ECE650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Final Project Report

Authors:
Zhengkang Chen (ID: 20789905)
Wenting Ju (ID: 20777538)

Date: December 5, 2023

1 Introduction

Our main objective is to help the local police department with their installation of security cameras at traffic intersections. We solve this kind of optimization problem, the Vertex Cover problem. The idea is for the police to be able to minimize the number of cameras they need to install, and still be as effective as possible with their monitoring.

Given an undirected graph G = (V, E), where V is the set of vertices and E is the set of edges, a vertex cover is a subset V' of V such that for every edge (u, v) in E, at least one of u or v belongs to V'. The minimum vertex cover is the smallest possible subset of vertices that still covers all the edges in the graph.

In this project, we implemented a multi-threaded program with three different methodologies to solve this minimum vertex cover problem and analyzed the efficiency of each approach using running time and approximation ratio.

2 Algorithms

2.1 CNF-SAT-VC:

This algorithm is a polynomial-time reduction from VERTEX-COVER to CNF-SAT. It is an NP-complete algorithm. The undirected graph is implemented as a set of literals and clauses in CNF form according to the encoding method. The CNF will be used in the MiniSat SAT solver which can determine whether the CNF is satisfied. The vertex cover exists when the CNF is satisfied.

2.2 Approx-VC-1:

In this algorithm, it consistently picks a vertex of the highest degree (most incident edges). This vertex will be added to the vertex list and then remove all the edges it connects to. Then it completes the process until there are no edges left. Finally, the vertex list is the vertex cover.

2.3 Approx-VC-2:

In this algorithm, it picks an edge $\langle u, v \rangle$, and adds both u and v to your vertex cover. Throw away all edges attached to u and v. Then it repeats till no edges remain.

3 Analysis

In this project, we analyzed how efficient each approach is for various inputs. "Efficient" is characterized in one of two ways: (1) running time, and (2) approximation ratio. For each method, we used randomly generated graphs for $V \in [5, 50]$ and 10 graphs for each V. Then we computed the average and standard deviation across those results.

3.1 Running Time for CNF-SAT-VC:

We used `std::chrono::high_resolution_clock::now()` before and after the thread executed to check the running time. The reason why we didn't use `pthread_getcpuclockid` is because we are using macOS system. Based on their documentation, `pthread_getcpuclockid` is not supported on macOS. We plotted the results into a graph with the y-axis as the average running time and the x-axis is the number of vertices. Figure 1 shows the running time for algorithm CNF-SAT-VC. According to the graph, the running time increases exponentially along with the increase in the number of vertices. The running time increases dramatically when increasing V from 15 to 20 as well as the standard deviation. As it took too long to run when V is bigger than 20 and the algorithm will be time out. Therefore, only the running time when V is smaller than 20 is recorded.

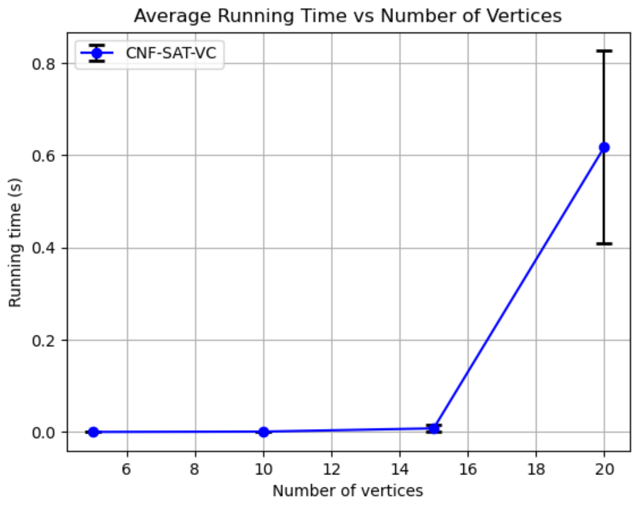

Figure 1: Running time of CNF-SAT-VC with vertices range [5:20].

3.2 Running Time for APPROX-VC-1 and APPROX-VC-2:

Figure 2 shows the running time for the algorithm Approx-VC-1 and Approx-VC-2. According to the graph, the running time for both algorithms increases linearly. And these two algorithms have similar running time. The error bars on each point represent the standard deviation of the running time for each number of vertices. The standard deviation is a measure of the amount of variation or dispersion in a set of values. Here, larger error bars indicate that there was more variability in the running times at that particular number of vertices of graph. Conversely, smaller error bars indicate that the running times were more consistent. From the graph, it seems that APPROX-VC-1 has smaller standard deviations as compared to APPROX-VC-2, suggesting that its running times are more consistent across the trials for different graphs. The APPROX-VC-2 algorithm shows greater variability, especially as the number of vertices increases.

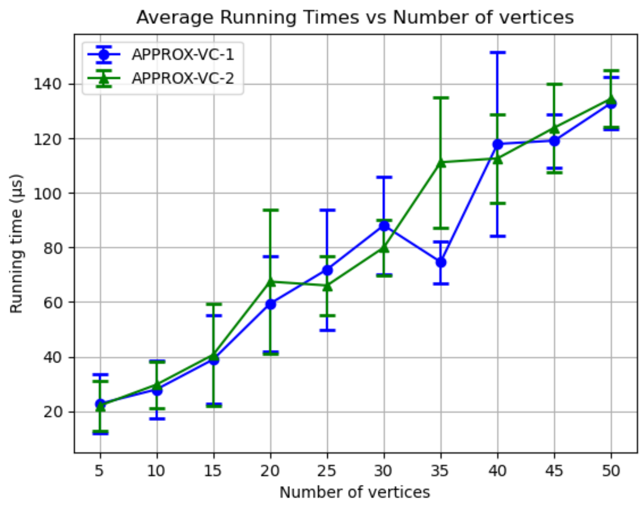

Figure 2: Running time of APPROX-VC-1 and APPROX-VC-1 with vertices range [5:50].

3.3 Approximation Ratio:

We characterized the approximation ratio as the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover. For measuring the approximation ratio, compare it to the output of CNF-SAT-VC, which is guaranteed to be optimal. Figure 3 shows the approximation ratio for Approx-VC-1 and Approx-VC-2. According to the graph, Approx-VC-1 has a better performance than Approx-VC-2 since the approximation ratio of Approx-VC-1 is equal or almost equal to 1. That means it produces almost the same vertex cover as the CNF-SAT-VC which is guaranteed optimal solution. And the approximation ratio of Approx-VC-2 is around 2 which means it produces around two times vertices in the vertex cover that the optimal solution.

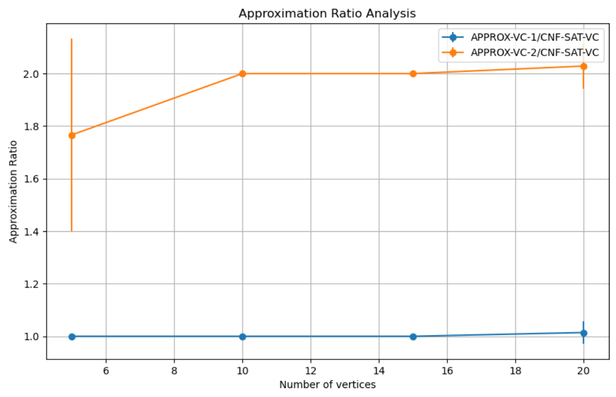

Figure 3: Approximation ratio of APPROX-VC-1/CNF-SAT-VC and APPROX-VC-2/CNF-SAT-VC with vertices range [5:20].

4 Conclusion

In conclusion, the CNF-SAT-VC will guaranteed to provide optimal solution as it will the SAT-solver with CNF. However, it is an polynomial time algorithm so the running time will increase exponentially as the number of vertices increase. It is inefficient to use it while the number of vertices is larger than 20. Comparing Approx-VC-1 and Approx-VC-2, they have the similar running time and they are linear time algorithms. However, the approximation ratio of Approx-VC-1 is around 1 and smaller than the approximation ratio of Approx-VC-2. That means Approx-VC-1 produces almost the same as the optimal solution but it has smaller running time than CNF-SAT-VC while the number of vertices is large. In addition, Approx-VC-1 has a smaller standard deviation so it is more consistent across different graphs. Therefore, Approx-VC-1 is efficient when the number of vertices larger than 20.