot V_i^t$ - the particle tends to maintain its trajectory (inertia)
    - $c_1 \cdot r_1 \cdot (P_{best}(i)^t - P_i^t)$ - the particle is attracted to the best position reached up to the current iteration
    - $c_2 \cdot r_2 \cdot (P_{bestglobal}^t - P_i^t)$ - the particle is attracted to the best solution found globally up to the current iteration

– updating the particle's position

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

* w - inertia factor
* c_1 - the cognitive factor (the tendency to choose a previous good position)
* c_2 - the social factor (the tendency to choose the best solution)
* r_1, r_2 - random numbers from the interval [0,1]

Analyzing by the number of nodes, we made a graph (Figure 1) which shows that the PSO algorithm deviates more easily than the ACO and in fewer cases.

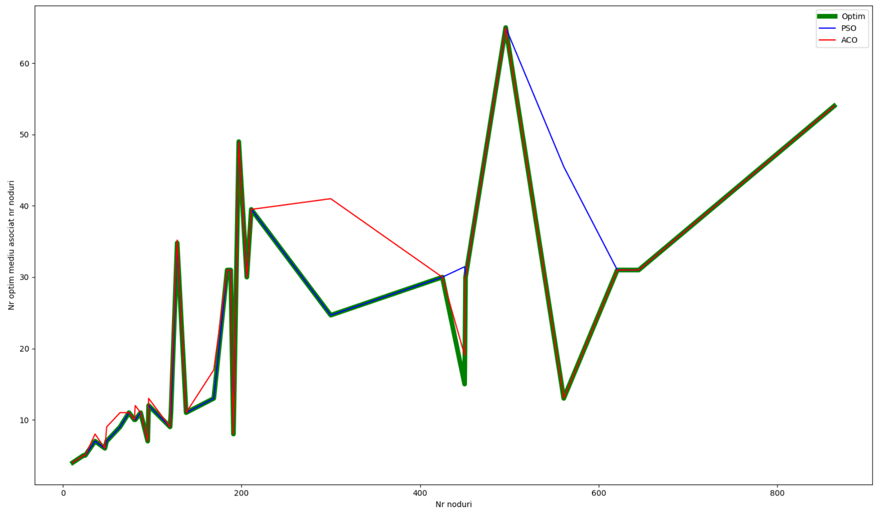

Figure 1: Visualization of the average results obtained by the PSO and ACO algorithms, depending on the number of nodes

4.2 Acceleration Based Particle Swarm Optimization (APSO)

If $c_1 \ll c_2$, particles may converge prematurely as they are more strongly attracted to the global best position than to their personal best positions. Conversely, if $c_2 \ll c_1$, particles may converge slowly or not at all because they are more strongly attracted to their personal best positions than to the global best. Since Particle Swarm Optimization (PSO) relies on a balance between personal and social knowledge of the search space, the coefficients c_1 and c_2 are typically chosen to be nearly equal.

Excessively large values of c_1 and c_2 cause particle velocities to accelerate too quickly, leading to swarm divergence. Conversely, excessively small values of

c_1 and c_2 result in slow convergence as particles move too slowly. Proper control over global exploration and local exploitation is crucial for efficiently finding the optimal solution. High diversity is generally needed early in the search to cover the full range of the search space, while fine-tuning is essential in the latter stages to converge accurately on the global optimum. Thus, maintaining a balance between these two components is vital for accurate and efficient optimization.

Particles are most effective when c_1 and c_2 are adaptive, facilitating both exploration and exploitation of the search area. To enhance PSO performance, the Acceleration-based PSO (APSO) was introduced.

APSO is designed to address the premature convergence issue of standard PSO. In APSO, the acceleration coefficients are chosen based on the fitness value, which improves result accuracy. The selection of these acceleration coefficients in the APSO algorithm is detailed as follows:

$$nc_1 = c_1 \times (1 - \lambda)$$
$$nc_2 = c_2 \times (1 + \lambda)$$

$$\lambda = \frac{\chi\,(1 + \varphi\,(f_{\max} - f_{\min})^{\omega} - (f_{avg}{}^{\omega}))}{\delta\,(f_{\max} - f_{\min})^{\omega} - f_{avg}^{\omega}}$$

$$\delta = \left(\frac{f_{\max} - f_{\min}}{f_{avg}} \right)^{\omega}$$

Where,

χ - Alteration probability

ω, φ - Coefficient factors

$F_{\max}, F_{\min}, F_{avg}$ - Maximum, minimum and average fitness of the particles

4.3 Ant Colony Optimization approach

Our approach of Ant Colony can be divided into two major components: the ant logic, meaning what processes an individual ant follows, and the main body, denoting the global goal of the algorithm. [9]

Note: For a more detailed view of the ACO results, please check the *aco.xlsx* file.

4.4 Ant Movement

When a new ant is created, it first chooses randomly a vertex to start in. Also, it assigns the first color that is available to the current vertex, marking the fact that it has been visited (so it knows to not return later on).

After the initialization process is done, an ant will do the following things until all vertices are visited:

- It chooses a candidate node to visit. In order to select such a vertex, it takes into consideration the values of α, β, the pheromone level that lies on the path to the possibly next node, and the number of colors in use by the neighbours of the potentially next node, in order to create a total score. Then, only the candidate vertices that have achieved the greatest score out of all vertices are considered for the next part of selection. Last, a random vertex is chosen as the candidate node out of the ones that have made it this far.

- Next, the ant colors the node that it reached and marks the vertex as being visited. The color used is the first available one, meaning that there are no conflicts with the neighbours colors.

- After the two steps from above, the ant calculates the value of the current solution - how many different colors were used until this point -. In order to do this, the ant keeps track of the colors assigned to nodes.

4.5 Main Body

The main body handles the events that are happening during each iteration of the program, followed by the final computation of the result. In a generation there are various things done, such as:

- Generating a starting colony. The size of the colony depends on the value of the *colony_size* hyper-parameter.

- Letting each ant do its job and finish visiting all the nodes in the graph. The detailed process of movement is explained in the previous sub-section.

- Applying a factor of decaying over the pheromone levels. This means that each edge will suffer a decreasing in value for its pheromone level. This is determined by the value of the *pheromone_decay* hyper-parameter.
- Updating the overall best solution and finding the ant of the current generation that provided the best results. This ant - the elitist ant - is the only ant that is allowed to leave pheromone trails behind in this generation of ants.
- Gathering of all results so the final statistics can be computed.

4.6 Pheromone Level

On our original design of ACO, the elitist ant would leave a pheromone trail. On each run in which the ant is allowed to leave pheromone, the value increased by 1.

On a variation of ACO, we have decided to follow the ideas of Ehsan Salari and Kourosh Eshghi, in which the elitist ant would not provide 1 to the overall value of pheromone level. Instead, the value would increase by the sum of squares of the number of vertices that use a color, for each color used in the overall schema. This way we have defined ACO-P, the variation of our ACO design, but with the updated value of the pheromone trail. This provided better results on some extent. [8]

The update formula is:

$$\tau_{ij} = (1-\rho) * \tau_{ij} + \sum_{i=1}^{q} (C_i)^2, \forall i, j : v_i, v_j \in C_k, k : 1, 2, ..., q$$

4.7 Hyper-parameters

In our approach of Ant Colony Optimization, we have made use of the following hyper-parameters:

- *colony size* - The number of ants that will form the current generation.
- *generations* - The number of iterations of the algorithm the program will do.
- *pheromone decay* - The value of decaying applied to the pheromone levels.
- α - First control parameter.
- β - Second control parameter.

In terms of pairs of values used for the hyper-parameters, some common values are:

- *colony size* = 50, *generations* = 50 - for smaller instances.
- *colony size* = 30, *generations* = 30 - for medium instances.

- *colony size* = 15, *generations* = 15 - for larger instances.

As for the other parameters, possible pairs of values may be:

- *pheromone decay* = 0.77, $\alpha = 2$, $\beta = 5$,
- *pheromone decay* = 0.95, $\alpha = 3$, $\beta = 1$,
- *pheromone decay* = 1, $\alpha = 1$, $\beta = 3$,
- *pheromone decay* = 0.2, $\alpha = 2$, $\beta = 5$.

4.8 Time and Memory

Depending on the type of the instance, the program would yield different execution times:

- Time of 2-3 seconds for smaller instances (i.e. anna, jean, huck)
- Time of 20-80 seconds for medium instances (i.e. myciel7, zeroin.i.1, queen8_8)
- Time of 250-400 seconds for larger instances, excluding latin_square_10 (i.e. le450_25d, queen16_16)

As for the memory consumed by the program, every run required between 60 and 350 KB, meaning that ACO does not need a lot of memory to operate.

5 Results

A side-by-side comparison of the results we have obtained can be found below.

Table 1: Experimental Results

Instance	**Vertices**	**Edges**	**Expected Result**	**ACO Result**	**ACO-P Result**	**PSO Result**	**APSO Result**	**Total Steps**
fpsol2.i.1	496	11654	65	65	65	65	65	50/23
fpsol2.i.2	451	8691	30	30	30	30	30	46/19
fpsol2.i.3	425	8688	30	30	30	30	30	63/32
inithx.i.1	864	18707	54	54	54	54	54	49/25
inithx.i.2	645	13979	31	31	31	31	31	72/37
inithx.i.3	621	13969	31	31	31	31	31	69/40
latin_square_10	900	307350	97	144	-	-	122	100/58
le450_15a	450	8168	15	16	15	24	20	60/27
le450_15b	450	8169	15	16	15	25	21	59/23
le450_15c	450	16680	15	23	22	23	23	72/37
le450_15d	450	16750	15	24	24	26	24	35/16
le450_25a	450	8260	25	25	25	27	25	45/22
le450_25b	450	8263	25	25	25	25	25	57/32
le450_25c	450	17343	25	28	27	25	25	60/38
le450_25d	450	17425	25	28	27	26	25	57/29
le450_5a	450	5714	5	9	8	21	15	73/45
le450_5b	450	5734	5	9	8	23	14	28/18
le450_5c	450	9803	5	6	5	19	12	47/32
le450_5d	450	9757	5	6	5	21	14	78/62
mulsol.i.1	197	3925	49	49	49	49	49	68/34
mulsol.i.2	188	3885	31	31	31	31	31	26/15
mulsol.i.3	184	3916	31	31	31	31	31	33/12
mulsol.i.4	185	3946	31	31	31	31	31	29/17
mulsol.i.5	186	3973	31	31	31	31	31	31/13
school1	385	19095	14	14	14	15	14	27/18
school1_nsh	352	14612	14	14	14	14	14	45/32
zeroin.i.1	211	4100	49	49	49	49	49	62/24
zeroin.i.2	211	3541	30	30	30	30	30	56/42
zeroin.i.3	206	3540	30	30	30	30	30	68/42
anna	138	986	11	11	11	11	11	40/21
david	87	812	11	11	11	11	11	45/23
homer	561	3258	13	13	13	40	24	19/9
huck	74	602	11	11	11	11	11	31/18
jean	80	508	10	10	10	10	10	18/9
games120	120	1276	9	9	9	9	9	14/7
miles1000	128	6432	42	42	42	42	42	59/32
miles1500	128	10396	73	73	73	73	73	60/36
miles250	128	774	8	8	8	8	8	39/25
miles500	128	2340	20	20	20	20	20	42/22
miles750	128	4226	31	31	31	31	31	61/34
queen10_10	100	2940	11	13	12	11	11	24/10
queen11_11	121	3960	11	14	13	11	11	35/17
queen12_12	144	5192	12	15	13	12	12	29/14
queen13_13	169	6656	13	17	15	13	13	42/18
queen14_14	196	8372	14	18	17	14	14	38/24
queen15_15	225	10360	15	19	18	16	15	46/28
queen16_16	256	12640	16	20	18	18	16	42/19
queen5_5	25	320	5	5	5	5	5	38/23
queen6_6	36	580	7	8	7	7	7	17/10
queen7_7	49	952	7	9	7	7	7	23/9
queen8_12	96	2736	12	13	12	12	12	28/11
queen8_8	64	1456	9	10	9	9	9	34/20
queen9_9	81	2112	10	11	10	10	10	42/22
myciel3	11	20	4	4	4	4	4	23/11
myciel4	23	71	5	5	5	5	5	27/18
myciel5	47	236	6	6	6	6	6	25/14
myciel6	95	755	7	7	7	7	7	22/10
myciel7	191	2360	7	8	8	8	8	32/17

5.1 Remarks

Based on the table from above, we can extract the following observations:

- There are a total of 58 instances that were tested.
- PSO provides the optimal result in 43 of those instances.
- ACO provides the optimal result in 35 of those instances.

- PSO provides better results than ACO on 14 instances.
- ACO provides better results than PSO on 10 instances.
- On the other 33 instances, they provide the same result.
- If none of the methods are able to get the optimal result, ACO provides a much closer answer to the optimal one than PSO.
- PSO provides much better results on the *queen* instances than ACO.

6 Conclusion and future directions

Throughout the article, we have presented the main Graph Coloring problem, some state-of-the-art solutions to it, and our approaches. The latter one consisted of a Particle Swarm Optimization method and an Ant Colony Optimization one. Both provided decent results, the first method edging the second one on a few instances.

In terms of future directions, we were able to conclude the following:

- Perform hyper-parameter tuning and see if the results get better, especially on the instances for which out methods provided really close answers to the expected one.
- Check other articles related to our approaches and extract specific information that may aid us in getting better results. This means performing some variations to our already existing methods to see if we can boost the correctness and performance of the algorithms.

References

[1] https://www.geeksforgeeks.org/dsatur-algorithm-for-graph-coloring/

[2] https://www.gerad.ca/~alainh/RLFPaper.pdf

[3] https://res.ijsrset.com/page.php?param=IJSRSET1841049

[4] https://www.sciencedirect.com/science/article/pii/S0166218X07001023

[5] Comparative Performance of Modified Simulated Annealing with Simple Simulated Annealing for Graph C

[6] https://ieeexplore.ieee.org/document/1631360

[7] https://mat.tepper.cmu.edu/COLOR/instances.html

[8] https://www.m-hikari.com/ijcms-password2008/5-8-2008/eshghiIJCMS5-8-2008.pdf

[9] http://ijesc.org/upload/c6c0941d337a1b5b634062a54bb33d5c.Ant%20Colony%20System%20for%20Graph%20Coloring%20Problem.pdf