Soham Jain

Graph Coloring with Grover's Algorithm: Optimizing Time and Space Efficiencies

Yilmaz SysLab 2025

Introduction

Graph coloring, an NP-complete problem, involves assigning colors to vertices of a graph such that no adjacent vertices share the same color. With an exponential time complexity of $O(m^V)$, where m is the number of colors and V is the number of vertices, classical algorithms struggle due to high time complexity and memory consumption. This project leverages Grover's algorithm, a quantum search algorithm with $O\sqrt{N}$ time complexity, to efficiently address the graph coloring problem. This study focuses on implementing Grover's algorithm in Qiskit and Q# to color a map of the 50 US states. The goal is to compare time and space efficiencies across both programming languages.

Novelty

Grover's algorithm is a heuristic that quadratically speeds up unstructured searches. This project is the first to apply Grover's algorithm to graph coloring across multiple quantum programming languages. Unlike prior work which used Grover's algorithm for coloring small maps like Canada's 13 provinces, this study tackles a larger graph containing 50 vertices and 4 colors. No previous study has explored quantum memory management in Q# and Qiskit for graph coloring. The approach in this project generalizes to other constraint satisfaction problems as well, like course scheduling and Sudoku.

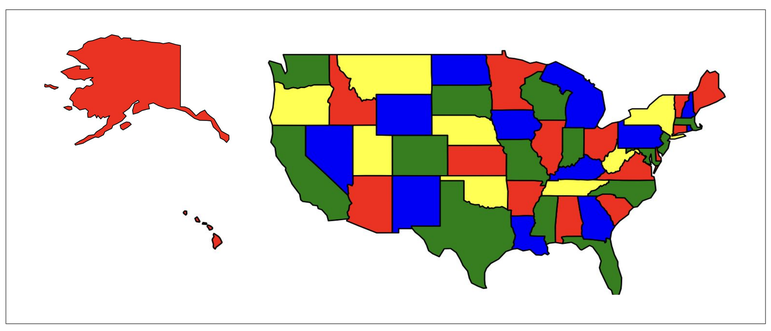

Fig. 1. Graph Coloring with Grover's Algorithm Output

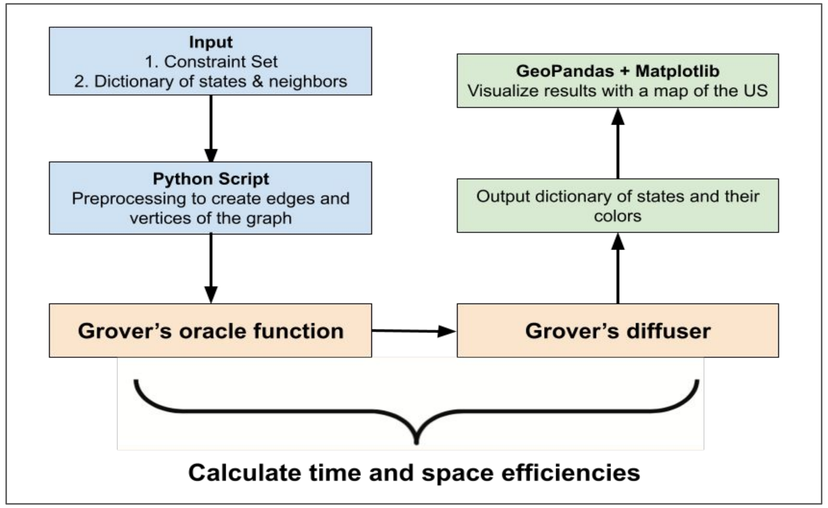

Fig. 2. Systems Architecture

Methods

Fig. 2 shows the systems architecture for this project. The input is a dictionary of states and their adjacent states with a constraint set of four colors: red, green, blue, and yellow. Grover's algorithm employs an oracle to mark valid colorings, as well as a diffuser to amplify their amplitudes. The mathematical formulas for Grover's algorithm are shown in Fig. 3. Quantum circuits are built using existing libraries, including Hadamard transform operations for superposition. The oracle encodes constraints ensuring no adjacent states share colors. Time efficiency is measured as the algorithm's execution time over 20 trials. Space efficiency is quantified by circuit depth, or the number of quantum gates. The output is a dictionary assigning valid colors to each state. Simulations are run on quantum simulators due to limited quantum hardware.

	Average Time (20 trials)	Circuit depth
Qiskit	14.94 seconds	34-42 gates
Q#	34.78 seconds	26-38 gates
Recursive (Python)	43.75 seconds	N/A

Table 1. Time and Space Efficiency Results

Results

The time and space efficiency results of applying Grover's algorithm to graph coloring are shown in Table 1. Qiskit achieved an average execution time of 14.94 seconds over 20 trials, outperforming Q# at 34.78 seconds. The classical recursive Python algorithm was slowest at 43.75 seconds. Furthermore, Q# demonstrated better space efficiency with a circuit depth of 26–38 gates compared to Qiskit's 34–42 gates.

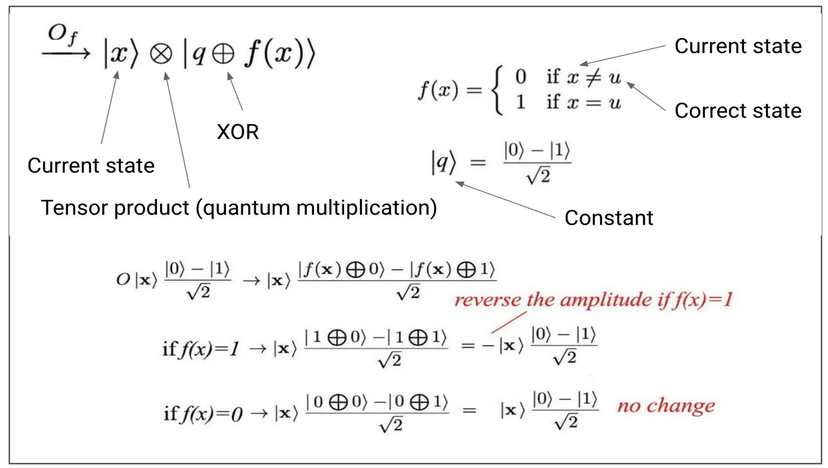

Fig. 3. Grover's Algorithm Formulas

Conclusion and Future Work

Grover's algorithm successfully colored a 50-vertex US map, as shown in Fig. 1. Moreover, this study's finding suggest that Qiskit was the most efficient programming language in terms of time, but Q# had better space efficiency. Quantum approaches showed competitive performance against classical methods. Current quantum hardware limits testing on larger graphs, however. Future work could apply the method to US counties or other constraint satisfaction problems. Additional quantum languages, such as Cirq and Ocean, could be also tested in the future.