EC504 Project

Project Title: Graph Coloring/Exam Scheduling Project

Group Members:

Gorkem Yar - U31957122

Nikolay Popov

Ajay Krishna

Saurabh Singh

Sally Shin

Professor:
Richard Brower

Date: 05.05.2023

1. Abstract

This report presents research on graph coloring problem using five heuristic algorithms: Dsatur, SDL, RLF, greedy, and Welsh-Powell. The algorithms were implemented using C++ and tested on a variety of graphs. The graphs are generated randomly based on the number of nodes and edges inputted by the user. The graph representations were illustrated via the Python NetworkX library. The results were then compared and analyzed. The report provides a detailed description of each algorithm, as well as a discussion of their relative strengths and weaknesses. In addition, a GitHub repository is available that includes the algorithms implemented, and instructions on how to use the algorithms with custom test cases. The findings of this study can be used to inform future research in the field of graph theory and algorithm design.

2. Introduction

Graph coloring is one of the fundamental problems in computer science. It is used in various applications, including scheduling, register allocation, and network optimization. Graph coloring aims to assign a color to each vertex so that no adjacent vertices have the same color (Brelaz, 1979). Also, it does not randomly assigns different colors for each node, the main goal of the algorithm is to determine the least number of colors. This problem is an NP-complete problem, which means finding an optimal solution is unmanageable for large graphs (Brelaz, 1979).

In this project, we examine five different heuristic algorithms for graph coloring: Dsatur, SDL, RLF, greedy, and Welsh-Powell. These algorithms have greedy heuristics to determine a good solution. We implemented these algorithms in C++ and tested them on various graphs, ranging from small to large, dense to sparse. The graphs were generated randomly based on the number of nodes and edges inputted by the user. We used Python to represent the graphs. We aim to compare and analyze the performance of these algorithms on different graphs, and to identify their strengths and weaknesses.

The rest of this report is organized as follows. In Section 3, we provide the algorithms and their heuristics. In Section 4, we present our experimental results and analyze the performance of the algorithms. Finally, in Section 5, we will show how to run the algorithms and test them.

3. Proposed Heuristics

Dsatur: This algorithm colors the nodes according to their maximum saturation degree. The saturation degree of a node is the count of adjacent and colored nodes. This algorithm picks the node with the maximum saturation degree and then colors it with the least available color (Brelaz, 1979). The least available color is the color that is assigned to the minimum number of nodes and is not adjacent to the current node.

RLF: This algorithm recursively picks a color that it tries to assign this color in such a way that the maximum number of edges can be removed. In each recursive iteration, the algorithm picks a different color then it sorts the remaining nodes according to their degree (Leighton, 1979). Then the algorithm picks as many nodes as it can. The basic approach is removing the most number of edges in a single iteration. When an iteration completes, this algorithm removes the picked nodes and edges from the original graph and then recursively continues.

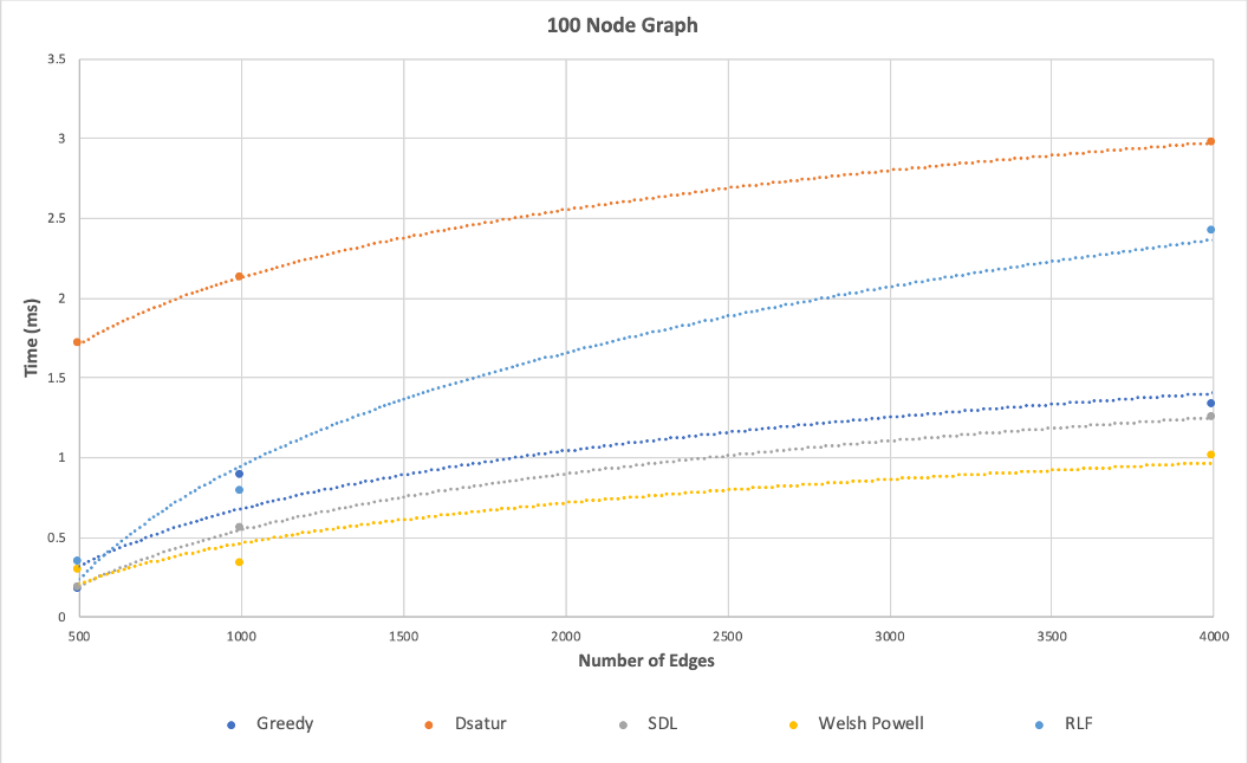
Welsh Powell: This algorithm sorts the nodes according to their degree and then iteratively walkthrough the nodes. For each node, it checks whether it is colored yet. If it is not colored, the algorithm picks a random color and with this color, it tries to color the remaining nodes.

Greedy and SDL: These two algorithms are quite like each other both of them sort the nodes according to their degrees. The difference is Greedy is in descending order SDL is in increasing order. They iterate in the nodes. In each iteration, if the node is not

colored they are trying to find an available color, if there is no available color they generate a random color.

4. Sample Results & Discussion

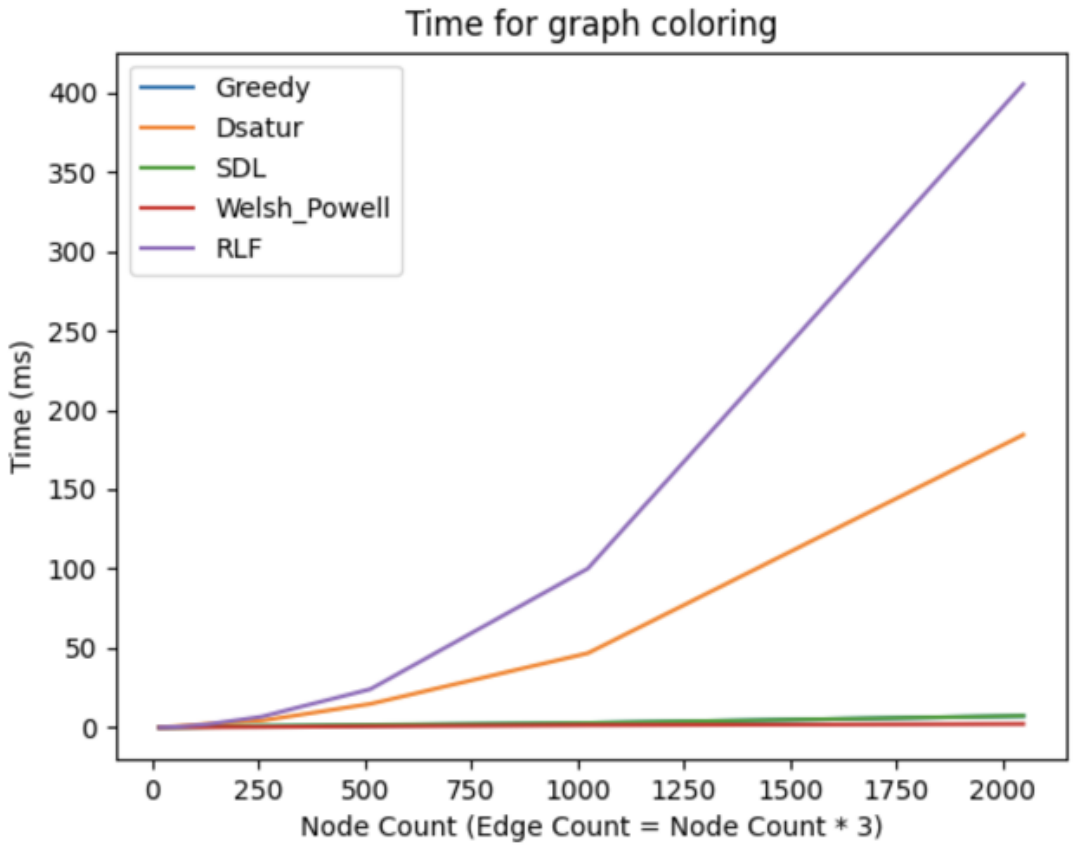
A. Examination with a fixed number of nodes



In this sample, the number of nodes is fixed at 100, and the number of edges varied between 500 to 4000. The figure shows how much time is needed for each algorithm to process 100 nodes with the corresponding number of edges. This figure demonstrates that for the 100 nodes graphs, the Welsh Powell algorithm takes the least time; whereas, the Dsatur algorithm takes the most.

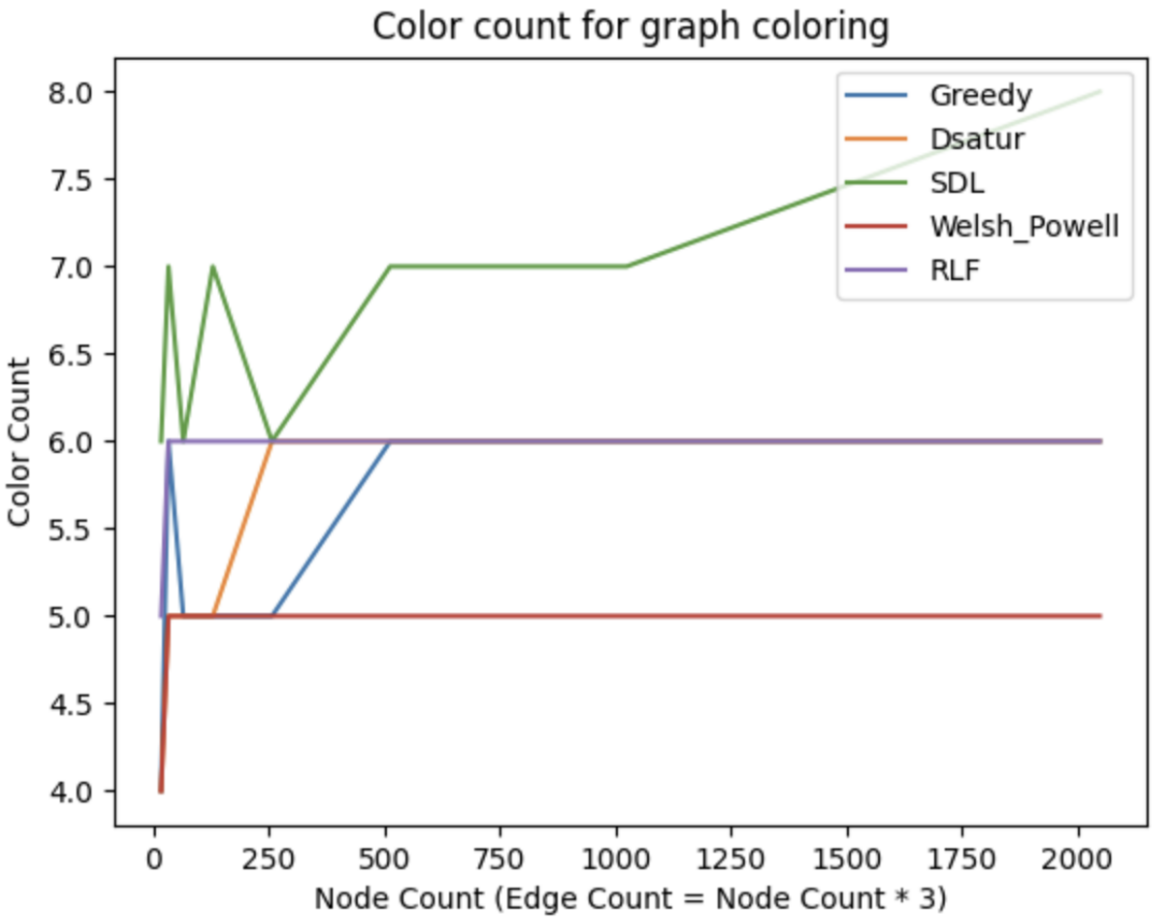
B. Examination of Sparse Graphs

In this context, we determine the edge count of the sparse graphs as a 3 * node count, As a result, one can safely assume that the average edge count per node is 3.



The figure shows how much time is needed for each algorithm to process the corresponding number of nodes and edges (3*number of nodes). For the sparse graphs, the Welsh Powell, SDL, and Greedy algorithms take less time than the Dsatur and RLF algorithms.

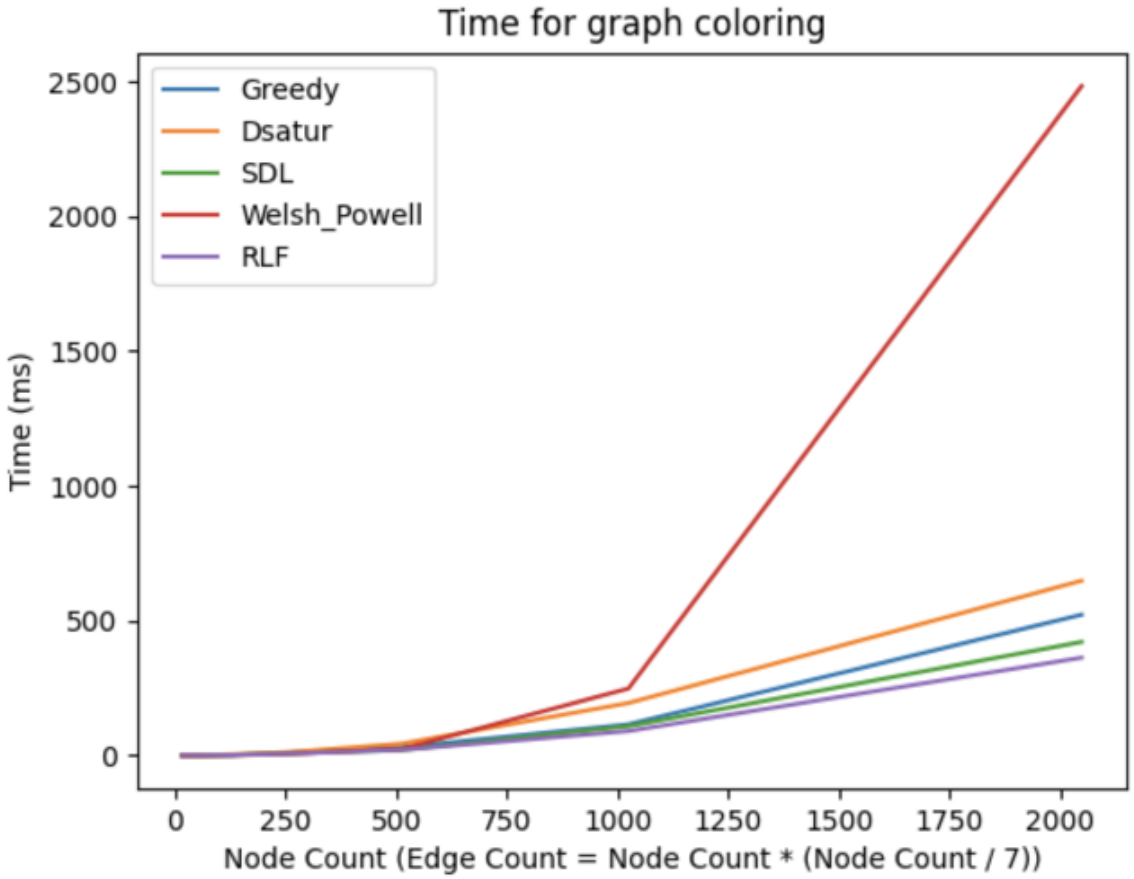
One should also examine how many colors are used by each of these algorithms.



This figure shows how many colors are used for each graph with the corresponding nodes and edges. In terms of color count, the Welsh Powell algorithm finds the best results, in contrast to the SDL algorithm which finds the worst results.

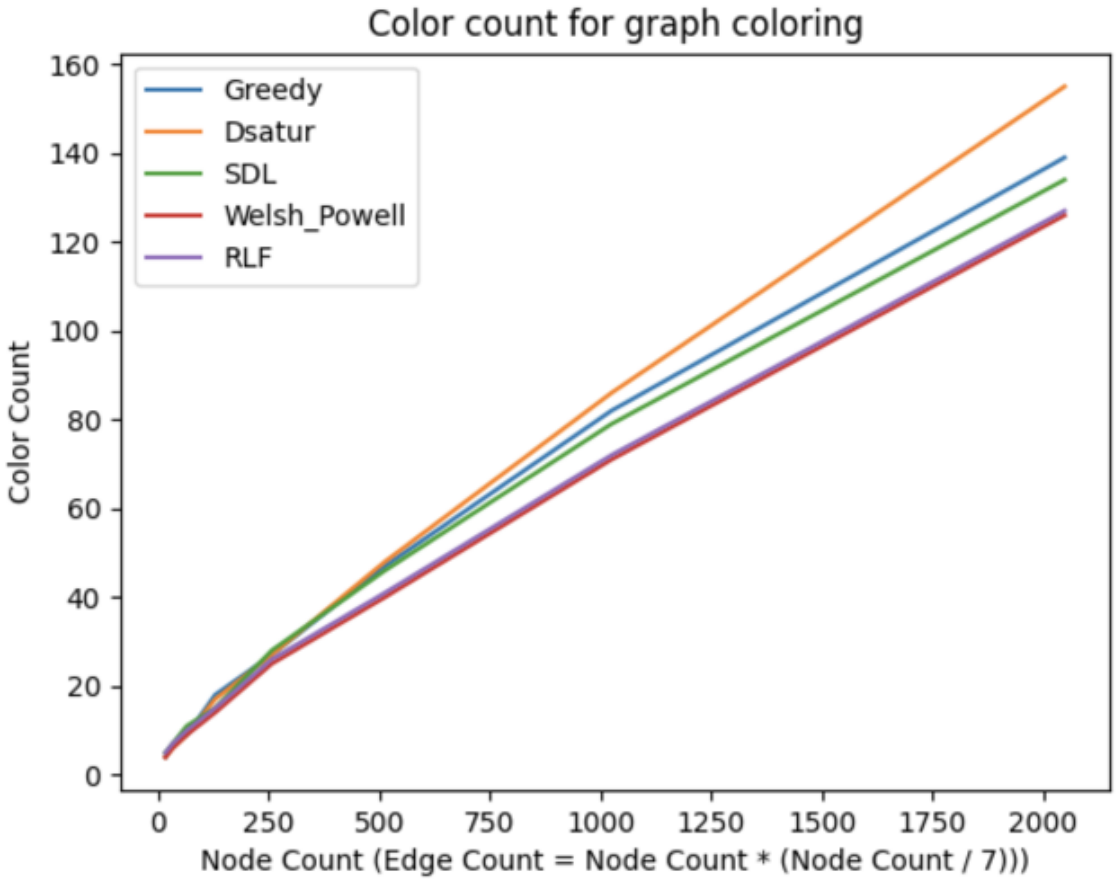
C. Examination of Dense Graphs

In this context, we determine the edge count of the sparse graphs as a node_count * (node count / 7).



The figure shows how much time is needed for each algorithm to process the corresponding number of nodes and edges (node_count * node_count / 7).

As can be seen from the figure, the Welsh Powell algorithm takes the most time. This stems from the design of implementation since it is a dense graph the complexity of Welsh Powell increased. We used an adjacency list representation inside of the graph which increases the time complexity of some helper functions which are used in the Welsh Powell algorithm. Apart from this, algorithms take the same amount of time to run.



This figure illustrates the color count needed for the dense graphs with the corresponding number of nodes, out of the 5 algorithms RLF and Welsh Powell algorithms performed best.

Overall Welsh Powell algorithm generally surpasses the other algorithms both in time and the least number of colors. One can argue that it performed best in our implementation.

5. Running the Code

Install it with the following command:

git clone https://github.com/gorkemyar/GraphColoring.git

There are several ways to run this project they are listed below.

a. Run the "**make**" command to create an executable file. Then run the program with the following command "**./main**". If you run the algorithm this way, you will run it with custom values which are 30 nodes and 100 edges. The resulting graph can be found in the "**results**" directory named "**graph.json**".
b. If you want to visualize the graph, you can use the "**./runall**" command. This will automatically run the make and ./main commands and then visualize the final graph as a png file in the path "results/result.png". As stated above, this will also run with default inputs which are 30 nodes and 100 edges.
c. To run with custom values after running the "**make**" command, you can use "**./main <node_count> <edge_count> <algorithm>**". This will also create the graph as a JSON file in the "results" directory.
d. To run with custom values and visualization use "**./runall <node_count> <edge_count> <algorithm>**" which will create an image of the graph in the path "results/result.png"

<node_count> and <edge_count> are integer values. For the <algorithm> you can use the following options: "g" for Greedy "l" for LDO "r" for RLF "d" for DSATUR "w" for Welsh Powell. An example command for running with custom values:

./main 20 50 r

Or

./runall 20 50 r

After the" runall" command the result.png file should have a coloring such that no adjacent vertices have the same color.

6. References

- Brélaz, D. (1979). New methods to color the vertices of a graph. *Commun. ACM, 22*, 251-256.
- Leighton F. T. (1979). A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of research of the National Bureau of Standards (1977)*, *84*(6), 489–506. https://doi.org/10.6028/jres.084.024
- Arifin, S., Muktyas, I. B., & Mandei, J. M. (2022, May). Graph coloring program for variation of exam scheduling modeling at Binus University based on Welsh and Powell algorithm. In *Journal of Physics: Conference Series* (Vol. 2279, No. 1, p. 012005). IOP Publishing.