CMSC 501: Advanced Algorithms

Spring 2018

Super Graph Coloring

**Rostam Panjshiri**

Instructor: Thang Dinh

Virginia Commonwealth University

College of Engineering

April 30th, 2018

1. The source code for the project is attached in the submission – and can also be found here on my GitHub: <https://github.com/fluffykitty/Super-Graph-Coloring>.
2. The heuristic algorithm used to approximate this optimal Graph Coloring project was a degree-based ordering algorithm where we prioritize the next node to color based off saturation degree. The saturation degree of a node is the number of colored nodes that are adjacent to it. No adjacent vertices may be the same color, but finding the minimal covering is an NP-complete problem. As such, for very large graphs this algorithm may not prove to be optimal, as it is only a heuristic. The idea for this algorithm is based off the research done by Dr.Hussein Al-Omari**1**.

An analysis of my implementation of the strategy follows:

Algorithm:

while (there exists uncolored nodes) **O(n3)**

setColors() O(n)

selectNodeToColor() O(n2)

for i🡪n

getSaturationDegree() O(n)

getSaturatedColors() O(n)

color node 🡨 set addition, subtraction, finding max of collection O(n\*log(n))

Max time: **O(n3)**

In reality, a more thorough analysis will uncover many linear time operations such as set operations that will add up to yield a more honest run time analysis. However, the big-O analysis is contained at a cubic O(n3) runtime.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nodes** | **Execution Time (ms)** | **Colors Used** | **Max Degree** |
| 50 | 12 | 6 | 16 |
| 100 | 30 | 9 | 31 |
| 150 | 59 | 11 | 43 |
| 200 | 77 | 14 | 58 |
| 250 | 109 | 16 | 69 |
| 300 | 145 | 18 | 79 |
| 350 | 212 | 21 | 88 |
| 400 | 284 | 24 | 103 |
| 450 | 368 | 25 | 111 |
| 500 | 506 | 27 | 127 |

*Figure 1* – Shown above are the runtimes (in milliseconds) for graph sizes ranging from 50, 100, 150, 200, … 500. The respective amount of colors used to color the graph, along with the corresponding max degree, is shown. Pictures for this data are shown on the GitHub ~/pics.

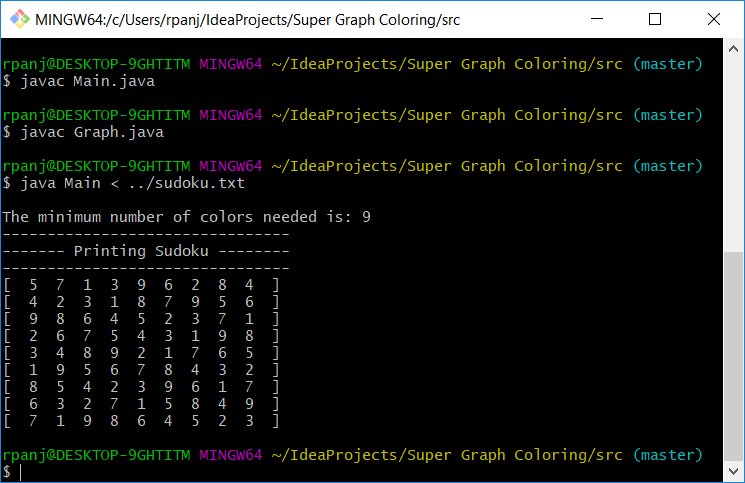
*Figure 2* – Graph size compared to runtime shows at least a quadratic trend. A little different than the predicted O(n3) behavior from the algorithm analysis. The sample size only extends to 500 nodes however, and perhaps the trend would appear more cubic at larger inputs.

For performance analysis purposes, it should be mentioned my laptop running this program has the following specifications is an Intel i5-4200H CPU (2 cores) and 12GB of RAM.

*Figure 3* – Graph size compared to colors used seems to show a linear, or perhaps even logarithmic trend.

*Figure 4* – Interestingly, the max degree of nodes seems to correspond fairly linearly (with strong confidence) with the amount of colors used. Thus we can assume that as the max degree of the nodes within the graph increases, so will the number of colors necessary to color the entire graph.

1. Sudoku Analysis



Shown above is a modified version of the Super-Graph-Coloring solver – one that calls different print methods specifically for sudoku listed inside the Graph.java class. The lines have been uncommented for the purposes of better displaying the sudoku puzzle.

As shown, the example sudoku puzzle has been aptly solved.