Pol Pujol Santaella

Neuronal & Evolutionary Computation

Optimization with Genetic Algorithms

UNIVERSITAT ROVIRA I VIRGILI

2025

INDEX

1. Introduction

This project involves the design and implementation of a Genetic Algorithm (GA) to solve the Graph Coloring Problem (GCP), a classic combinatorial optimization challenge in computer science. The primary objective is to assign a color to each vertex of a given graph such that no two adjacent vertices share the same color (valid coloring), while simultaneously minimizing the total number of colors used (chromatic number optimization). By leveraging evolutionary principles such as natural selection, recombination, and mutation, the algorithm explores the vast search space of potential coloring to converge on high-quality solutions. The efficacy of the GA is evaluated across the three graph instances of increasing complexity: small, medium and large graphs, using a variety of genetic operators to identify the most robust parameter configuration for different problem scales.

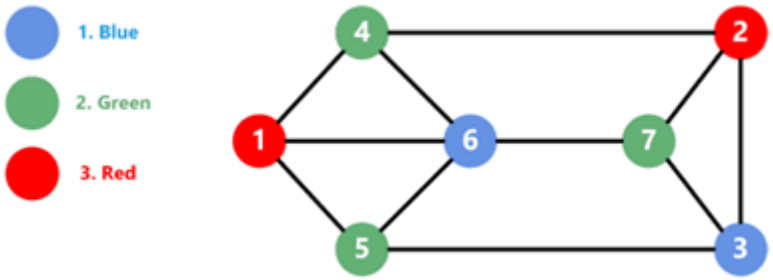
GitHub Repository: [PolPujolSantaella/Activity2-NEC-ML: Optimization with Genetic Algorithms]

The Graph Coloring Problem

Starting from a graph consisting of V vertices and E edges connecting them, we have to assign a color to each vertex of the graph such that the two following conditions are fulfilled:

- **No two adjacent vertices have the same color.** Remember that two vertices are considered adjacent if an edge connects them.
- **The number of colors used is the minimum possible.**

For example, let's consider the following graph, which complies with the two requirements described before:



2. Chromosome Description and Algorithm Adaptations

2.1. Chromosome Representation

File: *src/chromosome.py*

To solve the Graph Coloring Problem (GCP), we have implemented an integer-based vector encoding. This representation serves as the “blueprint” of a potential solution, where each individual in the population represents a complete coloring assignment for the graph.

Each solution is encapsulated within the Individual class. The chromosome is structure as follows:

- **Type**: Integer-based vector (using numpy.ndarray for computational efficiency)
- **Length (V)**: The length of the vector corresponds exactly to the number of vertices in the graph.
- **Genes (g_i)**: Each gene $\mathbf{g_i}$ at index *i* represents node *i*. The value of the gene is an integer *c {0, 1…, k -1},* where k is the maximum number of colors allowed.

The initial population is generated using Random Initialization. Each gene is assigned a value from a discrete uniform distribution between 0 and *max_colors - 1.* This ensures high genetic diversity at the start of the evolutionary process.

2.2. Fitness Function and Validation

File: *src/fitness.py*

The objective is twofold: to ensure no two adjacent vertices share the same color and to minimize the total number of colors used. We have implemented a fitness function that penalizes both constraint violations (conflicts) and the cardinality of the color set.

The function evaluate_fitness calculates a numerical value representing the “cost” of a solution. Since this is a minimization problem, a lower fitness score indicates a better-performing individual.

- Conflict Detection: We compare the color assigned to node u with node v for every edge (u, v) in the graph. A conflict occurs if ucolor == vcolor.
- Color Cardinality: We calculate the number of unique integers present in the chromosome to determine the total colors used.
- Weighted Penalty: To prioritize validity over optimization, we apply a conflict_weight. This ensures that a solution with even a single conflict is considered significantly worse than any valid solution, regardless of how many colors the valid solution uses.

Function

- W is the conflict_weight (penalty multiplier)
- C is the total number of edge conflicts.
- K is the number of unique colors used.

$$Fitness = (W \times C) + K$$

A chromosome is considered a valid solution only when the number of conflicts C equals 0. While the algorithm may explore “invalid” individuals during the evolutionary process, the high penalty weight ensures the population evolves toward the feasible region (0 conflict) before focusing on minimizing K.

2.3. Genetic Operators Implementation

File: src/genetic_operators.py

To ensure robustness and meet the project requirements, we implemented two distinct techniques for each genetic operator:

Selection

Selection determines which individuals from the current population will pass their genetic information to the next generation. We focus on identifying individuals with lower fitness values, as our objective is cost minimization.

1. **Tournament Selection:** A subset of k individuals is chosen at random, and the one with the best (lowest) fitness is selected.

2. **Roulette Wheel Selection:** Since the GCP is a minimization problem, we adapt the standard roulette wheel by using the inverse of the fitness (1.0 / (Fitness + 1.0)). This ensures that individuals with fewer conflicts and fewer colors have a higher probability of being selected.

Crossover

Crossover (recombination) combines genetic material from two parents to produce offspring.

1. **Single-Point Crossover:** A single split point is randomly selected along the chromosome. The genes to the left of the point are taken from the 1st parent, and the genes to the right are taken from the second parent.

2. **Uniform Crossover:** For every node in the graph, we decide which parent provides the color based on a fixed probability 0.5. This provides a higher level of shuffling and is particularly useful for exploring diverse color combinations across the graph.

Mutation

Mutation introduces random variations to the population to prevent the algorithm from getting stuck in local optima.

1. **Single Gene Mutation:** Exactly one node in the graph is selected at random, and its color is changed to a different valid color within the range [0, k - 1]. This represents a small controlled step in the search space.

2. **Independent Gene Mutation:** This operator specifically targets conflicted nodes. The algorithm first identifies all vertices u and v that are a conflict. Each node has a probability (mutation_rate) of being assigned a new random color.

2.4. System Dynamics and Adaptations

Parameters across the three scales

Choosing the appropriate hyperparameters is critical for GA success, as the search space grows exponentially with the number of vertices (V) and the available colors (k).

The different parameters are chosen based on the graph's complexity.

- **Small Graphs:** ~10 nodes
- **Medium Graphs:** ~80 nodes
- **Large Graphs:** ~450 nodes

PARAMETER	SMALL	MEDIUM	LARGE	RATIONALE
Population	100	400	500	Scales with node count to ensure sufficient genetic diversity for recombination
Max Generations	500	1000	2000	Increased to allow the system to coordinate colors across thousands of edge constraints.
Max Colors	10 (Chromatic Number = 4)	14 (Chromatic Number = 10	22 (Chromatic Number = 15	Set slightly above the known chromatic number to force the GA to optimize color count.
Elitism	2	5	10	Preserves the best solutions found so far, preventing the loss of the optimal coloring during stochastic selection.
Mutation Rate	0.1	0.05	0.02	High in small graphs to avoid local minima; low in large graphs to prevent destroying fit solutions.

Elitism

We utilize Elitism to ensure that the “hall of fame” individuals (the valid colorings with the minimum number of colors) are never lost due to the randomness of crossover or mutation. By carrying over the top 2 to 10 individuals directly to the next generation, we guarantee that the fitness of the population monotonically improves or stays the same, never degrades.

Stationary State Identification

To optimize execution time and prevent unnecessary computations, we implemented a mechanism to detect when the algorithm reaches a stationary state.

- The STATIONARY_LIMIT: We track the fitness of the best individual across generations. If the best fitness value does not improve for a specific number of consecutive generations, the system is considered to have converged.

- Dynamic Thresholds
    - Small Graphs: 100 generations
    - Medium Graphs: 300 generation
    - Large Graphs: 500 generations

For small graphs, the limit is set to 100, as the search space is small and convergence happens quickly. For Large graphs, the limit is increased to 500 to allow the algorithm more time to escape plateau regions in a much more complex fitness landscape.

3. Experimental Results and Analysis

3.1. Small Graph Instance: myciel3.col

Dataset Description

- **Source:** https://mat.tepper.cmu.edu/COLOR/instances.html
- **Characteristics:** This instance is based on the Mycielski transformation. It is a "triangle-free" graph designed to have a high coloring number relative to its size.
- **Complexity:**
  - *Nodes (V):* 11
  - *Edges (E):* 20
  - **Target**: The known optimal chromatic number for this instance is 4

Comparative Analysis of Parameters

Exp	Selection	Crossover	Mutation	Best Fitness	Conflicts	Colors
1	Tournament	Single-Point	Single Gene	4	0	4
2	Tournament	Single-Point	Independent	4	0	4
3	Tournament	Uniform	Single Gene	4	0	4
4	Tournament	Uniform	Independent	4	0	4
5	Roulette Wheel	Single-Point	Single Gene	4	0	4
6	Roulette Wheel	Uniform	Independent	4	0	4

In this small-scale instance, every experimental combination successfully reached the global optimum. A fitness value of 4 was achieved in all cases, representing 0 conflicts and exactly 4 colors.

This uniformity indicates that for low-complexity graphs (~10 nodes), the Genetic Algorithm is highly robust. The search space is sufficiently small that any combination of selection pressure and genetic shuffling can navigate to a valid, optimal coloring before reaching the stationary state.

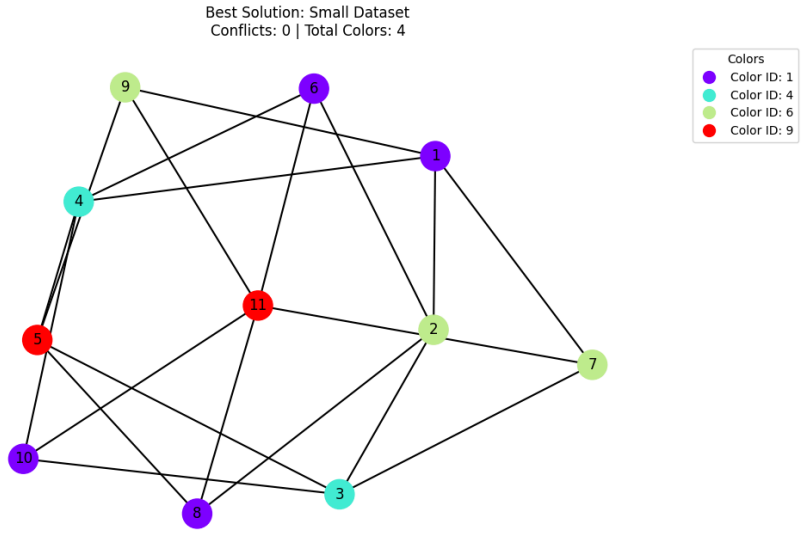
Best Solution Performance

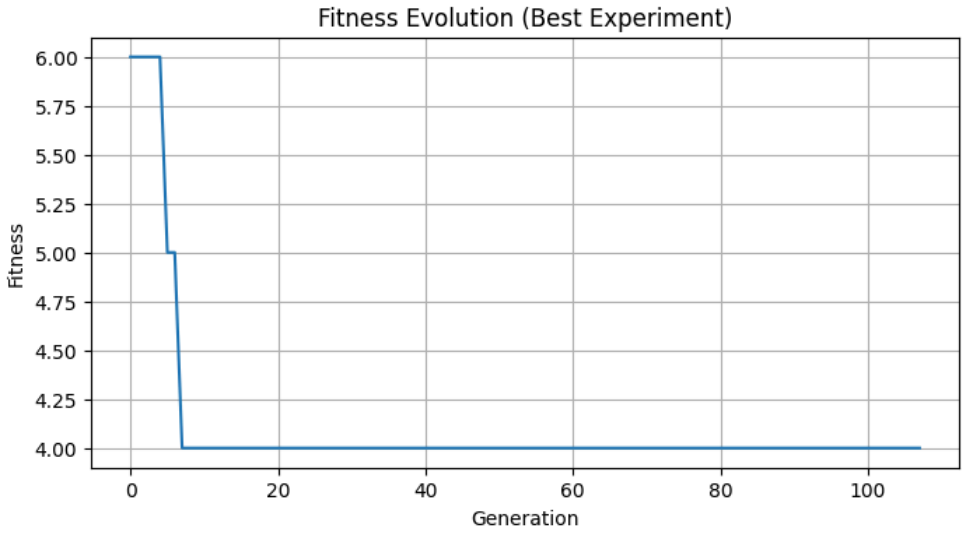
The best individual achieved a perfect coloring using only 4 colors with 0 conflicts.

Fitness Evolution

The algorithm starts with a fitness of 6.0, this means that the algorithm achieved a solution with 0 conflicts and 6 colors used. It successfully resolves initial edge conflicts and then reduces the color count, dropping sharply to a fitness of 5.0.

Between generations 2 and 10, the population maintains a plateau, and around generation 10, the algorithm successfully finds a 4-color valid configuration (Fitness = 4.0). After generation 16, the fitness remains constant until the termination criteria are met at approximately generation 106. This demonstrates that the STATIONARY LIMIT correctly identifies when no further optimization is possible.





3.2. Medium Graph Instance: queen9_9.col

Dataset Description

- **Source:** https://mat.tepper.cmu.edu/COLOR/instances.html
- **Characteristics:** This graph represents the legal moves of a queen on a 9 x 9 chessboard. Each node represents a square on the board, and edges connect any two squares where a queen can move legally (horizontally, vertically, or diagonally).
- **Complexity:**
    - *Nodes (V):* 81
    - *Edges (E):* 2112
    - **Target**: The known optimal chromatic number for this instance is 10.

Comparative Analysis of Parameters

Exp	Selection	Crossover	Mutation	Best Fitness	Conflicts	Colors
1	Tournament	Single-Point	Single Gene	13	0	13
2	Tournament	Single-Point	Independent	14	0	14
3	Tournament	Uniform	Single Gene	14	0	14
4	Tournament	Uniform	Independent	14	0	14
5	Roulette Wheel	Single-Point	Single Gene	13	0	13
6	Roulette Wheel	Uniform	Independent	14	0	14

This medium-scale instance highlights the difficulty of reaching the theoretical minimum of 10 colors as graph density increases.

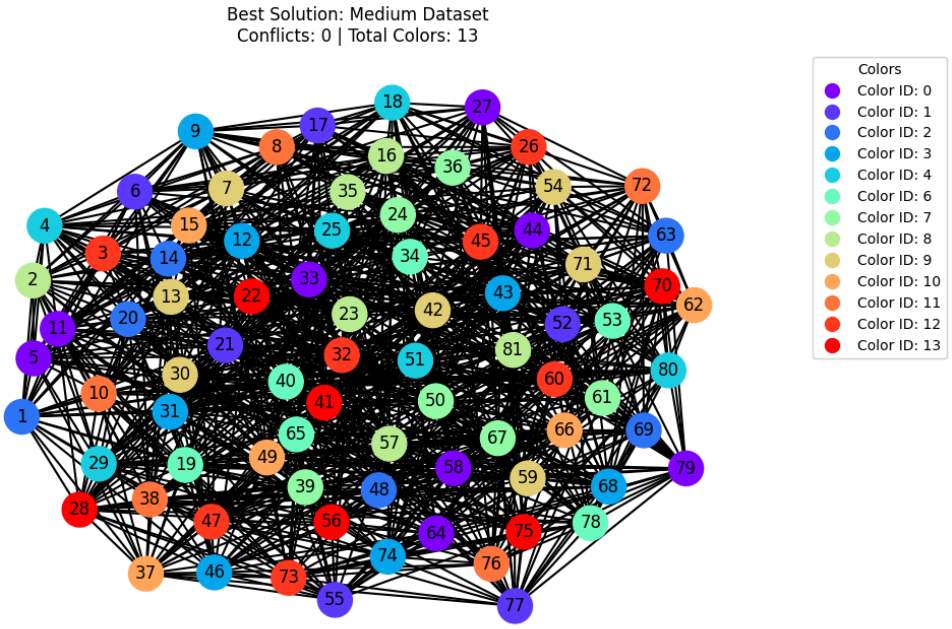
- All configurations achieved a valid coloring (0 conflicts). This confirms that the algorithm successfully navigates the hard constraints of the Graph Coloring Problem even when the vertex-to-edge ratio is high
- The experiment 1 (Tournament) and experiment 5 (Roulette Wheel), both utilizing one-point crossover and Single Gene Mutation, reached the best result of 13 colors.
- While the algorithm found valid colorings, there remains a gap between out result (13) and the global optimum (10).

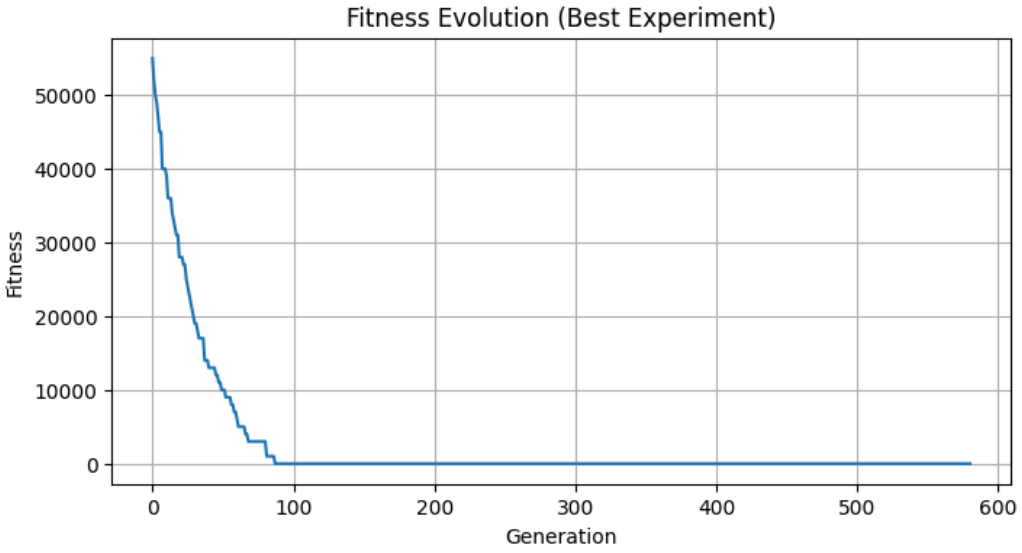
Best Solution Performance

The best individual achieved a perfect coloring using only 13 colors with 0 conflicts.

Fitness Evolution

The plot begins at an extremely high fitness value (> 50.000), which is dominated by the conflict_weight penalty. A sharp, exponential decline occurs within the first 100 generations as the GA effectively resolves the majority of neighbor conflicts. From generation 100 through nearly generation 600, the fitness remains stable at 13.0. This indicates that while the GA successfully maintained a zero-conflict solution, it reached a stationary state where no further reduction in the number of colors was achieved before the STATIONARY LIMIT triggered termination.





3.3. Large Graph Instance: le450_15b.col

Dataset Description

- **Source:** https://mat.tepper.cmu.edu/COLOR/instances.html
- **Characteristics:** This instance is a Leighton graph generated with a specific clique vector, including 40 cliques of size 25. It is designed to be a “sparse” but difficult benchmark for coloring algorithms.
- **Complexity:**
  - *Nodes (V):* 45
  - *Edges (E):* 8169
  - **Target**: The known optimal chromatic number for this instance is 15

Comparative Analysis of Parameters

Exp	Selection	Crossover	Mutation	Best Fitness	Conflicts	Colors
1	Tournament	Single-Point	Single Gene	1022	1	22
2	Tournament	Single-Point	Independent	22	0	22
3	Tournament	Uniform	Single Gene	22	0	22
4	Tournament	Uniform	Independent	22	0	22
5	Roulette Wheel	Single-Point	Single Gene	22	0	22
6	Roulette Wheel	Uniform	Independent	62022	62	22

This large scale instance represents the most significant challenge for the GA, revealing critical insights into operator performance at scale:

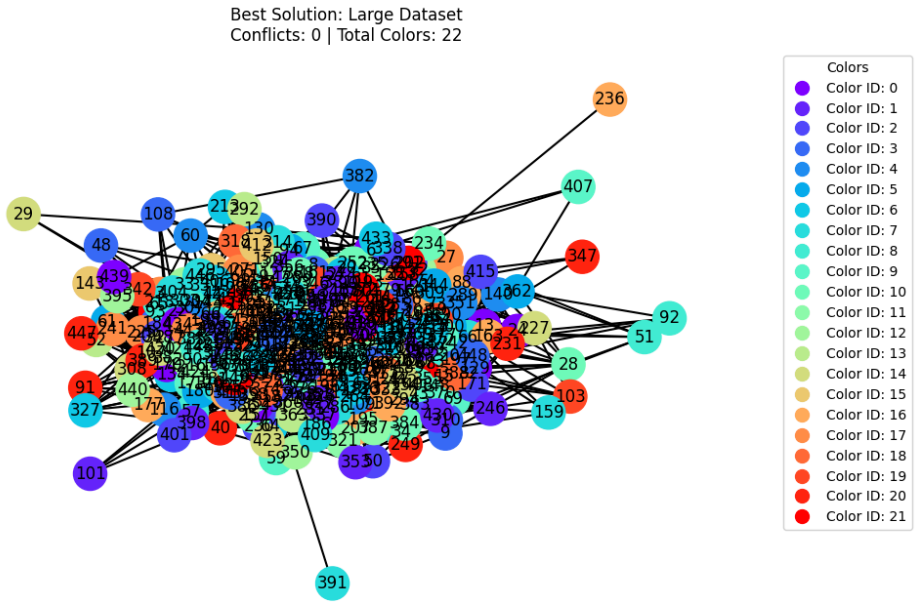
- In Experiment 2, the Targeted Independent Mutation successfully reached a feasible solution (0 conflicts), whereas the Single Gene Mutation in Experiment 1 failed to resolve the final conflict within the same parameters.
- While most combinations reached a valid coloring with 22 colors, Experiment 6 struggled significantly, ending with 62 conflicts. This suggests that for very large graphs, the lower selection pressure of Roulette Wheel combined with high-disruption Uniform Crossover can hinder the algorithm’s ability to “converge” on a valid solution.
- While we achieved 0 conflicts with 22 colors, this is significantly higher than the theoretical optimum of 15.

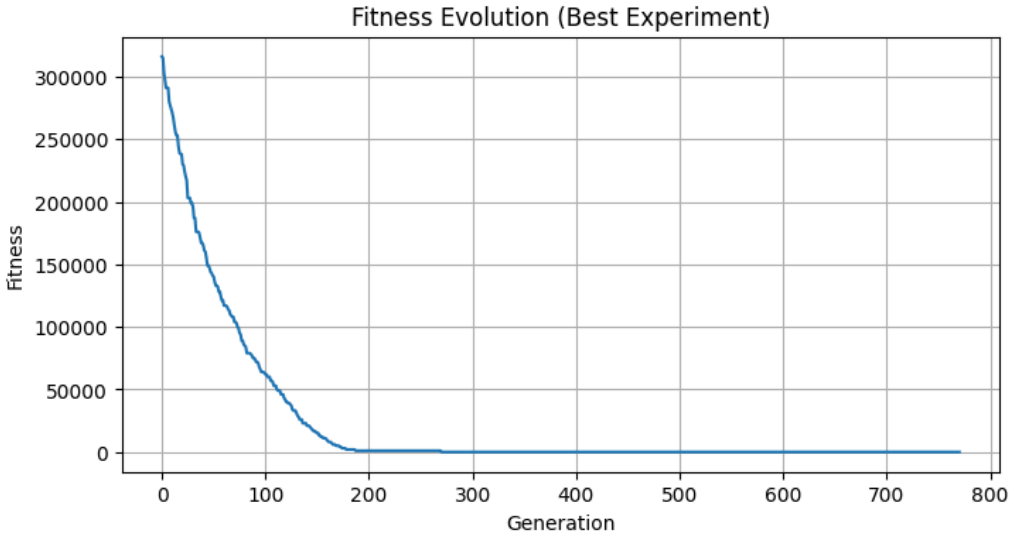
Best Solution Performance

The best individual achieved a perfect coloring using only 22 colors with 0 conflicts.

Fitness Evolution

The algorithm begins with a fitness value exceeding 300.000. This extreme starting cost is a result of the high number of initial edge conflicts across the 8169 edges when nodes are randomly colored. Similar to the medium dataset, there is a consistent and steep decline in fitness during the first 200 generations. This period represents the “constraint satisfaction” phase where the algorithm aggressively eliminates conflicts. The fitness reaches a stable plateau around generation 250, indicating that a valid coloring has been achieved. The algorithm continues to run until approximately generation 780 without further improvement in fitness.





4. Optional: Comparative Study with Alternative Optimization Methods

To evaluate the efficiency of the Genetic Algorithm, two additional optimization metaheuristics were implemented: Simulated Annealing (SA) and Tabu Search (TS). Both methods were adapted to reuse the Individual class and the evaluate_fitness function developed for the GA to ensure a fair comparison.

4.1. Methodology and Parameter Selection

Simulated Annealing

Simulated Annealing is a stochastic local search algorithm that mimics the cooling process of metals. It allows “bad moves” (accepting worse solutions) with a probability that decreases over time, governed by the temperature.

- **Parameters:** We tested three cooling rates (0.95, 0.99, 0.999)
- **Selection:** The 0.999 rate was chosen for the final comparison as it provides a slower cooling process, allowing the algorithm more time to escape local optima in complex graphs.

Tabu Search

Tabu Search is a deterministic metaheuristic that uses a “Tabu List” to record recently visited solutions, preventing the algorithm from cycling and forcing the exploration of new areas in the search space.

- **Parameters:** We tested Tabu List sizes of 10, 30, and 70
- **Selection:** A size of 70 proved most effective for larger datasets, as it provides a larger “memory” to avoid local optima in high-dimensional spaces like the Large Dataset.

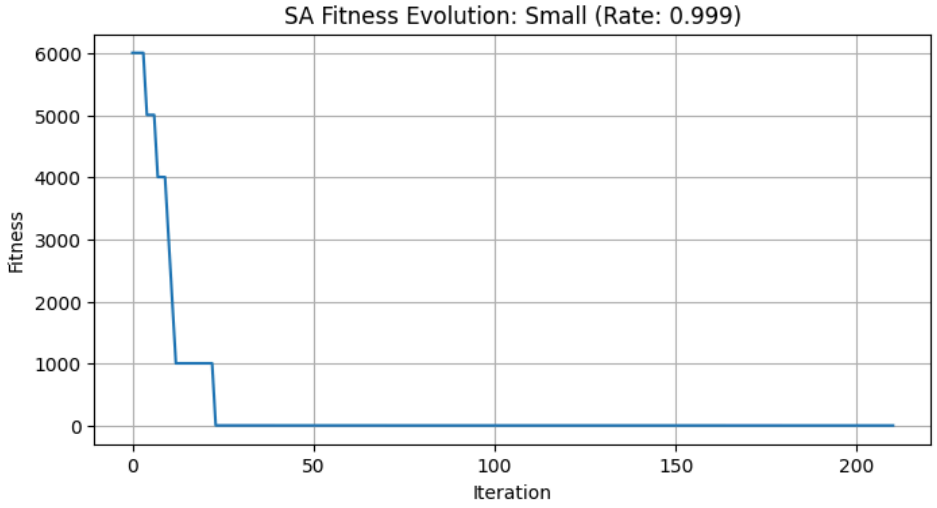
4.2. Experimental Results

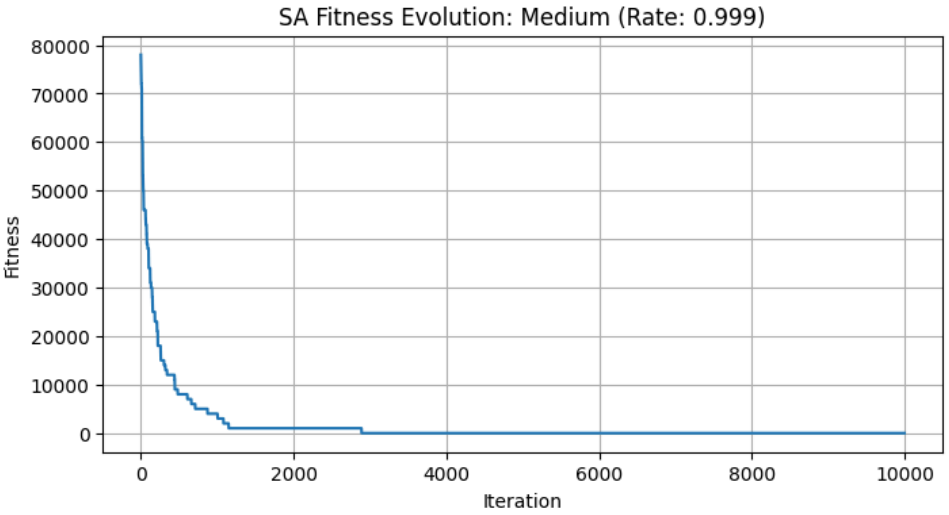
The following table summarizes the best performance of each method across the three dataset (using the best parameter configuration for each):

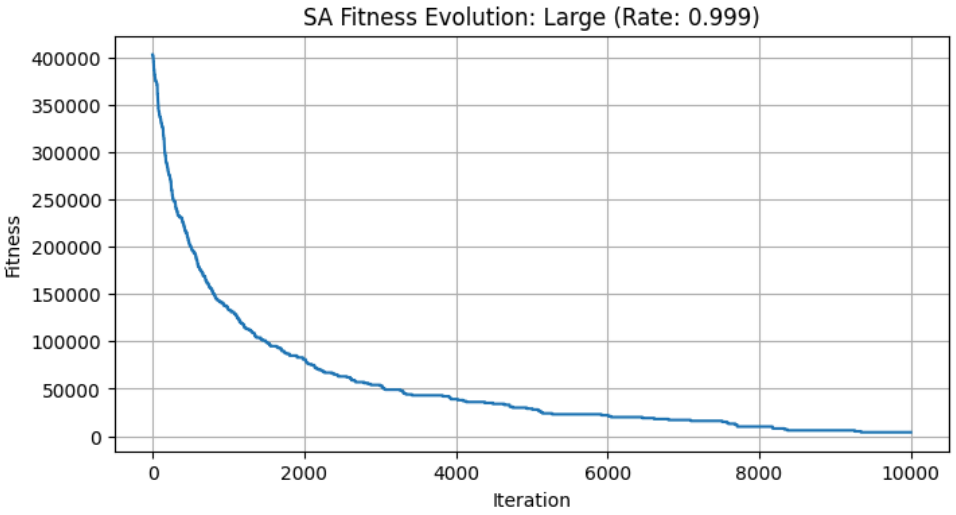
DATASET	METHOD	FITNESS	CONFLICTS	COLORS	TIME
SMALL	Genetic Algorithm	4	0	4	0.172
	Simulated Annealing	4	0	4	0.142
	Tabu Search	4	0	4	0.406
MEDIUM	Genetic Algorithm	14	0	14	3.150
	Simulated Annealing	14	0	14	0.164
	Tabu Search	14	0	14	0.549
LARGE	Genetic Algorithm	22	0	22	23.705
	Simulated Annealing	4022	4	22	0.379
	Tabu Search	1022	1	22	1.421

4.3. Fitness Evolution Plots (Simulated Annealing & Tabu Search)

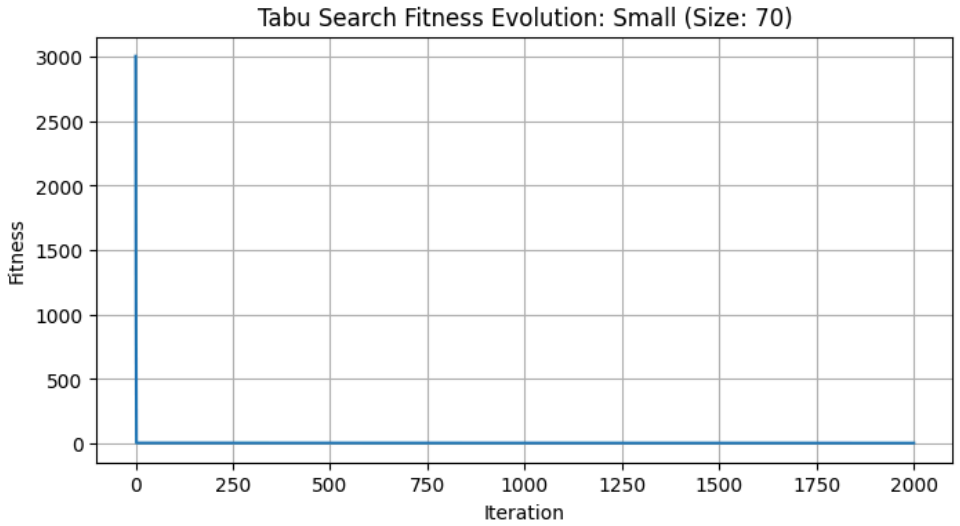
Simulated Annealing

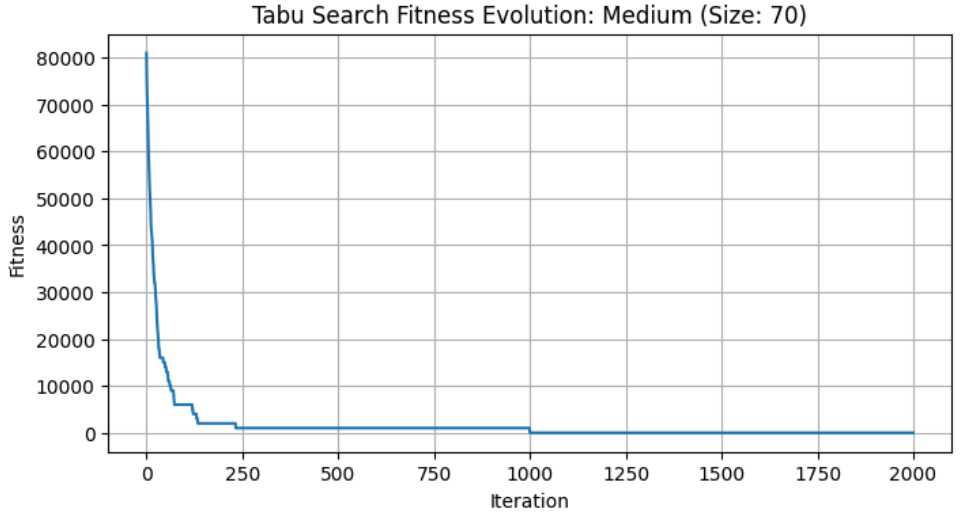






Tabu Search


Tabu Search Fitness Evolution: Small (Size: 70)
Fitness
Iteration
3000
2500
2000
1500
1000
500
0
0
250
500
750
1000
1250
1500
1750
2000


Tabu Search Fitness Evolution: Medium (Size: 70)
Fitness
Iteration
80000
70000
60000
50000
40000
30000
20000
10000
0
0
250
500
750
1000
1250
1500
1750
2000


Tabu Search Fitness Evolution: Large (Size: 70)
Fitness
Iteration
350000
300000
250000
200000
150000
100000
50000
0
0
250
500
750
1000
1250
1500
1750
2000

4.4. Comparative Analysis and Conclusions

1. **Solution Quality vs. Complexity:**

In Small and Medium datasets, all 3 algorithms successfully reached a valid coloring (0 conflicts).

In the Large Dataset, the Genetic Algorithm was the only method capable of reaching a perfect solution (0 conflicts). This suggests that for highly complex graphs with thousands of edges, the population-based approach of the GA is superior at maintaining diversity and finding global optima compared to single-state trajectory methods like SA or TS.

2. **Time Efficiency:**

Simulated Annealing is significantly faster than the other methods, completing executions in less than 0.4 seconds even for the large dataset. However, its speed comes at the cost of solution quality in the most complex instance.

Tabu Search showed a very consistent time performance but struggled slightly to eliminate the last few conflicts in the Large dataset within the allocated iteration.

The Genetic Algorithm is the most computationally expensive (23.7s for the large graph) because it evaluates a full population in every generation, but it proves to be the most robust for constraint satisfaction.

3. **Conclusion**

While Simulated Annealing is excellent for quick approximations, the Genetic Algorithm is the most reliable choice for the Graph Coloring Problem when the objective is to find a valid solution in complex, large-scale constraints.