Solving the Graph Coloring Problem (GCP) using a Genetic Algorithm (GA) and a Simulated Annealer (SA)

Albert Alexandru

January 2024

Abstract

The Graph k-Colorability Problem (GCP) is an NP-hard challenge focused on determining the minimum number of colors (k) required to paint the vertices of a graph, ensuring connected vertices have distinct colors. This problem is crucial in resource allocation, scheduling, and network design.

This study implements a genetic algorithm (GA) and a simulated annealer (SA) to tackle the Graph k-Colorability Problem (GCP). The primary objective is to assess their effectiveness, specifically in determining the chromatic number for graphs of varying complexity. The results reveal that, while successful in finding the chromatic number for simple and medium-sized graphs, both algorithms falter when confronted with more challenging instances. Notably, their performance falls short compared to more advanced genetic algorithms and alternative methods documented in the literature.

1 Introduction

The graph coloring problem stands out as a widely recognized challenge in graph theory. In its decision form, the task involves a graph and a natural number $k \in N$, with the objective of determining if k colors suffice for a proper coloring. This decision problem falls into the NP-Complete category.

On the other hand, the optimization variant only requires a graph as input, aiming to identify the smallest number of colors needed for a proper coloring. This minimal number is referred to as the chromatic number, typically represented by the Greek letter χ. Computing the chromatic number for arbitrary graphs is recognized as an NP-Hard problem.

While certain graph coloring algorithms in the literature provide exact solutions, their efficiency diminishes significantly on arbitrary instances, demonstrating exponential running times. Consequently, these algorithms prove effective primarily for small-scale instances of the problem. For larger instances, heuristics, metaheuristics, and approximation algorithms emerge as the more practical options.

The majority of graph coloring algorithms are heuristic in nature, capable of producing solutions of considerable quality at a reasonable computational expense. However, a notable drawback is their inability to guarantee the solutions they generate. The quality of their outputs may vary from nearly optimal to arbitrarily suboptimal. Frequently, these algorithms necessitate repeated execution to assess the quality of a given solution.

This study aims to address the graph coloring problem by employing two heuristic methods: the simulated annealer and the genetic algorithm.

Simulated annealing is a stochastic optimization algorithm inspired by the annealing process in metallurgy. It shares similarities with hill climbing, yet it introduces an element of probabilistic exploration. Unlike its more deterministic counterparts, simulated annealing is willing to occasionally accept worse solutions. This adaptability helps it to escape local minima in complex and non-convex functions. However, it's worth noting that on simpler, convex functions, it may be surpassed in terms of accuracy by more deterministic approaches like hill climbers.

Genetic Algorithms, on the other hand, inspired by the principles of natural selection and genetics, employ evolutionary mechanisms to iteratively improve candidate solutions. This population-based approach allows for exploration of diverse regions within the solution space, making GAs particularly adept at handling complex, multi-modal problems.

We will evaluate these two algorithms – the simulated annealer and the genetic algorithm – using standard benchmark graphs in DIMACS standard format. Section 2 will delve into the implementation details of both algorithms, focusing on modifications derived from the standard algorithms. Sections 3 and 4 will cover the experimental setup and results, which will be compared with the best-known results. Finally, Section 5 will summarize our findings.

2 Methods

In this section, we will delve into the implementation details of the Genetic Algorithm (GA) and the Simulated Annealer (SA).

2.1 The Genetic Algorithm (GA)

The fundamental structure of the genetic algorithm is as follows:

```
$t \leftarrow 0$
Generate the starting population $P(t)$
Evaluate $P(t)$
while not StoppingCondition do
    $t \leftarrow t + 1$
    Select $P(t)$ from $P(t-1)$
    Mutate $P(t)$
    Crossover $P(t)$
    Evaluate $P(t)$
end while
```

2.1.1 Generating the start population

For a more efficient and quicker convergence to the optimum, the maximum outer degree is first calculated for every graph. As per Brooks's theorem, the maximum order of the chromatic number must equal the maximum degree of the graph.

After this upper bound for the chromatic number has been calculated, the chromosomes are generated with random colors from 0 to the upper bound, Δ.

2.1.2 The Evaluate Operator

The evaluate operator is a representation of the total number of distinct colors added to the product of the number of conflicting edges and the conflict penalty, resulting in a significant numerical value. This approach enables the validation of a solution, as an invalid solution would yield a value greater than the Δ value.

$$f(x) = \text{number of colors} + \Delta \times \text{number of conflicting edges}$$

2.1.3 The Mutate Operator

In the field of biology, a mutation signifies a modification in the nucleic acid sequence of an organism's genome, a virus, or extrachromosomal DNA. The genetic algorithm simplifies and emulates these processes using a basic unit: the bit. Unlike the intricate nature of mutations in biology, the genetic algorithm performs mutations by simply flipping a single bit—transitioning from 0 to 1 or from 1 to 0.

The genetic algorithm uses two different mutation operators, depending on how the fitness landscape looks at a certain moment.

The first mutation operator is designed to repair the chromosome in a random manner. Unlike the more conventional mutation operator, which conducts a probability test for each gene in a chromosome, this mutation performs such a test for the entire chromosome. Upon passing the test, each gene is scrutinized for potential conflicts. If a conflict is identified for a specific gene/vertex, the mutation operator scans the adjacency list of that node. It subtracts all colors associated with that node from the entire color range of the chromosome and then selects a random color from the remaining options.

The next two operators have the change to occur if and only if 50 generations have passed without any improvement.

Greedy Improvement

Before activating the second mutation type, a greedy method is applied to the best chromosome in the population. The algorithm scans through all non-conflicting colors in the adjacency list for each gene. If a gene change results in a fitness improvement, the modification is retained.

If the greedy improvement algorithm completes without any enhancements, hypermutation is triggered. This operator systematically examines every gene in each chromosome, conducting a probability test without considering whether the vertex has a conflict or not. Upon successful completion of the random probability test, the color of the vertex is changed to a random color from the chromosome.

2.1.4 The Crossover Operator

In the genetic algorithm, crossover emulates the biological concept of mating. It involves taking two parent chromosomes and combining their genetic material to create one or more offspring chromosomes. Unlike the simplicity of bit flipping in mutation, crossover operates by exchanging entire segments or genes between parent chromosomes.

The crossover process works by randomly selecting a crossover point along the chromosomes. The genetic material beyond this point is swapped between the parents, generating new offspring chromosomes. This mechanism introduces diversity and the potential for novel combinations of genetic information in the algorithm's population, aiding in the exploitation of the current material.

Similar to the mutation scenario, two different crossover operators are employed, each utilizing a single-point randomly generated cut point.

The first crossover operator executes crossover by fitness. The chromosomes are sorted by fitness, and with a certain probability, they are paired and subjected to the crossover operation. On the other hand, the second crossover operator is activated during the hypermutation phase. It involves randomly shuffling the population and then applying crossover. This approach aims to enhance exploration, considering that the mutation operator is quite effective at rectifying conflicting solutions.

2.1.5 The Selection Operator

In the genetic algorithm, the Wheel of Fortune selection method introduces a level of randomness and chance inspired by nature. Instead of directly choosing the best chromosomes, this operator employs a metaphorical spinning wheel to determine the probability of selecting each chromosome for the next generation. Think of each chromosome as a segment on this wheel, and the size of the segment is directly proportional to the chromosome's fitness or desirability.

During the selection process, the wheel spins, and where it stops dictates which chromosomes are chosen to form the next generation. Fitter chromosomes occupy more significant portions of the wheel, increasing their likelihood of being selected, while less fit ones have smaller segments and a lower probability of being chosen.

This approach introduces a stochastic element into the genetic algorithm, bringing variability to the selection. By simulating the element of chance, the Wheel of Fortune operator enhances the algorithm's exploration of diverse genetic material. This variability fosters the creation of novel offspring, contributing to the algorithm's ability to discover optimal solutions by encouraging the exploration of a broad solution space.

During the implementation of this operator, various experiments were carried out, exploring modified selection pressures and the inclusion of an elite pool. However, empirical evidence consistently demonstrated that the best results were achieved using the default operator, maintaining the non-modified selection pressure, and without employing an elite pool.

2.2 The Simulated Annealer (SA)

The cost function is essentially a measure of the total number of colors utilized in a solution, specifically referring to distinct integers assigned.

To initiate the process, an initial solution is created by randomly assigning colors to various nodes, drawing from a pool of $\Delta + 1$ colors, where Δ represents the the graph degree. The initial solution may or may not represent a valid coloration, and its validity is checked using an operator. If the solution is found to be invalid, it is not discarded; instead, a repair operator is employed to convert it into a valid one.

The repair operator assesses whether neighboring nodes share the same color. In cases where such a conflict is identified, one of the conflicting colors is replaced with a randomly generated color, excluding the original color. The repair process involves identifying the vertex with the maximum conflict in the solution and iteratively changing its color until a valid solution is obtained.

In addition to the repair operator, a random change operator is introduced for generating neighboring solutions. This operator operates at a specific point in the solution, randomly choosing a position and altering the color at that position using a randomly selected color from the range of available colors. The resulting solution is then checked for validity, and if it is valid, the process continues. If not, the repair operator is invoked to rectify the coloration.

2.3 The Graph Coloring Problem (GCP)

Let $G = (V, E)$ be a graph where V is a set of vertices and E is a set of edges. A k-coloring of G is a partition of V into k sets $\{V_1, \dots, V_k\}$, such that no two vertices in the same set are adjacent, i.e., if v, w belong to V_i, $1 \leq i \leq k$, then (v, w) does not belong to E. The sets $\{V_1, \dots, V_k\}$ are referred to as colors. The chromatic number, $\chi(G)$, is defined as the minimum k for which G is k-colorable. The Graph k-Colorability Problem (GCP) can be stated as follows. Given a graph G, find $\chi(G)$ and the corresponding coloring. GCP is an NP-hard problem.

3 Experimental Setup

The initial solutions for the Genetic Algorithm (GA) and Simulated Annealer (SA) were generated using a 32-bit Mersenne Twister in C++. The function `std::random_device` was employed to harness system entropy, enhancing the randomness of the initialization.

For the majority of the graphs, a statistical sample size of 30 experiments was utilized. However, for the last two instances (DSJC250.5 and DSJC500.5), a more modest number of 10 experiments was conducted. This adjustment was made due to their intricate structure and the limited time available for these experiments.

As for the Genetic Algorithm, the following parameters were applied:

- Generations number: 4000
- Population size: 50
- Mutation probability: 50% (meaning 1 in 2 chromosomes undergo the repair process)
- Crossover probability: 30%
- Hypermutation triggered after 50 generations without improvement with a 10% chance for every gene

For the Simulated Annealer:

- Initial temperature: 1000
- Decay factor: 0.001 substracted after every generation
- Stop condition: The temperature reaches 0

4 Experimental Results

Graph	$\|V\|$	$\|E\|$	$\chi(G)$	$\hat{GA}$	$\tilde{GA}$	$\hat{SA}$	$\tilde{SA}$
myciel5	47	236	6	6	6	6	8
myciel6	95	755	7	7	7	7	10
games120	120	638	9	9	9	11	14
miles1000	128	3216	42	42	44	47	52
fpsol2.i.1	496	11654	65	65	67	73	80
queen7_7	49	476	7	7	8	12	17
queen8_8	64	728	9	10	11	16	20
le450_5a	450	5714	5	7	9	15	21
DSJC250.5	250	31366	28	37	41	60	65
DSJC500.5	500	125249	47	62	66	81	87

Table 1: Summary of performance for the Graph k-Colorability Problem

5 Conclusion

In conclusion, this paper explored the application of a genetic algorithm and a simulated annealer in solving the graph coloring problem. The results revealed that the genetic algorithm exhibited superior performance in handling more intricate functions compared to the simulated annealer. However, it is crucial to note that the genetic algorithm fell short of achieving the chromatic number of more complex graphs and was surpassed by more robust algorithms documented in the literature.

References

[1] Musa M. Hindi, Roman V. Yampolskiy: Genetic Algorithm Applied to the Graph Coloring Problem `https://ceur-ws.org/Vol-841/submission_10.pdf`

[2] José Aguilar-Canepa1, Rolando Menchaca-Méndez1, Ricardo Menchaca-Méndez1, Jesús García: A Structure-Driven Genetic Algorithm for Graph Coloring `https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-55462021000300465`

[3] Huberto Ayanegui and Alberto Chavez-Aragon: A complete algorithm to solve the graph coloring problem `https://ceur-ws.org/Vol-533/09_LANMR09_06.pdf`

[4] Anindya Jyoti Pala, Biman Rayb, Nordin Zakariaa, Samar Sen Sarma: Comparative Performance of Modified Simulated Annealing with Simple Simulated Annealing for Graph Coloring Problem `https://www.sciencedirect.com/science/article/pii/S187705091200155X?ref=cra_js_challenge&fr=RR-1`

[5] Eugen Croitoru, Teaching: Genetic Algorithms `https://profs.info.uaic.ro/~eugennc/teaching/ga/`

[6] GeeksforGeeks: Genetic Algorithms for Graph Colouring — Project Idea https://www.geeksforgeeks.org/project-idea-genetic-algorithms-for-graph-colouring/

[7] Graph Coloring Instances https://mat.gsia.cmu.edu/COLOR/instances.html

[8] Wikipedia, Simulated annealing. https://en.wikipedia.org/wiki/Simulated_annealing

[9] Wikipedia, Genetic Algorithms. https://en.wikipedia.org/wiki/Genetic_algorithm

[10] Wikipedia, Crossover. https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[11] Wikipedia, Crossover. https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[12] Wikipedia, Selection. https://en.wikipedia.org/wiki/Fitness_proportionate_selection