Graph Coloring Optimization using Genetic Algorithm and Simulated Annealing

Florea Robert-Andrei

12 January 2025

1 Abstract

This paper presents a hybrid approach combining genetic algorithms and simulated annealing to solve the graph-coloring problem. The algorithm employs multiple parent selection and mutation techniques that adapt according to the fitness of the current solution. This flexibility accelerates convergence toward the global optimum compared to using a single parent selection or mutation method. The approach is tested on a custom set of graph data, where a population of potential solutions is evolved through crossover, mutation, and simulated annealing. The algorithm iterates through several generations, adjusting the number of colors used for the coloring problem until a valid solution is found or a termination criterion is met. The results indicate that the hybrid method is effective in minimizing the number of colors required for coloring the graph while also optimizing computational efficiency. Additionally, the algorithm demonstrates improvements over traditional methods, achieving better performance on certain graph instances by finding solutions with fewer colors than standard graph coloring algorithms.

2 Introduction

The graph coloring problem is a fundamental issue in graph theory with applications in various fields such as resource allocation, scheduling, and communication networks. The objective of the problem is to assign colors to the nodes of a graph in such a way that no two adjacent nodes share the same color, using the minimum number of colors. This minimum number is known as the chromatic number of the graph.

The graph coloring problem is NP-complete, which means there is no known algorithm that can solve it efficiently for all cases. As a result, approximation techniques and heuristic methods are commonly used to find good solutions in a reasonable amount of time. In this context, genetic algorithms and optimization techniques such as simulated annealing are employed to tackle the problem by exploring large solution spaces more effectively.

In this paper, we propose a hybrid approach that combines genetic algorithms with simulated annealing to solve the graph coloring problem. Our algorithm evolves a population of possible solutions, applying parent selection, crossover, and mutation methods to converge toward an optimal solution.

Additionally, the algorithm incorporates simulated annealing to refine the solutions and avoid getting stuck in local optima.

The algorithm is tested using the standard DIMACS benchmark suite, which provides a set of graph instances with known chromatic numbers. The approach adapts the number of colors based on the graph structure, iterating through generations to find the optimal coloring. Results show that the hybrid method is effective in minimizing the number of colors required for graph coloring, outperforming traditional methods in terms of solution quality and computational efficiency.

Given a number of vertices, which form a connected graph, the objective is to color each vertex such that if two vertices are connected in the graph they will be colored with different colors. Moreover, the number of different colors that can be used to color the vertices is limited and a secondary objective is to find the minimum number of different colors needed to color a certain graph without violating the adjacency constraint.

If $k = 1, 2, 3...$ and $P(G, k)$ is the number of possible solutions for coloring the graph G With k colors, then

$$\chi(G) = min(k : P(G, k) > 0) \tag{1}$$

3 Methods

This code implements two popular optimization techniques—Genetic Algorithms (GA) and Simulated Annealing (SA)—to solve the graph coloring problem. The goal of graph coloring is to assign colors to the nodes of a graph such that no two adjacent nodes share the same color, while minimizing the number of colors used. The code provides an exploration of both methods in finding near-optimal solutions for graph coloring.

Genetic Algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures as as to preserve critical information. Genetic algorithms are often viewed as function optimizer, although the range of problems to which genetic algorithms have been applied are quite broad.[1]

Simulated Annealing (SA) is a probabilistic optimization technique inspired by the physical process of slowly cooling a material, which allows it to settle into a state of minimum (or near minimum) energy. In the context of optimization problems, SA aims to minimize a cost or fitness function for a solution, similar to how a metallic structure stabilizes as its temperature decreases.[2]

In this implementation, the chromosome is represented by a vector of integers, where each element corresponds to a color assigned to a vertex in the graph. Each value in the vector is an integer representing the color assigned to the vertex, ranging from 1 to the maximum number of colors, which is determined dynamically based on the graph's structure.

The graph is represented by an adjacency matrix, where each element $graph[i][j] = 1$ indicates an edge between vertex i and vertex j, and $graph[i][j] = 0$ indicates no edge. The fitness of a chromosome is determined by how many "bad edges" (edges connecting two vertices with the same color) are present in the solution. The objective is to minimize this fitness score.

The population size is set to 200, which is relatively large and chosen to ensure diversity in the candidate solutions. The population size is constant throughout the execution, and a generational approach is used, meaning that new individuals are generated by applying selection, crossover, and mutation to the current population.

At each iteration, the tournament selection method is used to select the fittest individuals for mating. This selection mechanism is often effective in maintaining genetic diversity while improving the fitness of the population. After selection, crossover and mutation operations are applied to generate new offspring, which are then added to the population.

The algorithm runs for a maximum of 1000 generations, but it may terminate earlier if a solution with 0 bad edges is found, indicating a valid coloring of the graph. If no solution is found after 1000 generations, the algorithm attempts to further improve the population using simulated annealing.

Selection and Mutation Methods

1. **Tournament Selection**: In this method, two random individuals are selected from the population, and the one with the lower fitness score (i.e., fewer bad edges) is chosen to be part of the next generation. This process is repeated until a new population is formed.
2. **Two-Point Crossover**: After selecting two parents, the two-point crossover method is applied. This technique involves choosing two random crossover points in the chromosomes and swapping the segments between the two parents. This produces two offspring with mixed traits from both parents.
3. **Mutation**: Mutation is applied with a small probability (0.007) to alter the chromosome's color assignments. If a mutation occurs, the algorithm checks for conflicts (bad edges) and reassigns the colors of the conflicting vertices. This mutation introduces randomness into the search, preventing the algorithm from prematurely converging to a suboptimal solution.

The Simulated Annealing (SA) technique is integrated into the GA to further refine the candidate solutions. After each crossover and mutation, Simulated Annealing is applied to each offspring. This process allows the algorithm to escape local optima by accepting worse solutions with a certain probability, based on the temperature parameter. The probability of accepting a worse solution decreases as the temperature T reduces over time, following the cooling schedule $T = T \times \alpha$ where α is the cooling rate (set to 0.95 in this case). The simulated annealing process helps explore the solution space more thoroughly and avoid getting stuck in local minima.

4 Data

Data used to test our approach are derived from the DIMACS benchmarking graph collection. DIMACS is the Center for Discrete Mathematics and Theoretical Computer Science. It is part of Rutgers University (The State University of New Jersey Rutgers, et al. 2011). The data are frequently used in challenges involving constraint satisfaction problems. The files used have a .col extension. Each file contains a header with the number of vertices (p) and the number of edges (e):

p edge 11 20

A number of lines follow the header with each line denoting the connected edges and their vertex indices:

e 1 5

10 files were chosen from the DIMACS collection. The graphs the files represent vary in vertex count, edge count and overall complexity. The vertex count ranges between 11 and 87 and the edge count ranges from 20 to 728. The rationale behind the selection of these graphs other than the wide range of variation is that there is a known chromatic number for each of them, or at least a good approximation.

The following files were used in this approach: (myciel3.col, myciel4.col, myciel5.col, queen5_5.col, queen6_6.col, queen7_ 7.col, queen8_8.col, huck.col, jean.col, david.col).

5 Results

Table displays the following statistics for each file:

1. The number of vertices $|V|$
2. The number of edges $|E|$
3. The expected chromatic number $\chi(G)$
4. The minimum number of colors used by this algorithm k_{min}

File	—V—	—E—	Expected $\chi(G)$	k_{min}
myciel3.col	11	20	4	4
myciel4.col	23	71	5	5
queen5_5.col	23	160	5	5
queen6_6.col	25	290	7	**8**
myciel5.col	36	236	6	6
queen7_7.col	49	476	7	**8**
queen8_8.col	64	728	9	**11**
huck.col	74	301	11	11
jean.col	80	254	10	10
david.col	87	406	11	11

The following graphs plot the relationship between the fitness and the generation for a sample set of the files used:

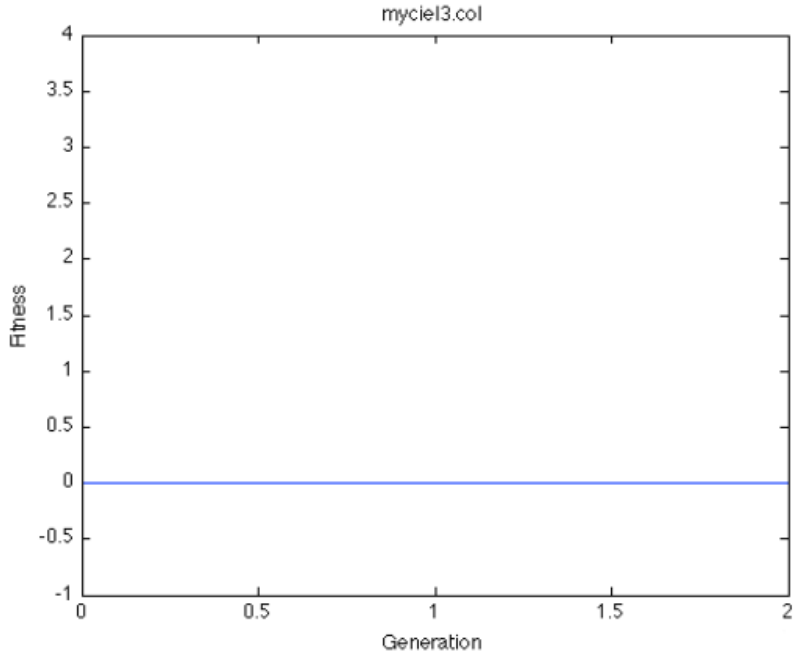

Figure 1: Fitness score over the number of generations for myciel3.col

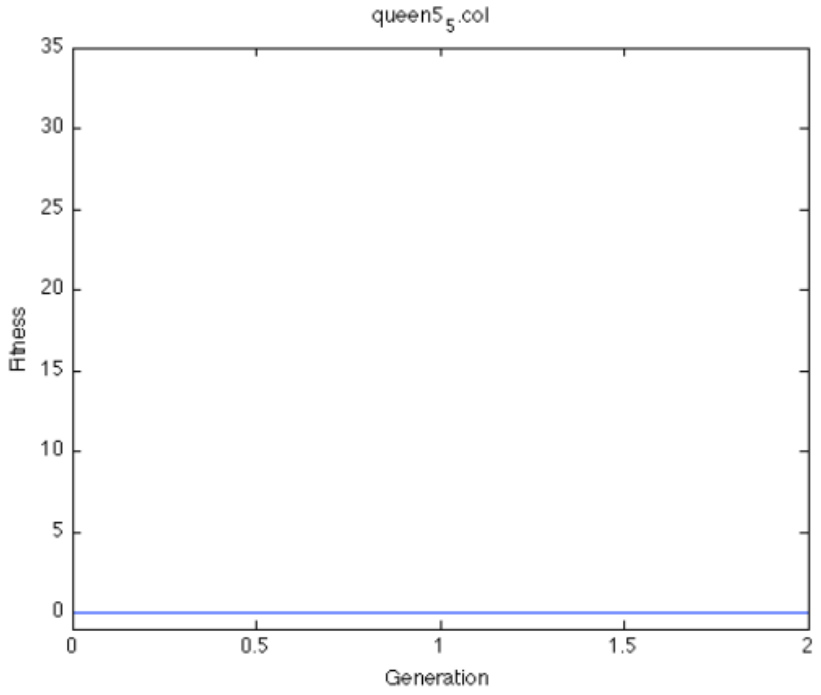

Figure 2: Fitness score over the number of generations for queen5_5.col

Figures 1 and 2 are not very interesting since the solutions are found after a few generations.

The next plot, however, is of particular interest since it clearly represents the erratic behavior of the fitness score between the initial rapid drop until a solution is ultimately found. In standard genetic algorithms the fitness score continues to increase or decrease (depending of the definition of better fitness) until the end of the run. This is not the case here.

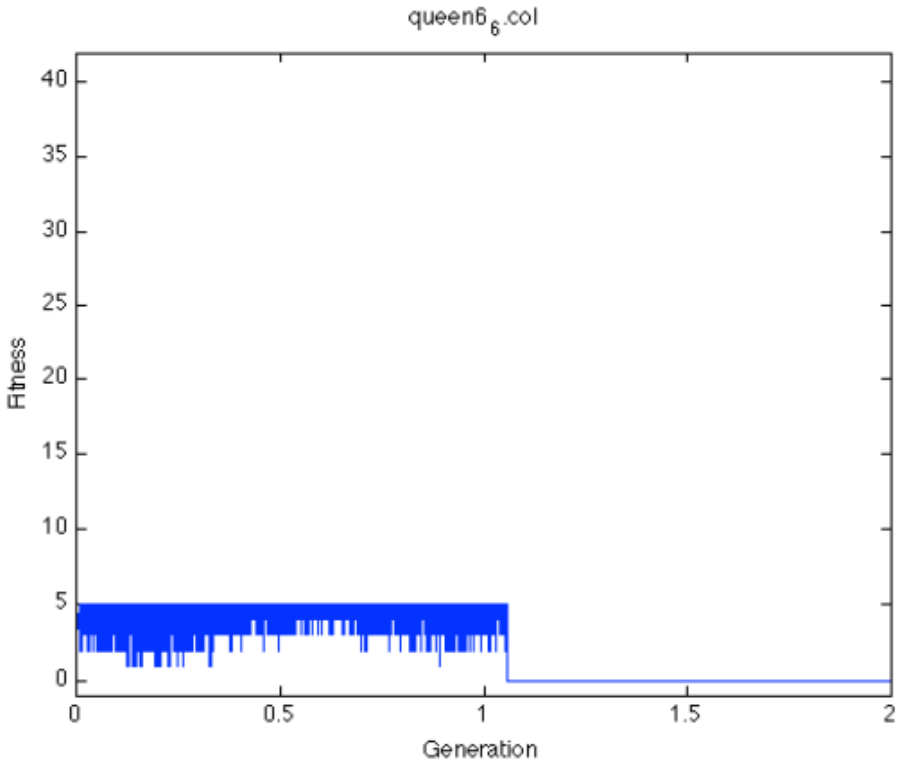

Figure 3: Fitness score over the number of generations for queen6.col

6 Conclusion

The combination of Genetic Algorithm and Simulated Annealing creates a powerful hybrid approach for solving the Graph Coloring Problem. The algorithm balances exploration and exploitation by using random initialization, tournament selection, crossover, and mutation to generate diverse offspring, while also using simulated annealing to refine solutions and escape local minima. By iterating over several generations and adjusting the number of colors dynamically, the algorithm efficiently finds a solution to the graph coloring problem, even for more complex graphs.

The use of dynamic temperature reduction and fitness-based selection further improves convergence, making the algorithm capable of finding high-quality solutions within a reasonable number of generations. Additionally, the statistical analysis of the results helps in evaluating the overall effectiveness and reliability of the algorithm in solving the problem.

References

[1] Hindi, Musa M., and Roman V. Yampolskiy. "Genetic algorithm applied to the graph coloring problem." Proc. 23rd Midwest Artificial Intelligence and Cognitive Science Conf. 2012. https://d1wqtxts1xzle7.cloudfront.net/81748178/download.pdf?1646474522=&response-content-disposition=inline%3B+filename%3DLinguistic_Profiling_and_Behavioral_Drif.pdf&Expires=1736694297&Signature=Xs5kvvHi2gp-7KHJLs~QCpAJft8Lt6Oqz3gwh7XGniaU3PV-LOar-Ek6J207Pd9SPFSKAQwoPQTp3saIqIk5SEnOZbrpfWLUJ-_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA#page=68.

[2] Marappan, Raja, and Gopalakrishnan Sethumadhavan. "A new genetic algorithm for graph coloring." 2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation. IEEE, 2013. https://ieeexplore.ieee.org/abstract/document/6663163.

[3] Assi, Maram, Bahia Halawi, and Ramzi A. Haraty. "Genetic algorithm analysis using the graph coloring method for solving the university timetable problem." Procedia Computer Science 126 (2018): 899-906. https://www.sciencedirect.com/science/article/pii/S1877050918313024.

[4] Pal, Anindya Jyoti, et al. "Comparative performance of modified simulated annealing with simple simulated annealing for graph coloring problem." Procedia Computer Science 9 (2012): 321-327.https://www.sciencedirect.com/science/article/pii/S187705091200155X.

[5] Titiloye, Olawale, and Alan Crispin. "Quantum annealing of the graph coloring problem." Discrete Optimization 8.2 (2011): 376-384. https://www.proquest.com/openview/f94764e57db57068481bad067e8bbc9b/1?pq-origsite=gscholar&cbl=18750&diss=y.

[6] Graph Coloring Instances DIMACS https://mat.tepper.cmu.edu/COLOR/instances.html.

[7] Matula, David W., George Marble, and Joel D. Isaacson. "Graph coloring algorithms." Graph theory and computing. Academic Press, 1972. 109-122.https://www.sciencedirect.com/science/article/abs/pii/B9781483231877500155.

[8] Barenboim, Leonid, and Michael Elkin. "Distributed graph coloring: Fundamentals and recent developments." (2013).https://books.google.ro/books?hl=en&lr=&id=ObRdAQAAQBAJ&oi=fnd&pg=PR13&dq=Graph+coloring+&ots=jRzz51mJEX&sig=vxN2lL9gcEgm6ftXl3UuljKe31M&redir_esc=y#v=onepage&q=Graph%20coloring&f=false.