A Graph-Aware Genetic Algorithm for Graph Coloring: Implementation and Empirical Evaluation

Yiduo Yu
yuyid@kean.edu

Yixuan Mi
miy@kean.edu

December 17, 2025

Abstract

Graph coloring is a classic NP-hard combinatorial optimization problem with broad applications such as timetabling, register allocation, frequency assignment, and map coloring. Since computing the chromatic number is infeasible for general graphs at scale, practical solutions rely heavily on heuristics and metaheuristics.

In this project, we implement and empirically evaluate six representative coloring approaches, ranging from greedy heuristics, to a dynamic saturation heuristic, to a randomized greedy baseline, and to a graph-aware genetic algorithm (GA). Our GA design incorporates several problem-specific mechanisms: a greedy-based upper bound that reduces the search space, a lexicographic fitness function that prioritizes feasibility before compactness, a repair operator that maintains valid colorings after variation, and a guided mutation strategy that prefers non-conflicting recolorings. Through controlled experiments on random graphs, structured graphs, and adversarial instances, we analyze both solution quality and efficiency. The results show that while DSatur achieves an excellent quality–time trade-off in most settings, the proposed graph-aware GA provides a flexible framework for integrating structural heuristics and remains a promising direction for further hybridization and optimization.

Our code is available at https://github.com/EthanYixuanMi/Improved-GA-for-Graph-Coloring

Keywords: Graph Coloring, Genetic Algorithm, Repair Operator, Lexicographic Fitness, DSatur, Heuristics.

1 Introduction

Graph coloring is a fundamental problem in combinatorial optimization and graph theory, playing a crucial role in understanding the structural complexity of discrete systems. The problem requires assigning colors to the vertices of a graph such that adjacent vertices have different colors, with the goal of minimizing the total number of colors used. This minimum number is called the chromatic number, which reflects fundamental combinatorial properties of the underlying graph and makes graph coloring a core tool in a wide range of application areas, including timetabling and course scheduling, register allocation in compilers[3], frequency assignment in wireless networks, pattern classification, and constraint-based scheduling.

Despite its long history and extensive research, determining the chromatic number remains an NP-hard problem, and even approximating it within tight bounds is computationally intractable for general graphs[9]. Classic exact algorithms, such as backtracking or branch-and-bound methods, guarantee optimality but become computationally prohibitive as the size or density of the graph increases. To address these limitations, a large number of heuristic and metaheuristic methods have been developed. These heuristics often achieve impressive performance in practical applications, but they still suffer from some critical vulnerabilities: small changes in graph structure, vertex ordering, or density can drastically affect the quality of the solution. Therefore, understanding when and why certain heuristic strategies succeed, and identifying which graph families expose their weaknesses, remains an open and practically relevant research [10].

Inspired by these challenges, this project conducts a systematic study of heuristic methods for graph coloring. We investigate a range of algorithms—from basic greedy approaches to enhanced heuristics such as DSATUR-based selection rules and local search-driven improvement procedures—and analyze their performance on graphs with varying structural characteristics[1]. We aim to elucidate how heuristic design impacts algorithmic efficiency and solution quality, and to identify "hard" input types that may cause simple greedy strategies to fail. By combining empirical evaluation with careful analysis of graph properties, this work contributes to a deeper understanding of the inherent trade-offs in heuristic graph coloring and provides relevant insights for both theoretical research and practical applications of coloring algorithms.

Contributions:

This work makes two main contributions. First, we design and implement a *graph-aware genetic algorithm* for graph coloring by integrating domain-specific components, including (i) a greedy-based upper bound on the color range, (ii) a feasibility-preserving repair operator applied after crossover and mutation, and (iii) a lexicographic fitness objective that prioritizes conflict elimination before minimizing the number of used colors. Second, we conduct a systematic empirical evaluation across random, structured, and adversarial graph families to characterize when classical heuristics (especially DSatur) dominate in the quality–time trade-off and when the proposed GA provides competitive or complementary behavior.

2 Related Work

Graph coloring, a fundamental NP-hard problem in combinatorial optimization, has been extensively studied. Early research focused on exact algorithms and the theoretical limits of chromatic numbers, but due to computational complexity, practical applications primarily rely on heuristic algorithms. Greedy coloring algorithms, which color vertices sequentially according to a fixed order, are among the earliest and simplest methods for solving this problem. To improve the performance of greedy algorithms, variants such as Maximum Degree First (LDF) and Welsh-Powell sorting have been proposed, prioritizing vertices by height [13]. While these heuristic algorithms are computationally efficient, their performance may be poor on adversarial graphs or highly structured graphs.

A significant advancement in the field of heuristic graph coloring is the DSatur algorithm proposed by Brélaz in 1979[1]. The DSatur algorithm dynamically selects the vertex with the highest saturation degree, allowing the algorithm to adapt to changing local coloring situations. Due to its strong computational performance and relatively low computational cost, DSatur and its extensions remain influential to this day. Many hybrid heuristic algorithms build upon DSatur by adding local search or backtracking components to improve the algorithm[12].

Besides deterministic heuristic algorithms, a large body of research has applied metaheuristic algorithms to the graph coloring problem. Genetic algorithms (GAs) were among the earliest evolutionary algorithms used to solve this problem [6]. In typical GA-based approaches, coloring schemes are encoded as chromosomes, and the fitness function penalizes conflicts or aims to reduce the total number of colors. Later studies introduced specialized crossover operators, adaptive mutation strategies, or hybrid schemes combining GAs with local search (e.g., tabu search or Kempe chain moves) [5]. However, due to the generality of the evolutionary operators, many early designs suffered from slow convergence or a high proportion of infeasible individuals.

Our work follows this metaheuristic research direction but emphasizes *problem-aware* GA design for graph coloring. Specifically, we incorporate (i) a greedy-based upper bound on the number of colors to reduce the evolutionary search space, (ii) a lexicographic fitness function that prioritizes feasibility (conflict minimization) before compactness (color count), (iii) a repair operator to maintain valid colorings after crossover and mutation, and (iv) a guided mutation strategy that preferentially assigns non-conflicting colors. These mechanisms are consistent with the general principle that evolutionary operators should be tailored to domain structure [4], and they provide a concrete, implementation-level template for feasibility-preserving GA-based graph coloring.

3 Methodology

In this project, we study and compare six graph coloring algorithms, ranging from simple greedy heuristics to a more advanced evolutionary approach.

The most basic method is the greedy coloring algorithm, which processes vertices sequentially in the given order. When a vertex is considered, it is assigned the smallest available color that does not conflict with its already colored neighbors. This algorithm is computationally efficient, but its performance is highly sensitive to the vertex ordering and does not guarantee a near-optimal solution.

The Largest Degree First (LDF) algorithm improves upon basic greedy coloring by reordering vertices in descending order of degree before coloring. The intuition is that vertices with more neighbors impose stronger constraints and should be colored earlier while more color choices remain available. Although still greedy in nature, LDF often yields better colorings than an arbitrary ordering.

The Welsh–Powell algorithm further extends this idea by assigning the same color to as many non-adjacent vertices as possible in each iteration. After sorting vertices by decreasing degree, it repeatedly selects the first uncolored vertex, assigns it a new color, and then colors all other uncolored vertices that do not conflict with it using the same color. This batch-coloring strategy can reduce the total number of colors compared to simple greedy methods.

The DSatur algorithm adopts a dynamic ordering strategy based on vertex saturation degree, defined as the number of distinct colors already present in a vertex's neighborhood. At each step, the algorithm selects the uncolored vertex with the highest saturation degree; ties are broken by choosing the vertex with the largest degree. By continuously adapting the ordering based on partial coloring information, DSatur generally produces higher-quality colorings than fixed-order heuristics.

The randomized greedy algorithm mitigates the ordering sensitivity of greedy coloring by running the greedy process multiple times with different random vertex permutations. Among all runs, the coloring that uses the fewest colors is selected as the final result. In our experiments, the randomized greedy algorithm is executed 20 times for each graph instance.

Finally, we implement a genetic algorithm (GA) as a metaheuristic approach to explore the global solution space more effectively. In our design, each individual represents a complete coloring scheme encoded as an array of color assignments. Unlike standard GA implementations, our approach integrates a repair mechanism that automatically resolves color conflicts after crossover and mutation, ensuring that all individuals represent valid colorings. The fitness function jointly considers both the number of color conflicts and the total number of colors used, guiding the population toward feasible and compact solutions. To further improve convergence, we incorporate elitism and use a greedy coloring solution as an upper bound on the color range, which significantly reduces the search space. Depending on graph size, the population size is set between 30 and 60 individuals, and the algorithm evolves for 40 to 100 generations. The detailed procedure of the proposed graph-aware genetic algorithm is summarized in Algorithm 1.

Algorithm 1 Graph-Aware Genetic Algorithm (GA) for Graph Coloring

```
Require: Graph $G = (V, E)$; population size $P$; generations $T$; mutation rate $p_m$; elite ratio $\rho$
Ensure: Valid coloring $x^\star$ with small number of colors
 1: $K \leftarrow$ GreedyColorCount$(G) + 2$                                ▷ color upper bound
 2: $E \leftarrow \max(1, \lfloor \rho P \rfloor)$
 3: $\mathcal{P} \leftarrow$ InitPopulation$(|V|, K, P)$;   $\mathcal{P} \leftarrow$ RepairAll$(G, \mathcal{P})$
 4: $x^\star \leftarrow \arg\min_{x \in \mathcal{P}}$ Fit$(G, x)$
 5: for $t = 1$ to $T$ do
 6:     $F(x) \leftarrow$ Fit$(G, x)$ for all $x \in \mathcal{P}$              ▷ $F = (conflicts, colors)$
 7:     $x^\star \leftarrow \min(x^\star, \arg\min_{x \in \mathcal{P}} F(x))$      ▷ lexicographic
 8:     $\mathcal{P}' \leftarrow$ best $E$ individuals in $\mathcal{P}$ under $F$              ▷ elitism
 9:     while $|\mathcal{P}'| < P$ do
10:         $(p_1, p_2) \leftarrow$ TournamentSelect$(\mathcal{P}, F)$     ▷ size=3, pick best; repeat twice
11:         $(y_1, y_2) \leftarrow$ Crossover$(p_1, p_2)$                      ▷ single-point
12:         for $y \in \{y_1, y_2\}$ do
13:             $y \leftarrow$ Repair$(G,$ Mutate$(G, y, K, p_m))$
14:             if $|\mathcal{P}'| < P$ then add $y$ to $\mathcal{P}'$
15:         end for
16:     end while
17:     $\mathcal{P} \leftarrow \mathcal{P}'$
18: end for
19: $x^\star \leftarrow$ CompressColors(Repair$(G, x^\star)$)
20: return $x^\star$
    Auxiliary procedures (compact).
21: function Fit$(G, x)$
22:     return $(\#\{(u, v) \in E : x[u] = x[v]\},\ |\{x[u] : u \in V\}|)$
23: end function
24: function Repair$(G, x)$
25:     sequentially fix each conflicted vertex by assigning the smallest color not used by its neighbors
26: end function
27: function Mutate$(G, x, K, p_m)$
28:     for each vertex, with prob. $p_m$, assign the first non-conflicting color in $[0, K-1]$ (if exists)
29: end function
30: function CompressColors$(x)$
31:     remap used colors to $0..k-1$ preserving assignments
32: end function
```

4 Experiments

To comprehensively evaluate algorithmic performance under diverse structural conditions, we consider several classes of graphs, including random, structured, and adversarial instances.

Random graphs are generated using the Erdős–Rényi model $G(n, p)$, where the number of vertices n ranges from 30 to 200 and the edge probability p varies between 0.1 and 0.7. These graphs represent common inputs without special structure and serve as baseline test cases.

To evaluate robustness beyond random instances, we include structured graphs with known theoretical properties. In particular, we consider the Mycielski graph of order 5, which contains no triangles but has a chromatic number of 5. This graph is used to test whether an algorithm can handle cases where local structure provides limited information about global coloring requirements. We also include crown graphs with parameter n, which are bipartite graphs with $2n$ vertices. Although crown graphs have an optimal chromatic number of 2, greedy algorithms may perform poorly on them when vertices are processed in an unfavorable order, making them a well-known adversarial family.

In addition, we generate custom adversarial graphs designed to amplify ordering effects in greedy coloring. These graphs contain many edges between low-index and high-index vertices, while having relatively few edges among high-index vertices. Finally, we consider bipartite-like graphs, which are constructed by first generating a random bipartite graph and then adding a small number of edges within each partition, resulting in chromatic numbers slightly greater than 2.

Algorithm performance is evaluated using the following three metrics:

- the number of colors used as a measure of solution quality.
- execution time measured using a high-precision performance counter.
- validity of the coloring, verified by checking all edges for color conflicts.

All experiments use fixed random seeds to ensure reproducibility.

For fairness, all heuristic algorithms are executed once per instance, while randomized greedy is repeated 20 times and reports the best coloring found. For the proposed GA, we follow the same parameter ranges described in Algorithm 1: population size $P \in [30, 60]$, generations $T \in [40, 100]$, and elitism ratio ρ fixed across comparable settings. Each GA run reports the best feasible individual under the lexicographic fitness objective.

5 Discussion

5.1 Random Graph Performance

The first experiment compares all six algorithms on a random graph with 80 vertices and edge probability 0.35, containing 1107 edges. The results show significant variation in solution quality. Greedy produced a 14-color solution, while DSatur achieved 11 colors, representing a 21% improvement. LDF and Welsh-Powell both used 13 colors, and the Genetic algorithm found a 12-color solution. Regarding execution time, Greedy completed in 0.09ms, DSatur required 1.33ms, and the Genetic algorithm took 548.90ms. This demonstrates that DSatur provides the best balance between solution quality and computational cost.

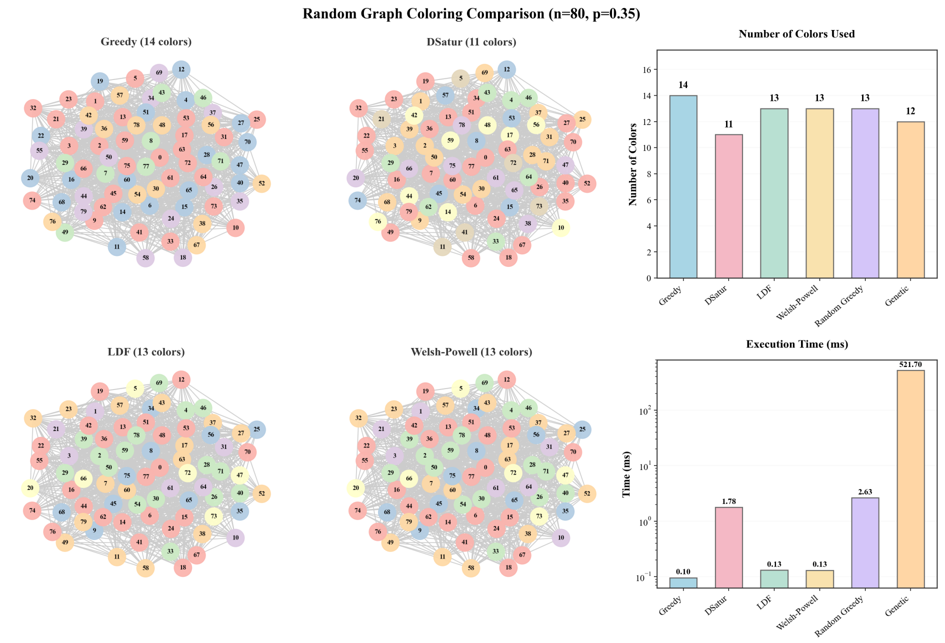

Figure 1: Random Graph Coloring Comparison

5.2 Challenging Graph Structures

The second experiment evaluates performance on graphs with known properties. On the Mycielski-5 graph with 23 vertices and 71 edges, all algorithms achieved the optimal chromatic number of 5. The Crown-8 graph with 16 vertices dramatically exposed the weakness of ordering-dependent algorithms. Greedy used 8 colors on this bipartite graph, while DSatur and the Genetic algorithm both achieved the optimal 2 colors. This represents a 75% improvement by using appropriate heuristics. On the Adversarial-30 graph, Greedy used 9 colors compared to 8 for DSatur. The Bipartite-like graph with 30 vertices required 4 colors from all algorithms.

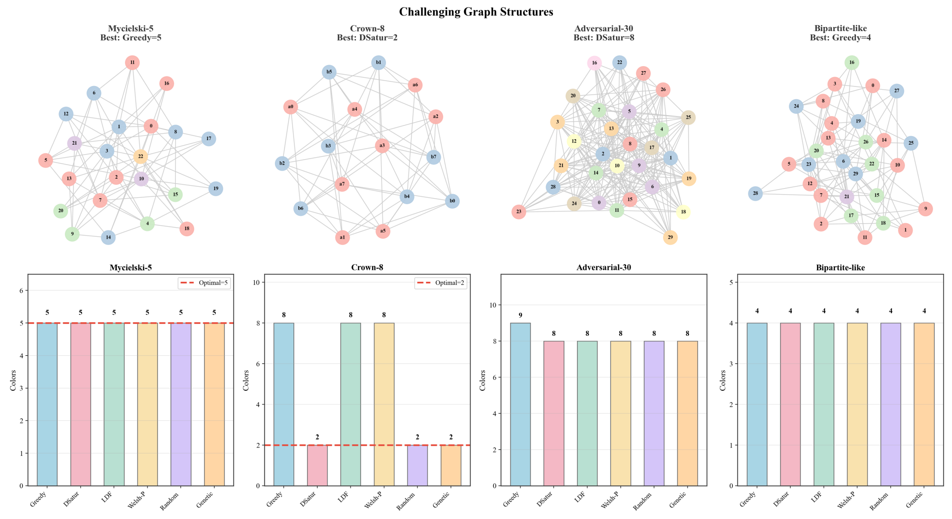

Figure 2: Challenging Graph Structures

5.3 Scalability Analysis

The third experiment examines how algorithms scale with graph size. Testing on random graphs with n ranging from 30 to 200 and fixed density 0.3 reveals clear patterns. DSatur consistently uses fewer colors than other algorithms across all sizes, with the gap widening for larger graphs. At n=200, Greedy requires 23 colors while DSatur needs only 19. Execution time grows polynomially for all algorithms, but the Genetic algorithm shows significantly higher overhead, requiring 331.9ms at n=200 compared to 13.2ms for DSatur. The simple Greedy and LDF algorithms remain fastest at under 0.5ms even for n=200.

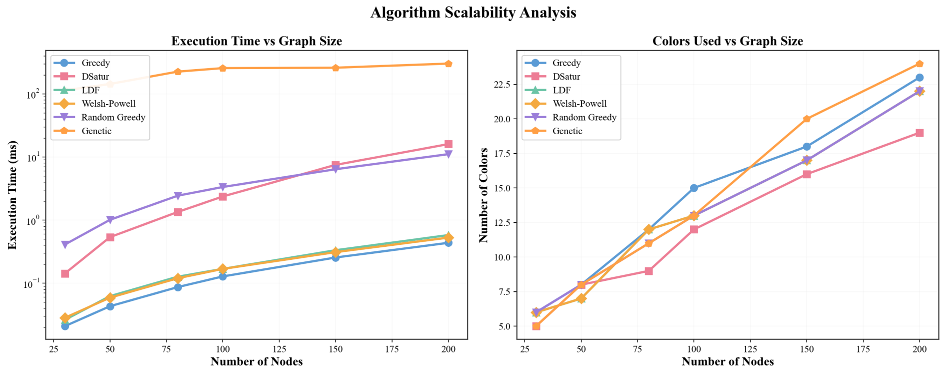

Figure 3: Algorithm Scalability Analysis

5.4 Effect of Graph Density

The fourth experiment varies edge probability from 0.1 to 0.7 on graphs with 60 vertices. As expected, all algorithms require more colors as density increases since denser graphs have higher chromatic numbers. DSatur maintains its advantage across all densities, using 4 colors at p=0.1 compared to 6 for Greedy, and 18 colors at p=0.7 compared to 23 for Greedy. The relative improvement of DSatur over Greedy remains approximately 20-30% regardless of density.

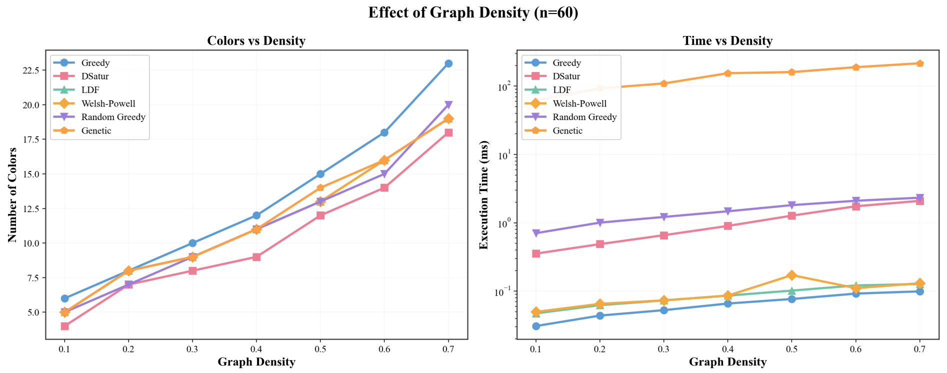

Figure 4: Effect of Graph Density

5.5 Crown Graph Analysis

The fifth experiment provides detailed analysis of the Crown graph, which represents a worst-case input for naive greedy algorithms. Testing Crown graphs with n from 4 to 10 shows that Greedy always uses exactly n colors while the optimal is 2. The improvement achieved by DSatur and Genetic algorithms increases linearly with n, reaching 80% at n=10. This experiment definitively demonstrates that simple greedy algorithms can perform arbitrarily poorly on certain graph structures, and that saturation-based heuristics effectively avoid these pitfalls.

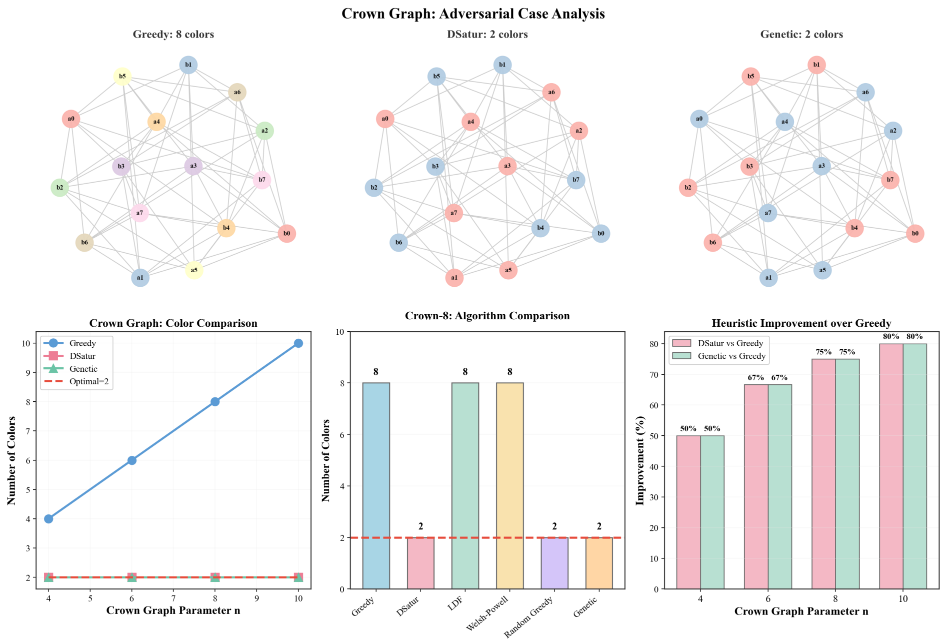

Figure 5: Crown Graph Adversarial Case Analysis

5.6 Statistical Analysis

The sixth experiment runs 30 independent trials on random graphs with n=60 and p=0.35 to assess result consistency. DSatur achieves mean 9.37 colors with standard deviation 0.55, compared to Greedy's mean 11.17 with standard deviation 0.58. The minimum colors found by DSatur across all trials is 8, while Greedy never found solutions better than 10. Mean execution times show DSatur at 0.68ms and Genetic at 82.12ms, confirming that DSatur provides excellent quality at reasonable computational cost. Together with the results from random and structured graphs, this analysis confirms that DSATUR not only improves average solution quality but also reduces variance across runs.

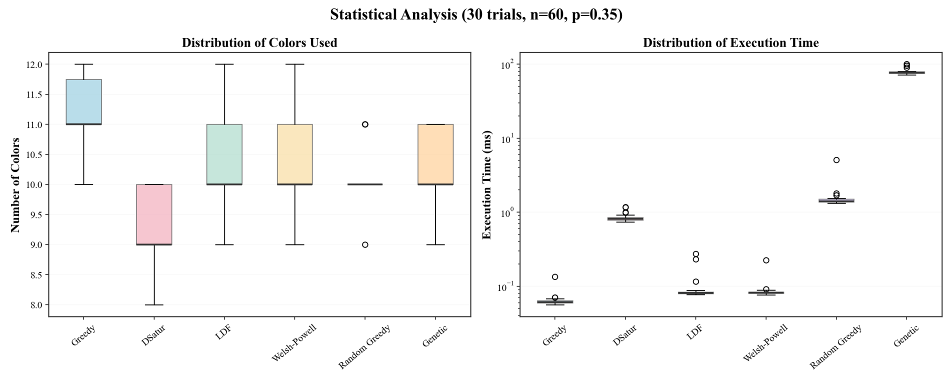

Figure 6: Statistical Analysis

Overall, the experimental results reveal consistent patterns across different graph types, sizes, and densities. In particular, heuristic strategies that adapt vertex ordering dynamically exhibit significantly more stable performance than static greedy approaches. The following subsections analyze these behaviors in detail.

Takeaway for the proposed GA:

Although the proposed graph-aware GA is computationally more expensive than classical heuristics, it provides a flexible framework for integrating structural knowledge (e.g., greedy bounding and feasibility repair) into global search. The GA consistently produces valid colorings and matches optimal solutions on adversarial families such as crown graphs, indicating that feasibility-preserving evolutionary search can be effective when ordering-based heuristics fail. These observations motivate future improvements through hybridization with DSatur and local search, as well as parallel implementations to reduce runtime overhead.

6 Future Work

While the proposed graph coloring framework demonstrates the effectiveness of combining classical heuristics with evolutionary search, several important directions remain open for further improvement.

A key limitation of the current genetic algorithm lies in its mutation operator, which is largely random and does not explicitly exploit graph structure or search history. Although random mutation encourages exploration, it may also introduce unnecessary perturbations that slow convergence and reduce solution stability. Future work could address this limitation by incorporating heuristic- or learning-guided mutation strategies that adaptively bias the search toward promising regions.

One natural extension is the deeper hybridization of the genetic algorithm with classical heuristic methods such as DSatur. Rather than using DSatur only as a standalone algorithm, DSatur-generated colorings could be injected into the initial population or periodically reintroduced during evolution. Hybrid genetic approaches that combine evolutionary search with heuristic guidance have been shown to significantly improve convergence speed and solution quality in graph coloring problems [7]. Building on this idea, heuristic-aware population management could further strengthen feasibility preservation and diversity control.

Another promising direction is the integration of local search techniques within the genetic algorithm framework. Previous studies have demonstrated that local refinement strategies, including hill climbing and Kempe chain-based recoloring, can effectively resolve conflicts and escape poor local optima in graph coloring [8]. Incorporating such local search operators after crossover or mutation steps could combine global exploration with local exploitation, leading to more robust solutions.

Beyond heuristic guidance, future work could explore learning-based mechanisms to guide evolutionary operators. For example, reinforcement learning or lightweight machine learning models could be used to learn mutation or recoloring policies based on graph features such as vertex degree, saturation level, or historical conflict frequency. Such approaches may be viewed as a natural extension of heuristic-guided evolutionary strategies, allowing the algorithm to adapt its search behavior to different graph structures over time.

From a computational perspective, the relatively high runtime cost of genetic algorithms motivates the exploration of parallel and distributed implementations. Many components of the proposed algorithm, including fitness evaluation, mutation, and repair operations, are inherently parallelizable. Prior work has shown that parallel genetic algorithms can achieve substantial speedups without sacrificing solution quality [2].

Finally, evaluating the proposed methods on established benchmark datasets, such as the DIMACS graph coloring instances [11], would enable direct comparison with state-of-the-art heuristic and metaheuristic approaches. Such benchmark-driven evaluation would further validate the scalability and generality of the proposed framework.

7 Conclusions

This paper empirically investigates heuristic and metaheuristic strategies for graph coloring, focusing on understanding how heuristic algorithm design affects the quality, robustness, and efficiency of solutions. Results show that while naive greedy algorithms are computationally inexpensive, they are highly sensitive to vertex order and can perform poorly on certain structured graphs. In contrast, heuristics employing dynamic, constraint-aware ordering—especially saturation-based methods such as DSATUR—consistently achieve higher quality and more stable coloring results across various graph types, sizes, and densities.

While DSatur provides the strongest quality–time balance in our experiments, the proposed graph-aware GA highlights a complementary advantage: it offers a general and extensible optimization framework where domain-specific mechanisms (greedy bounding, feasibility repair, and lexicographic objectives) can be systematically integrated. Even without additional local search, the GA reliably maintains feasibility and achieves optimal results on adversarial families such as crown graphs. These results suggest that feasibility-preserving evolutionary search is a promising direction for graph coloring, especially when combined with hybrid heuristics and efficiency-oriented implementations.

Future research could explore hybrid strategies, combining the efficiency of saturation-based heuristics with local search optimization, and explore parallel implementations of metaheuristics to improve scalability. Evaluating these methods on standardized benchmark suites will help further verify their generality and practical applicability.

References

[1] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[2] Erick Cantu-Paz. *Efficient and accurate parallel genetic algorithms*, volume 1. Springer Science & Business Media, 2000.

[3] Gregory J Chaitin. Register allocation & spilling via graph coloring. *ACM Sigplan Notices*, 17(6):98–101, 1982.

[4] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015.

[5] Agoston E Eiben, Jan K Van Der Hauw, and Jano I van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[6] Charles Fleurent and Jacques A Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of operations research*, 63(3):437–461, 1996.

[7] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 3(4):379–397, 1999.

[8] Philippe Galinier and Alain Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006.

[9] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[10] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. John Wiley & Sons, 2011.

[11] David S Johnson and Michael A Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[12] Pablo San Segundo. A new dsatur-based algorithm for exact vertex coloring. *Computers & Operations Research*, 39(7):1724–1733, 2012.

[13] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.