Comparative Analysis of Algorithms for the Graph Coloring Problem: Minimizing Chromatic Number

Muhammad Youshay
my07103@st.habib.edu.pk
Habib University

Nabila Zahra
nz07162@st.habib.edu.pk
Habib University

Simal Anjum
sa07716@st.habib.edu.pk
Habib University

ABSTRACT

The graph coloring problem, a challenge of assigning colors to graph vertices such that adjacent vertices do not share the same color. This problem is a fundamental challenge in computer science with far-reaching implications for scheduling, resource management, and network optimization. In this report, we explore the strengths and limitations of various approaches to solving this problem, including brute force, greedy algorithms, and Ant Colony Optimization. Through computational testing and algorithmic analysis, we aim to uncover the trade-offs between speed and solution quality offered by each method, shedding light on the effective strategies for tackling this complex challenge.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; *Design and analysis of algorithms.*

KEYWORDS

graph coloring, combinatorial optimization, NP-complete problems, brute force, greedy algorithms, ant colony optimization, computational complexity

ACM Reference Format:
Muhammad Youshay, Nabila Zahra, and Simal Anjum. 2024. Comparative Analysis of Algorithms for the Graph Coloring Problem: Minimizing Chromatic Number. In *Proceedings of Algorithms Design and Analysis Course (Habib University).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Background and Importance

Graph coloring is a pivotal concept in computer science and combinatorial optimization that requires colors assigned to the vertices of a graph to not coincide at adjacent vertices. The importance of this problem is significant with numerous applications such as timetable scheduling, register allocation in compilers, and the management of frequencies in cellular networks, where minimal color usage is often essential for cost reduction and efficiency[1].

1.2 Challenge of Chromatic Number

The crux of the matter lies in determining the chromatic number of a graph, which represents the minimum number of colors needed to color the vertices in such a way that no two adjacent vertices share the same color. This seemingly simple task harbors complexity, as discovering the chromatic number is often a daunting challenge.

One significant aspect contributing to the difficulty is the NP-completeness of the problem. NP-completeness implies that while it is easy to verify a solution, finding one efficiently (i.e., in polynomial time) is considered unlikely, especially as the size of the graph increases. This classification places graph coloring among a class of problems for which no known polynomial-time algorithm exists that guarantees finding the optimal solution for all instances.

As graphs grow in size and complexity, the computational demands of determining their chromatic number become increasingly burdensome. Even modest-sized graphs can pose significant computational challenges, let alone large-scale or real-world graphs encountered in practical applications.

1.3 Approaches

Given the complexity of the problem, different algorithmic strategies have been developed, ranging from exact solutions to heuristic and metaheuristic methods. This paper delves into three distinct methodologies:

1. **Brute Force:** An exhaustive strategy that guarantees the discovery of an optimal solution but is feasible only for very small graph instances due to its exponential time complexity.
2. **Greedy Algorithms:** It sequentially assigns the minimum feasible color to each vertex. This way it is able to provide a quick solution but the solution may not always be optimal.
3. **Ant Colony Optimization (ACO):** A metaheuristic algorithm inspired by the food foraging behavior of ants, ACO offers a balance between the quality of solution and computational effort taken to find the solution. This makes it suitable for larger and complex graphs.

1.4 Objective and Scope

The objective is to compare the chosen algorithms on different graphs to assess their practicality and effectiveness. The analysis of these algorithms will help identify which algorithms are more efficient and under different circumstances. This will offer insights beneficial for both theoretical research and practical applications.

2 ALGORITHMS AND DESIGN TECHNIQUES

In this section, the algorithmic approaches of Brute Force, Greedy Algorithms, and Ant Colony Optimization are explored in tackling the graph coloring problem. Each algorithm is analysed to understand how it addresses the challenge of coloring a graph and achieving minimal chromatic numbers. The depth of analysis includes conceptual explanations, algorithmic steps (shown in pseudo code), theoretical complexities and specifics of their application.

2.1 Brute Force Approach

Concept Overview.

- The Brute Force method explores every possible combination of colors for the vertices of a graph and evaluates them until a valid solution with fewest colors is found. This method guarantees the discovery of the optimal solution.

Algorithmic Detail.

- **Initialization**: All possible color combinations are found according to the the number of vertices.
- **Validation**: Each coloring is tested to make sure it is a valid coloring. A coloring is valid if no two adjacent vertices share the same color.
- **Result Extraction**: The solution using the least number of colors is selected as the optimal output.

```
1. Initialize a list of potential colors.
2. Generate all combinations of these
colors for the vertices.
3. For each color combination:
   a. Check if it forms a valid coloring
   of the graph.
   b. If valid and uses fewer colors than
   previously found solutions, update the
   optimal solution record.
4. Return the color combination found to
use the least colors.
```

Time Complexity Analysis.

- **Overview**
  The brute force graph coloring algorithm explores every possible vertex color assignment, checking each for validity by ensuring no adjacent vertices share the same color.
- **Key Points**
  Vertices and Colors:
  - Let n be the number of vertices.
  - The algorithm iterates through all color configurations from 1 to n colors.
- **Color Configurations:**
  - For each color count, configurations are num_colors^n. Total configurations over all color counts is:

$$\sum_{k=1}^{n} k^n$$

- **Validation Complexity:**
  - Each configuration is checked in $O(n^2)$ due to pairwise comparisons between vertices and their neighbors.
- **Total Time Complexity**
  Combining configuration generation and validation, the time complexity sums to:

$$\sum_{k=1}^{n} k^n \times O(n^2)$$

This sums up to $O(n^n \times n^2)$ and it reflects exponential computational demands typical in NP-complete problems like graph coloring, emphasizing the practical limitations of the brute force approach for large graphs.

2.2 Greedy Algorithm

Concept Overview.

- The Greedy Algorithm for graph coloring takes a local optimization approach, where colors are assigned to vertices one by one based on the minimum available color that hasn't been used by its adjacent vertices.

Algorithmic Details.

- **Sequential Coloring**: Each vertex is considered in sequence, and the lowest feasible color is assigned.
- **Color Update**: When a vertex is colored, the selection for subsequent vertices considers the already used colors by neighboring vertices to avoid conflicts.

```
1. Initialize an empty color assignment
for all vertices.
2. For each vertex:
   a. Gather colors used by its
   adjacent vertices.
   b. Assign the lowest color number not
   used by its neighbors.
3. Continue until all vertices are
colored.
4. Return the color assignments and the
highest color number used.
```

Time Complexity Analysis.

- **Overview**
  The greedy graph coloring algorithm assigns colors to vertices by checking the lowest possible unused color among its adjacent vertices. It progresses sequentially through all the vertices, ensuring each is colored such that no two connected vertices share the same color.
- **Key Points**
  Vertices and Initialization:
  - Let n be the number of vertices and m be the number of edges.
  - Each vertex is examined once, and a color is assigned based on adjacent vertices' colors.

  Operation on Each Vertex:
  - The algorithm seeks the smallest unused color that none of the adjacent vertices have.

  Adjacent Vertices Check:

  - For each vertex, iterate through its neighbors (adjacent vertices) to check their colors and exclude these from the possible colors for the current vertex.
  - Worst-case scenario for each vertex checking Δ (maximum degree of the graph) neighbors.
- **Computational Operations**
  - **Color Selection Complexity:**
    Identifying if a color is used by any neighbor requires looking up colors in a dictionary and possibly removing these from a set of available colors. The complexity for this is $O(\Delta)$, where Δ is the maximum vertex degree in the graph.
  - **Assigning Colors:**
    Assigning the smallest missing color from a list or set of size n (number of vertices) can be done in $O(1)$ using set operations when implemented efficiently.
- **Total Time Complexity**
  While iterating over each vertex, and for each vertex checking up to Δ neighbors, the time complexity for the whole coloring process involves: $O(n\Delta)$.
  This takes into account the worst-case scenario where every vertex requires checking all its neighbors to determine the correct color.

2.3 Ant Colony Optimization (ACO)

Concept Overview.

- Inspired by the behavior of ants in finding paths from their colony to food sources, ACO applies a similar heuristic to graph coloring by allowing a colony of artificial 'ants' to explore colorings, updating their paths based on the quality of solutions and pheromone trails[2].

Algorithmic Details.

- **Pheromone Initialization**: Begin with a uniform distribution of pheromones.
- **Solution Building**: Ants construct solutions iteratively, choosing vertices and colors based on pheromone strength (indicative of prior success) and heuristic information (like vertex degree).
- **Pheromone Update**: After each iteration, pheromones are updated to reflect the success of different colorings, reinforcing pathways to better solutions.

```
1. Initialize pheromones for all possible
vertex-color pairs.
2. For a set number of iterations:
   a. Each ant constructs a coloring of
   the graph.
   b. Evaluate the quality of the graph
   coloring.
   c. Update pheromones based on the
   quality of solutions found.
3. At the end of iterations, select the
best coloring found across all ants.
```

Time Complexity Analysis.

- **Overview**
  Ant Colony Optimization (ACO) utilizes a combination of heuristic and probabilistic techniques to explore feasible solutions for graph coloring. The complexity of this approach is influenced by multiple factors involving the behavior and interaction of artificial ants, pheromone updating, and heuristic guidance.
- **Key Factors Influencing Complexity:**
  - **Number of Ants (m):** Represents the number of ants in the model which simultaneously construct solutions.
  - **Number of Vertices (n):** Each ant attempts to color all vertices.
  - **Maximum Iterations (t):** The total number of iterations or generations of ant colonies allowed in the algorithm.
- **Computational Analysis:**
  - Each ant constructs a solution involving n vertices per iteration. At each step, an ant chooses the next vertex to color by considering the pheromone level and heuristic value. The complexity of this selection can be estimated as $O(n)$ per vertex.
  - After constructing their paths or solutions during each iteration, the pheromone level on the paths used by the ants needs to be updated. Pheromone update typically has a complexity proportional to the number of edges, estimated as $O(m \cdot n^2)$ for dense graphs.
- **Total Time Complexity:**
  From initialization, solution construction, and pheromone updates, the overall time complexity of ACO for graph coloring can be represented as:
  $$O(t \cdot m \cdot (n + n^2)) = O(t \cdot m \cdot n^2)$$
  Here, t represents the number of iterations or generations, m the number of ants, and n^2 the consideration of all vertex pairs for pheromone updates in dense scenarios.

3 DESIGN OF EXPERIMENTS

3.1 Experimental Goals

The primary objective of our study was to evaluate the performance and scalability of Brute Force, Greedy, and Ant Colony Optimization across graphs of various sizes. Our aim was to assess these algorithms under controlled conditions to provide an insight into their usage particularly when used to solve practical problems.

3.2 Experimental Setup

Programming Language: Python 3.12 was chosen to implement all algorithms.

Machine Specifications: Experiments were conducted on a machine equipped with an Intel Core i5-63000U CPU @ 2.4GHz, 8 GB RAM, running Windows 11.

Data Set Description: To comprehensively evaluate the performance and computational complexity of the three algorithms, we employed four different datasets of varying sizes and characteristics:

(1) **smallest_data.col**: A self-generated dataset containing 10 nodes and 15 edges, representing a relatively small graph instance.
(2) **smaller_data.col**: Another self-generated dataset with 20 nodes and 30 edges, slightly larger than the previous dataset.
(3) **queen11_11.col**: This dataset, obtained from the Stanford GraphBase, consists of 121 nodes and 3,960 edges, representing a moderately sized graph with a higher density of edges.[3]
(4) **le450_15b.col**: The Leighton graph, a larger dataset with 450 nodes and 8,169 edges, providing a more challenging instance for the algorithms to test their scalability and performance.[4]

3.3 Methodology

We executed each algorithm on the described datasets, measuring execution time and memory usage. Due to the extensive computational demands, especially for more complex datasets and the Brute Force algorithm, each algorithm was run three times per dataset. This number of repetitions was a balance between obtaining reliable data and managing practical constraints such as extended execution times. We calculated the average performance metrics from these runs to ensure that our results weren't only reliable but also representative of typical performance.

3.4 Constraints and Limitations

The experiments were limited due to computational constraints. We were unable to test the larger datasets extensively across all algorithms, particularly the Brute Force approach to its exponential time complexity.

4 RESULTS AND DISCUSSION

4.1 Algorithmic Analysis

The three algorithms discussed in this study exhibit unique characteristics and behaviors for solving the graph coloring problem. The brute force approach, although guarantees an optimal solution has exponential time complexity, which makes it impractical for all but the smallest graph instances. As the number of vertices increases, the computational demands of this design technique grows exponentially, quickly becoming uncontrollable.

In contrast, the greedy algorithm offers a more efficient approach, with a time complexity of $O(n\Delta)$, where n is the number of vertices and Δ is the maximum degree of the graph. This algorithm's simplicity and speed make it a applicable option for larger graphs, although it does not guarantee optimal solution[1]. The quality of the solution is very dependent on the order in which vertices are processed and the availability of colors for each vertex.

The Ant Colony Optimization (ACO) method finds the middle ground between solution quality and computational effort. While its time complexity is of $O(t \cdot m \cdot n^2)$, where t is the number of iterations and m is the number of ants, which is higher than that of the greedy algorithm, it has the potential to find better solutions through its iterative and probabilistic nature. ACO's ability to explore larger solution space and reinforce promising paths through pheromone updates can lead to more efficient colorings, particularly for complex graph structures[5].

When dealing with different types of data sets, the performance and effectiveness of these algorithms can vary. For small, sparse graphs with low vertex degrees, the brute force approach may be feasible, as the search space remains small. However, as the graph density and size increase, the exponential complexity of the brute force method quickly makes it impractical.

The greedy algorithm's performance is majorly dependent on the order in which vertices are processed and the distribution of vertex degrees within the graph. In graphs with a high degree of regularity or specific structural patterns, the greedy approach may yield reasonably good solutions. However, in more complex or irregular graphs, its greedy nature can lead to suboptimal colorings, as early decisions may limit the availability of colors for subsequent vertices.

ACO's strength lies in its ability to adapt and explore the solution space more effectively, making it well-suited for larger and more intricate graph structures. By leveraging the collective behavior of multiple ants and the reinforcement of successful paths through pheromone updates, ACO can navigate complex graph topologies and potentially discover more efficient colorings.

4.2 Computational Complexity

To further analyze, we present a computational analysis of running each of these algorithms on different data sets of varying sizes to test their performance.

Table 1: Computational Complexity Comparison (Seconds)

Algorithm	D-1	D-2	D-3	D-4
BF	0.0208	16.2699	-	-
GA	0.0	0.0013	0.0050	0.0203
ACO	1.4223	1.4700	189.5601	1663.47

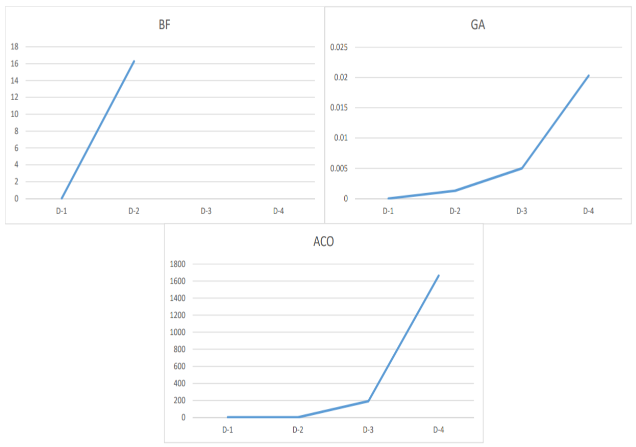
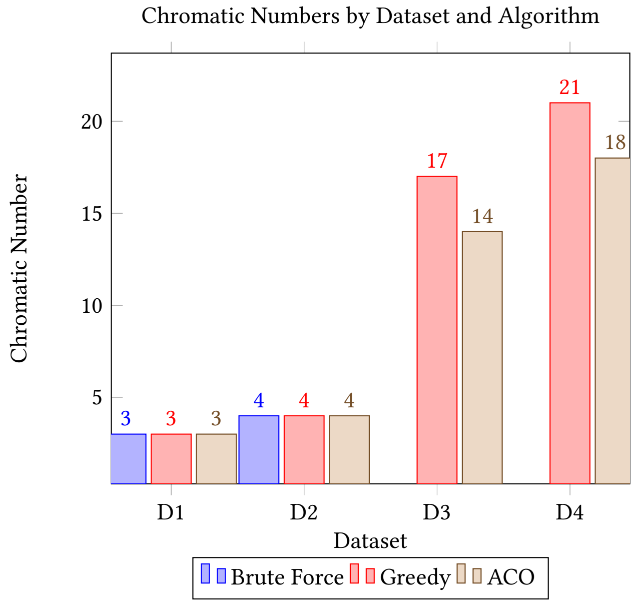


Figure 1: Execution time comparison of Algorithms on different datasets

By including these diverse datasets, ranging from small, self-generated instances to larger, benchmark graphs, we aimed to comprehensively assess the algorithms' behavior under various graph complexities and sizes. This approach allowed us to identify potential strengths, weaknesses, and practical limitations of each algorithm in tackling the graph coloring problem across a spectrum of scenarios.

4.3 Results

The bar chart below shows the difference in chromatic numbers in the datasets between algorithms. The optimized result was given by ACO for large datasets but for smaller datasets, each algorithm gave the same result in terms of chromatic number.



The Graphs generated for each of the algorithm on different data sets along with their Chromatic Number are given below.

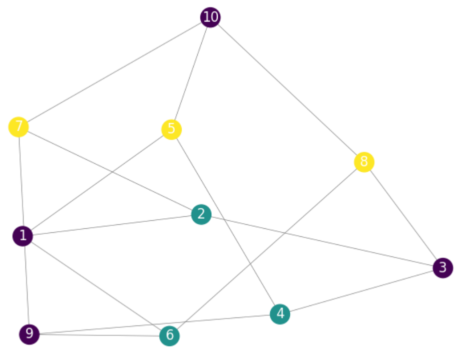


Figure 2: Algorithm - BF, DataSet - D1, Chromatic Number - 3

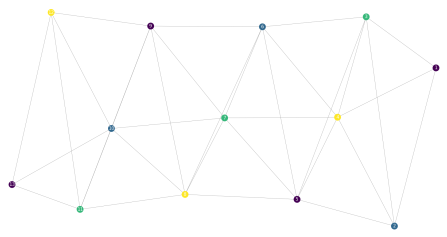


Figure 3: Algorithm - BF, DataSet - D2, Chromatic Number - 4

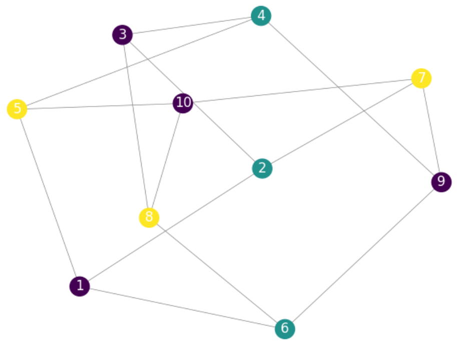


Figure 4: Algorithm - GA, DataSet - D1, Chromatic Number - 3

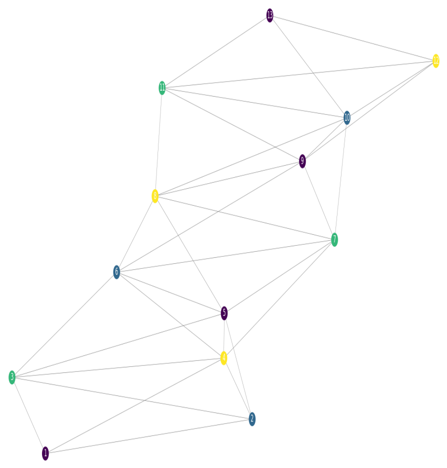


Figure 5: Algorithm - GA, DataSet - D2, Chromatic Number - 4

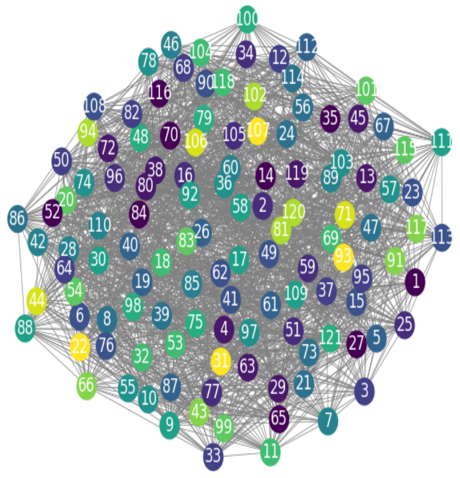


Figure 6: Algorithm - GA, DataSet - D3, Chromatic Number - 17

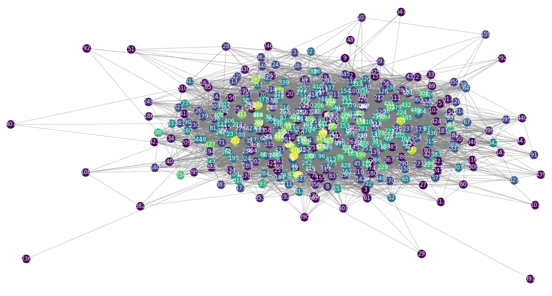


Figure 7: Algorithm - GA, DataSet - D4, Chromatic Number - 21

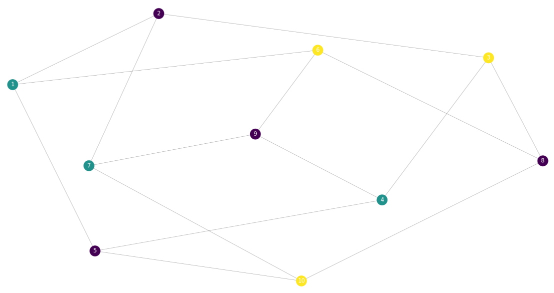


Figure 8: Algorithm - ACO DataSet - D1 Chromatic Number - 3

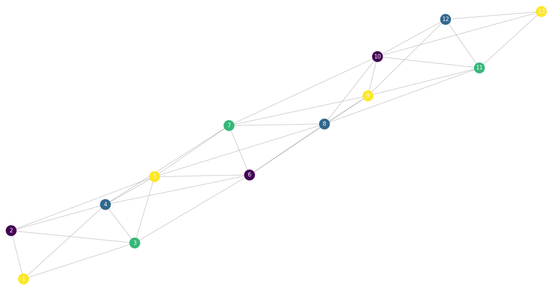


Figure 9: Algorithm - ACO DataSet - D2 Chromatic Number - 4

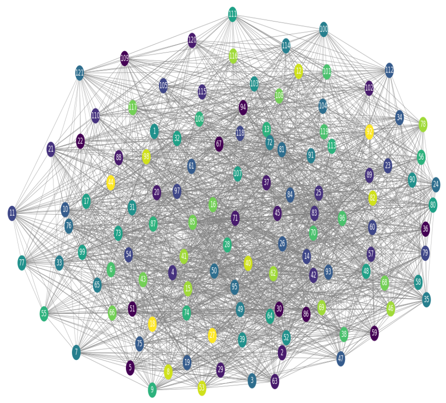


Figure 10: Algorithm - ACO DataSet - D3 Chromatic Number - 14

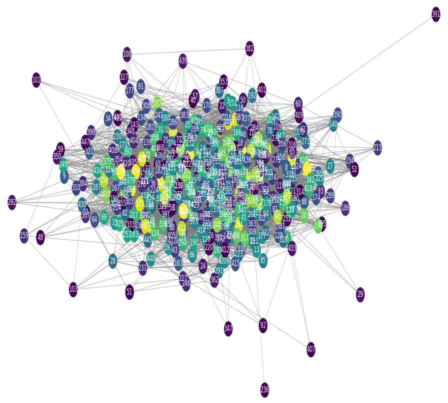


Figure 11: Algorithm - ACO DataSet - D4 Chromatic Number - 18

5 ANALYSIS

5.1 Algorithmic Complexity Analysis

The graph coloring problem engages several algorithmic strategies, which vary dramatically in terms of performance and efficacy, particularly across different scales and densities of graphs. Our focus has been on analyzing the complexity and performance of three primary algorithms—Brute Force, Greedy Algorithm, and Ant Colony Optimization (ACO).

Brute Force Approach

Computational Complexity: Brute Force suffers from exponential growth in computational time as graph size increases. This pattern was starkly visible. When we attempted to apply this method to larger datasets, it could not complete within reasonable time frames, making it impractical for everyday computing hardware on larger sets.

Performance Analysis: On smaller datasets (e.g., *smallest_data.col*), Brute Force managed to process within a limited time, yielding optimal solutions. However, its utility diminishes rapidly with size increment, highlighted by its inability to compute for datasets like *queen11_11.col* and *le450_15b.col.*

Greedy Algorithm

Computational Complexity: Exhibiting polynomial time complexity, typically around $O(n\Delta)$, where n is the number of vertices and Δ their maximum degree, the Greedy Algorithm provides a notably more efficient alternative to Brute Force.

Performance Trends: While it computes significantly faster across all data scales and successfully completes even on large datasets, the Greedy Algorithm often yields solutions that are not optimally minimized. This efficiency versus optimality trade-off is pronounced on complex topologies or higher density graphs.

Ant Colony Optimization (ACO)
Computational Complexity: Although ACO does not reach the exponential time costs of Brute Force, it incurs considerable computational overheads, particularly under complex problem sets ($O(t \cdot m \cdot n^2)$, with t denoting iterations, m the ant count, and n the number of vertices). Its execution time is intermediate, more than Greedy but less than Brute Force's infeasible durations on larger datasets.

Optimality and Performance: ACO stands out by producing more optimized solutions compared to the Greedy Algorithm and is capable of completing computations within feasible times for large datasets, albeit slower than Greedy. The optimization quality justifies the additional time expense, especially for applications demanding high accuracy in solution quality.

5.2 Reasoning Behind Performance Trends

Brute Force: The primary limitation here is the sheer number of possible configurations it evaluates (k^n for each color count k up to n), leading to exponential growth in computations. This method tests each combination thoroughly, ensuring optimality but at impractical computation costs for large graphs.

Greedy Algorithm: Its efficiency stems from a straightforward approach—assigning the first available color, which drastically reduces the solution space it explores. The algorithm's quick decisions, however, can lead to suboptimal coloring due to early commitment to certain colors which might restrict options later in the sequence.

Ant Colony Optimization (ACO): ACO's performance is buoyed by its probabilistic exploration and pheromone signaling mechanism, which help in navigating towards promising regions of the solution space more effectively than Greedy's fixed path. This method cleverly balances exploration and exploitation, adapting to find better solutions over iterations but at the cost of a higher computational overhead compared to Greedy.

5.3 Applicability Across Different Problem Sizes and Types

Small to Medium Scale Problems: The Greedy Algorithm and Brute Force are more applicable here. For very small datasets where computational resources are not that important, Brute Force could guarantee an optimal solution. The Greedy Algorithm, while sacrificing optimality for efficiency, remains viable for moderately sized instances.

Large Scale Problems: Ant Colony Optimization (ACO) emerges as a more suitable choice due to its ability to handle larger datasets with acceptable computational overhead. While it may not guarantee optimality as Brute Force does, it strikes a balance between solution quality and computational feasibility, outperforming Greedy in terms of solution quality on complex topologies or dense graphs.

6 CONCLUSION

This study has undertaken a comprehensive analysis of three algorithmic approaches to solving the graph coloring problem—Brute Force, Greedy Algorithms, and Ant Colony Optimization (ACO). Our investigation revolved around the central challenge of optimizing the chromatic number, a critical parameter in applications ranging from schedule optimization to resource allocation in networks.

6.1 Key Findings

6.1.1 ***Complexity vs. Practicality of Brute Force.*** The Brute Force method, while guaranteeing the discovery of an optimal solution through exhaustive search, proves impractical for large instances due to its exponential time complexity. It is only feasible for graphs with a very limited number of vertices.

6.1.2 ***Efficiency of Greedy Algorithms.*** Greedy algorithms provide a significant advantage in terms of computational speed. However, this comes at the cost of sometimes failing to find the optimal solution. The simplicity and speed of the Greedy approach make it suitable for moderately sized problems where approximate solutions are acceptable.

6.1.3 ***Balance Offered by Ant Colony Optimization.*** ACO, inspired by the foraging behavior of ants, introduces a metaheuristic that balances between the depth of search and computational efficiency. By iteratively improving upon solutions and using pheromone trails to guide search processes, ACO is often able to find good solutions that are near-optimal, particularly useful for complex and larger graphs.

6.1.4 ***Performance Across Different Graph Types.*** The performance of each algorithm varied across different datasets, from small to large. The Brute Force approach was limited to smaller graphs, Greedy Algorithms scaled relatively well to medium-sized graphs, and ACO demonstrated robustness in handling larger, more complex graphs efficiently.

6.2 Theoretical and Practical Implications

6.2.1 ***Theoretical Implications.*** The study enriches theoretical understanding by delineating the operational thresholds and efficiency boundaries of each algorithm. It provides a clear characterization of each method's strengths and weaknesses in the context of NP-complete problems like graph coloring.

6.2.2 ***Practical Implications.*** Practically, insights derived from this research can guide the selection of an appropriate algorithm based on the size and complexity of the problem at hand. For instance, in real-world applications where time is a constraint and exact solutions are not critical, Greedy Algorithms or ACO might be preferred over Brute Force.

In conclusion, while no single algorithm universally outperforms the others across all scenarios, each has unique advantages that make it suitable for specific types of graph coloring problems. This diversity in algorithmic strategy underscores the importance of context in the selection of computational methods for solving complex problems in computer science and operations research.

CODE

Link to Github Repository.

REFERENCES

[1] Velin Kralev and Radoslava Kraleva. 2023. An analysis between different algorithms for the graph vertex coloring problem. Int. J. Electr. Comput. Eng. 13, 3 (June 2023), 2972-2980. DOI:https://doi.org/10.11591/ijece.v13i3.pp2972-2980

[2] M. Bessedik, R. Laib, A. Boulmerka, and H. Drias, "Ant Colony System for Graph Coloring Problem," in Proc. International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), (2005), pp. 786-791, DOI: 10.1109/CIMCA.2005.1631360

[3] Knuth, D. E. (n.d.). The Stanford graphbase: A platform for combinatorial computing. Knuth: The Stanford GraphBase. https://www-cs-faculty.stanford.edu/ knuth/sgb.html

[4] Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. Journal of Research of the National Bureau of Standards, 84(6), 489. https://doi.org/10.6028/jres.084.024

[5] Costa, D. and Hertz, A., 1997. Ants can colour graphs. Journal of the Operational Research Society, 48(3), (March 1997),pp.295-305. DOI:10.1057/palgrave.jors.2600357