Research on Graph Coloring Algorithms

Through the Implementation of Graph Coloring and Comparison Software

Peyton Boggs

MATH 332 Course Project, Fall 2024

www.graphcoloring.xyz

Abstract

For my course project, I wanted to dive deeper into the differences between graph coloring algorithms. Coming from a Computer Science background, I had already worked with the greedy coloring algorithm that we learned in class, but I knew that greedy algorithms, while fast, might not always produce the best results. In my research, I found four interesting algorithms to test: Greedy Coloring, DSatur Coloring, Recursive Largest First Coloring, and Genetic Coloring. In order to visualize these colorings and further research the differences between them, I developed an application where a user could create a graph with any vertices and edges, and then see the results of the four algorithms side by side. The application determines how many colors were used by each algorithms, and users can compare algorithms for several prebuilt graphs that highlight differences between them. After researching the algorithms and testing different graphs that highlight the algorithm's strengths and weaknesses, I determined that the Greedy Coloring algorithm is the best in efficiency and largely determines the optimal coloring for a graph, and the DSatur algorithm picks up in the few cases that the Greedy Coloring algorithm falls off.

1 Introduction

Graph coloring is a fundamental concept in discrete mathematics and computer science that has captivated researchers for decades. The problem involves assigning colors to the vertices of a graph such that no two adjacent vertices share the same color. The challenge lies not only in finding a valid coloring but in determining the minimum number of colors required, known as the chromatic number of the graph.

The study of graph coloring has far-reaching implications, extending beyond theoretical interest to a wide array of practical applications. From optimizing schedules and allocating resources to designing efficient algorithms and solving complex network problems, graph coloring techniques have proven invaluable across various domains.

As we delve deeper into the world of graph coloring, we encounter a rich landscape of algorithms, each with its own strengths and limitations. From the straightforward greedy approach to more sophisticated methods like genetic algorithms, a diverse toolkit has been developed to tackle this NP-complete problem. These algorithms highlight the delicate balance between computational efficiency and solution quality.

In this paper, I will explore and analyze four of the most intriguing graph coloring algorithms: Greedy Coloring, DSatur Coloring, Recursive Largest First Coloring, and Genetic Coloring. I have developed a graph visualization application to further research the differences between them. By comparing their efficiency, performance, and applicability to different types of graphs, I aim to provide a comprehensive overview of the different coloring algorithms.

2 Greedy Coloring Algorithm

The Greedy Coloring Algorithm is a simple yet effective approach to graph coloring. It assigns colors to vertices sequentially, using the lowest available color for each vertex while ensuring no adjacent vertices share the same color. While the Greedy Algorithm may not always yield the optimal solution, its simplicity and efficiency make it a popular choice for many graph coloring applications.

2.1 How It Works

The input is an uncolored graph with a set of vertices and edges. For each vertex in the graph, examine the colors of its adjacent vertices, and assign the lowest-numbered color not used by any adjacent vertex. Repeat this for every vertex (in the order of vertices given) until all vertices are colored.

2.2 Efficiency and Complexity

The Greedy Algorithm is known for its computational efficiency. It's time complexity is $O(V+E)$, where V is the number of vertices and E is the number of edges. It's space complexity is $O(V)$. This makes it suitable for large-scale applications and graphs with many vertices.

2.3 Advantages and Limitations

The advantages of the greedy coloring algorithm are that it is simple to implement and understand, computationally efficient for large graphs, and guarantees an upper bound of $(\Delta(G)+1$ colors. The disadvantages of the greedy coloring algorithm is that it does not always produce the optimal (minimum) number of colors, and that the result can depend on the order in which vertices are processed.

3 DSatur Coloring Algorithm

The DSatur (Degree of Saturation) Coloring Algorithm makes a modification to the Greedy algorithm to produce better results. Instead of selecting the vertices in order, it dynamically selects the next vertex to color based on the number of differently colored adjacent vertices. This heuristic often leads to better colorings than the simple Greedy approach, especially for certain types of graphs.

3.1 How It Works

The input is an uncolored graph with a set of vertices and edges. Initialize the saturation degree of all vertices to zero. In each step, select the uncolored vertex with the highest saturation degree (breaking ties by choosing the vertex with the highest degree in the uncolored subgraph). Assign this vertex the lowest-numbered color not used by its adjacent vertices. Update the saturation degrees of the uncolored neighbors. Repeat this process until all vertices are colored.

3.2 Efficiency and Complexity

The DSatur Algorithm has a time complexity of $O(V^2)$, where V is the number of vertices. This is because for each vertex, we need to update the saturation degrees of its neighbors. The space complexity is $O(V)$, as we need to store the color and saturation degree for each vertex. While slightly more complex than the Greedy algorithm, DSatur remains efficient for most practical applications.

3.3 Advantages and Limitations

The advantages of the DSatur coloring algorithm are that it often produces better colorings than the Greedy algorithm, especially for graphs with a wide range of vertex degrees, and it adapts its coloring strategy based on the partially colored graph. The main disadvantage is that it's slightly more complex to implement than the Greedy algorithm, and it may require more computational time for very large graphs due to the need to maintain and update saturation degrees.

4 Recursive Largest First Coloring Algorithm

The Recursive Largest First (RLF) Coloring Algorithm selects the largest independent set of vertices in the graph to be a color class. Once the first color class has been completed, it takes the uncolored graph and recursively calls itself with that graph, generating one complete color class at a time until all vertices have been colored.

4.1 How It Works

The input is an uncolored graph. The algorithm starts by selecting the vertex with the highest degree and assigns it the first color. It then recursively builds a maximal independent set of vertices that can receive this color. Once no more vertices can be added to this color class, the algorithm removes these colored vertices from consideration and repeats the process with the remaining uncolored subgraph, using a new color. This process continues until all vertices are colored.

4.2 Efficiency and Complexity

The RLF Algorithm has a time complexity of $O(V^3)$, where V is the number of vertices. This higher complexity compared to Greedy or DSatur algorithms is due to the recursive nature of building independent sets. The space complexity is $O(V^2)$, as it may need to maintain adjacency information for the subgraphs. While more computationally intensive, RLF can be effective for graphs of moderate size where coloring quality is paramount.

4.3 Advantages and Limitations

The advantages of the RLF coloring algorithm are that it often produces high-quality colorings, particularly for dense graphs, and it tends to use fewer colors than greedy approaches. It's also less sensitive to the initial vertex ordering. The main limitations are its higher time and space complexity, making it less suitable for very large graphs, and the increased implementation complexity compared to simpler algorithms.

5 Genetic Coloring Algorithm

The Genetic Coloring Algorithm leverages the principles of genetic algorithms to address the graph coloring problem. While the previous three algorithms have been deterministic, the Genetic Coloring Algorithm will have different results every time it runs. While this generally leads to poorer results, it is an interesting algorithm to study.

5.1 How It Works

Modeling itself after the Theory of Evolution, the Genetic Coloring Algorithm first starts with a population of completely random colorings called "Generation 0". Then, all of these colorings are tested for fitness; a coloring is deemed more fit if it uses less colors, and is deemed unfit if it is not a proper coloring. Then, the top half of the population is selected, parent colorings are crossed over to produce children colorings, and mutations occur to generate "Generation 1". This process is repeated for many generations until there are no significant gains in fitness over a span of generations.

5.2 Efficiency and Complexity

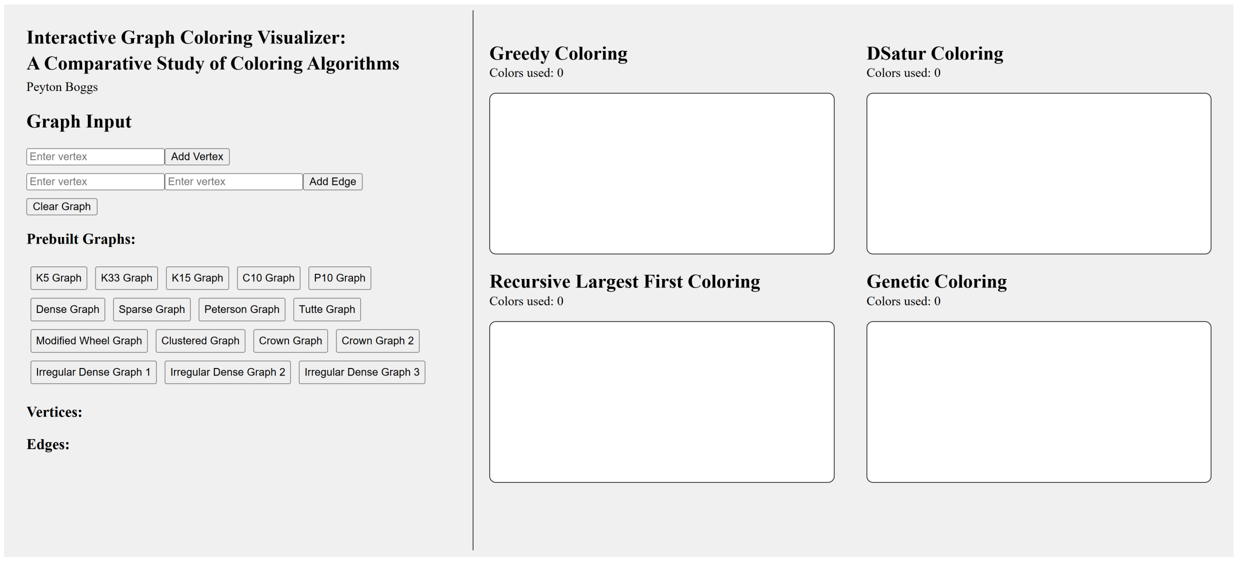
The efficiency of the Genetic Coloring Algorithm can vary based on several factors, including population size, mutation rate, and selection pressure. Generally, its time complexity is difficult to define precisely due to its stochastic nature; however, it is often polynomial in practice for many instances. The space complexity is primarily determined by the size of the population and the representation of chromosomes.

5.3 Advantages and Limitations

The advantages of the Genetic Coloring Algorithm is that it is flexible, it better explores the solution space, and it can be made parallel to suit large-scale problems. The disadvantages of the Genetic Coloring Algorithm are that there is no guarantee of optimality, the performance heavily relies on parameter settings, and it is computationally intensive.

6 Application

While researching these four algorithms and reading about the theoretical differences between them was interesting, my background in Computer Science and Software Development knew that there was a better way to do this research. All of these algorithms can be programmed to be run by computers, and there are ways to visualize graphs through programs, so the need to develop such an application became clear. Using React with JavaScript, I developed the Interactive Graph Coloring Visualizer to more directly compare the four algorithms.



Interactive Graph Coloring Visualizer:
A Comparative Study of Coloring Algorithms
Peyton Boggs

Graph Input

Prebuilt Graphs:

Vertices:

Edges:

Greedy Coloring
Colors used: 0

DSatur Coloring
Colors used: 0

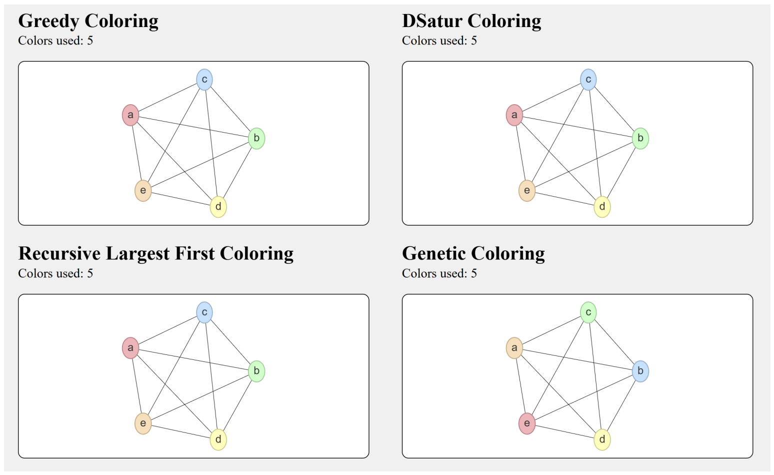
Recursive Largest First Coloring
Colors used: 0

Genetic Coloring
Colors used: 0

On the left side, users can build graphs to research. Through the ”Graph Input” section, a user can add vertices and edges to the graph to generate any graph. Of course, inputting vertices and edges one by one can be tedious, so users can also press any of the buttons under ”Prebuilt Graphs” to automatically construct and build off of one of those.

On the right side, users can see their graphs colored by each of the four algorithms. The program takes the list of vertices and edges given by the user as input and puts them through the four algorithms, which each produces a coloring of the graph. Then, using the prebuilt React component 'react-graph-vis', the application displays the graph four times - one with each coloring. Users can see how their graphs were colored by the four algorithms, as well as how many colors each algorithm used.

7 Usage


Greedy Coloring
Colors used: 5

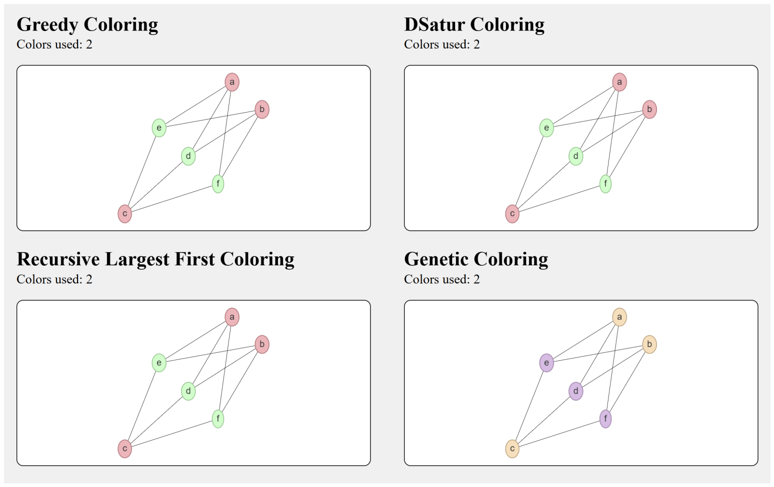
DSatur Coloring
Colors used: 5

Recursive Largest First Coloring
Colors used: 5

Genetic Coloring
Colors used: 5

7.1 K_5 Graph

The K_5 graph, the complete graph with 5 vertices, is one of the easiest graphs to color since all vertices are connected to each other. This forces every vertex to have a different color. All four algorithms recognized this and gave the proper coloring.


Greedy Coloring
Colors used: 2

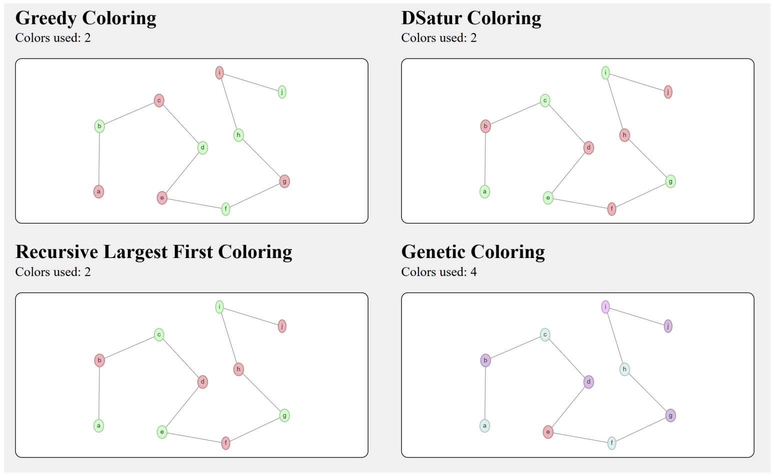
DSatur Coloring
Colors used: 2

Recursive Largest First Coloring
Colors used: 2

Genetic Coloring
Colors used: 2

7.2 $K_{3,3}$ Graph

The $K_{3,3}$ graph, the bipartite graph with two sets of three independent vertices, is well known to have a 2-coloring. Once again, these algorithms are able to generate this. However, it is interesting that the Genetic Algorithm, over multiple iterations, sometimes generates a 3-coloring and almost always uses different colors than the deterministic algorithms; this demonstrates its random nature.


Greedy Coloring
Colors used: 2

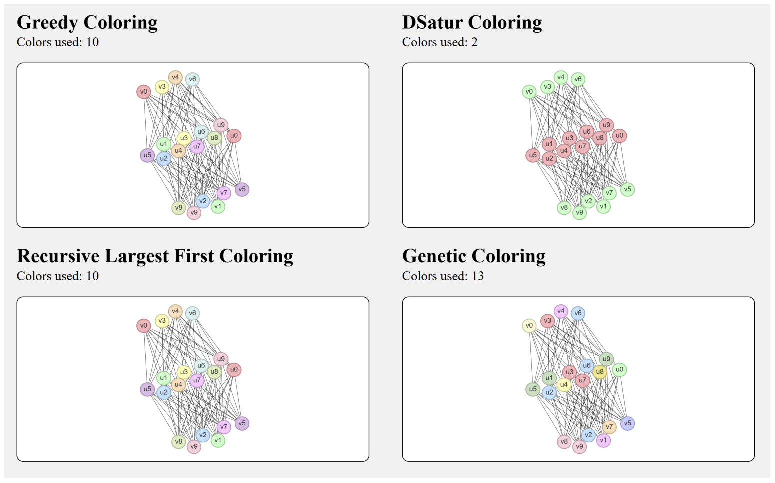
DSatur Coloring
Colors used: 2

Recursive Largest First Coloring
Colors used: 2

Genetic Coloring
Colors used: 4

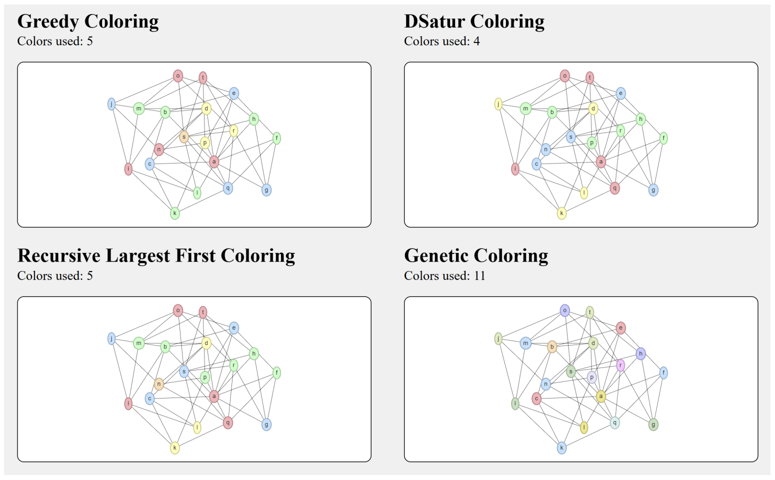
7.3 P_{10} Graph

The P_{10} graph, the path with 10 vertices, is another easy graph to color. Optimally, every other vertex has the same color, and so it can be colored with two colors. While the deterministic algorithms recognized this, the Genetic Coloring Algorithm did not produce a coloring with less than four colors.



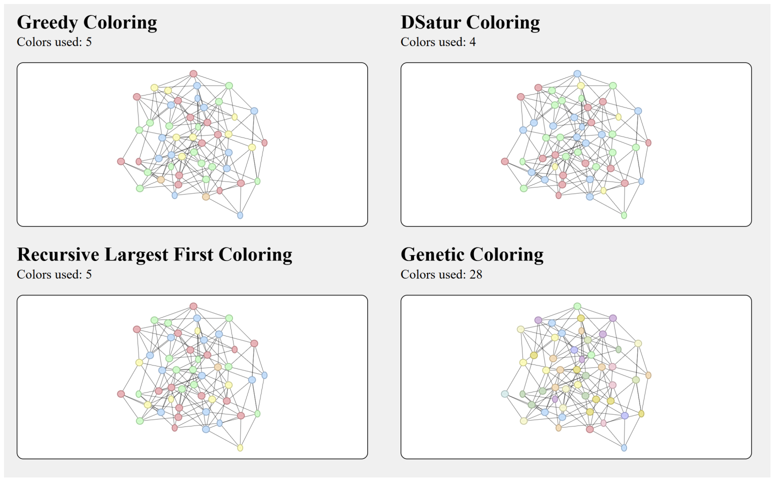
7.4 Crown Graph

The Crown Graph is a variation of the bipartite graph where there are two independent sets completely connected to each other, except each vertex in one set is not connected to it's opposite vertex in the other. The Crown Graph is famously tough for greedy coloring algorithms, and it shows here. While DSatur identifies the optimal 2-coloring, the Greedy and RLF algorithms generate a 10-coloring.



7.5 Irregular Graph 1

Since these algorithms generally work well with nice, symmetric graphs, we will next look at dense, irregular graphs to expose their weaknesses. This graph has 20 vertices, each with 4-6 connections. The Greedy and RLF algorithms come up with a 5-coloring, while DSatur is able to find a 4-coloring.



7.6 Irregular Graph 2

Finally, this irregular graph contains 50 vertices and 146 edges. Again, it is shown that the Greedy and RLF Algorithms produce a 5-coloring that would feel acceptable as the optimal coloring for such a dense graph, if not for the DSatur Algorithm producing a 4-coloring. The Genetic Coloring, on the other hand, was unable to evolve anything better than a 28-coloring.

8 Conclusion

After researching and mathematically analyzing each of the algorithms and using the software to generate and examine numerous graphs and colorings generated by the algorithms, I have concluded the following findings:

First, the Greedy Algorithm is the most popular for a reason: it's easy to comprehend, it's fast, and for the majority of use cases, it's as accurate as you need a coloring algorithm to be. Aside from the Crown Graph edge case, the Greedy Algorithm always gave the optimal (or near optimal) coloring. It's simplicity as a heuristic makes it easy to implement and it's low time complexity make it a great coloring algorithm.

Better, though, is the DSatur Algorithm. With just a little tweaking to the Greedy Algorithm, the added heuristic of selecting the vertex with the highest saturation before each iteration makes it significantly better. It can optimally color the Crown Graph, and can get the edge on the Greedy Algorithm in dense, irregular graphs - repeatedly finding a coloring with one less color. While it is more computationally more complex than the Greedy Algorithm, the difference is not very significant, especially compared to other algorithms.

Next, the Recursive Largest First Algorithm, while interesting in concept, hardly has anything special to offer when compared to Greedy and DSatur. It does to a fine job at coloring graphs, never worse than Greedy does, but it also doesn't excel in any area. Particularly, the RLF Algorithm is extremely complex for the colorings it generates. Taking more time and space while producing worse results than the DSatur and Greedy algorithms, the RLF Algorithm is suboptimal.

Finally, the Genetic Algorithm, abysmal results aside, really was interesting to look into and implement. It has by far the worst performance and by far the worst results, but the idea of taking Darwin's Theory of Evolution and applying it to Graph Theory was certainly a fun exercise. More could be investigated with parameter tuning like population size, number of generations, and the fitness calculation, but nothing that was tried did much better than these results. While it didn't go anywhere, it was an exciting algorithm to do research on.

For the majority of applications, then, this research recommends the Greedy and DSatur Algorithms to be used as the best available coloring algorithms. However, it must be said that the best fit algorithm depends on both the needs of the user and the implementation of the algorithm. There are many tweaks that could be made to the implementations in my application that could change the correctness and performance of each algorithm, and whether an application prioritizes space, time, or precision depends on the use case. While the Recursive Largest First and Genetic Algorithms surely have their place in Graph Theory, the Greedy Algorithm and its enhanced variant DSatur Algorithm consistently demonstrated the best performance throughout this study.