ECE 650: Methods and Tools for Software Engineering

Project

Mayav Rohan Antani - 21082382

Mehmet Ali Pelit - 21113622

1 Introduction

1.1 What is Vertex Cover

In graph theory, a vertex cover (sometimes node cover) of a graph is a set of vertices that includes at least one endpoint of every edge of the graph.

In computer science, the problem of finding a minimum vertex cover is a classical optimization problem. It is a typical example of an NP-hard optimization problem that has an approximation algorithm.

Formally, a vertex cover V' of an undirected graph $G = (V, E)$ is a subset of V such that $uv \in E \Rightarrow u \in V' \vee v \in V'$, that is to say it is a set of vertices V' where every edge has at least one endpoint in the vertex cover V'. Such a set is said to cover the edges of G.

1.2 How to Calculate Minimum Vertex Cover

The minimum vertex cover problem is the optimization problem of finding a smallest vertex cover in a given graph. The vertex cover problem is an NP-complete problem: it was one of Karp's 21 NP-complete problems. It is often used in computational complexity theory as a starting point for NP-hardness proofs.

1.2.1 CNF-SAT-VC

CNF-SAT is the following problem:

- Input: a propositional logic formula, F, in Conjunctive Normal Form (CNF). That is, $F = c_1 \wedge c_2 \wedge ... \wedge c_m$, for some positive integer m. Each such ci is called a "clause". A clause $ci = l_{(i,1)} \vee ... \vee l_{(i,p)}$, for some positive integer p. Each such $l_{(i,j)}$ is called a "literal." A literal $l_{(i,j)}$ is either an atom, or the negation of an atom.
- Output: True, if F is satisfiable, false otherwise. A polynomial-time reduction from VERTEX-COVER to CNF-SAT is presented.

1.2.2 APPROX-VC-1

Pick a vertex of highest degree (most incident edges). Add it to the vertex cover and throw away all edges incident on that vertex. Repeat till no edges remain.

1.2.3 APPROX-VC-2

Pick an edge $<u, v>$, and add both u and v to the vertex cover. Throw away all edges attached to u and v. Repeat till no edges remain.

2 Approach

This project's main goal is to support local law enforcement in installing surveillance cameras at junctions. Our goal is to monitor every roadway thoroughly while requiring the fewest possible cameras.

- We develop a multi-threaded application comprising four threads to execute CNF-SAT-VC, APPROX-VC-1, APPROX-VC-2 algorithms, and manage input/output operations.
- We employ graphs produced by graphGen as our dataset and do quantitative analysis of our function under different input conditions. Next, we compare and contrast the three functions according to their effectiveness.

3 Analysis

Graphs for various values of $|V|$ (number of vertices) are generated and the running time and approximation ratio are measured for each algorithm. This is done by generating graphs for $|V| \in [5, 50]$. The algorithms are run for vertices in increments of 5. That is, graphs with 5, 10, 15, . . . , 50 vertices are generated. At least 10 graphs for each value of $|V|$ are generated, then the time and approximation ratio for each such graph is computed. The running time for at least 10 runs of each such graph is measured. Then, the mean (average) and standard deviation across those 100 runs for each value of $|V|$ is computed. For the approximation ratio, if there is any random component (e.g., which edges is chosen, for APPROX-VC-2), then that is measured multiple times as well for each graph.

3.1 Running Time

The Running Time for each algorithm, i.e., CNF-SAT, APPROX-VC-1 and APPROX-VC-2, is the amount of time taken to execute. This running time is measured using the **pthread_getcpuclockid()**

function.

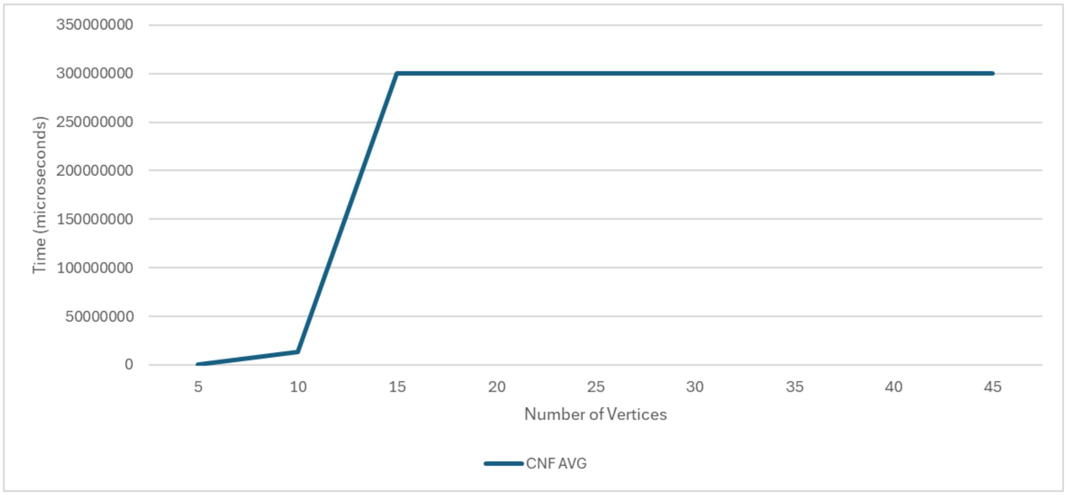

Figure 1: Number of vertices versus Running Time for CNF-SAT-VC

The figure 1 shows the the graph for running time (in μs) versus the number of vertices for CNF-SAT-VC algorithm. Looking at the graph, it can be observed that the runtime is quite low for 5 and 10 vertices. As number of vertices pass 15, the run time increases exponentially and stays extremely high for the rest of the test. This is because of the timeout condition.

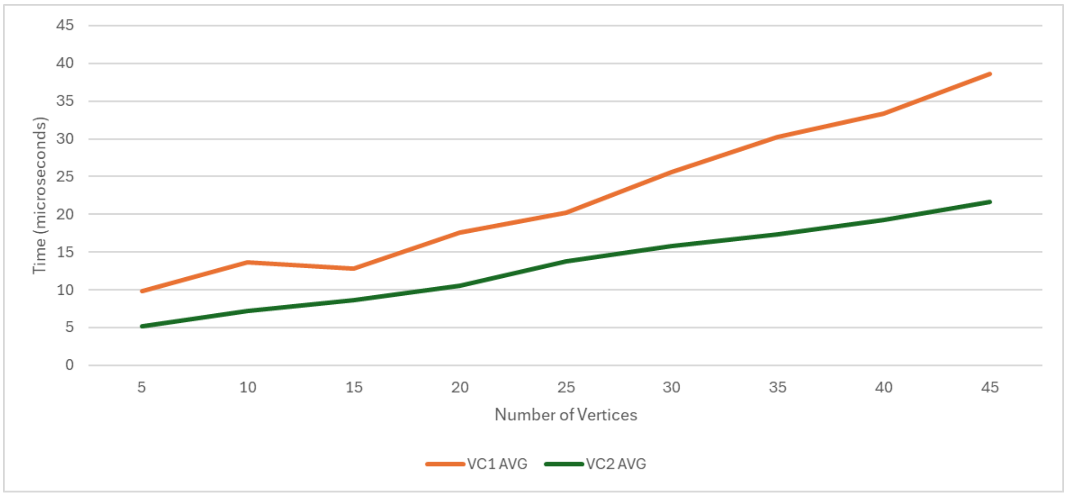

Figure 2: Number of vertices versus Running Time for APPROX-VC-1 and APPROX-VC-2

The figure 2 shows the graph for running time (in μs) versus the number of vertices for the algorithms APPROX-VC-1 and APPROX-VC-2. It can be seen that both the graphs are almost linearly increasing throughout the entire test. Even though the running time increases for both the algorithms, APPROX-VC-2 always has a lower running time than APPROX-VC-1. This means that APPROX-VC-2 is faster than APPROX-VC-1.

On comparing the running time values of all three algorithms, it is seen that APPROX-VC-2 is the fastest and CNF-SAT-VC is the slowest.

3.2 Approximation Ratio

Approximation Ratio is the ratio of the size of the computed vertex cover to the size of an minimum vertex cover.

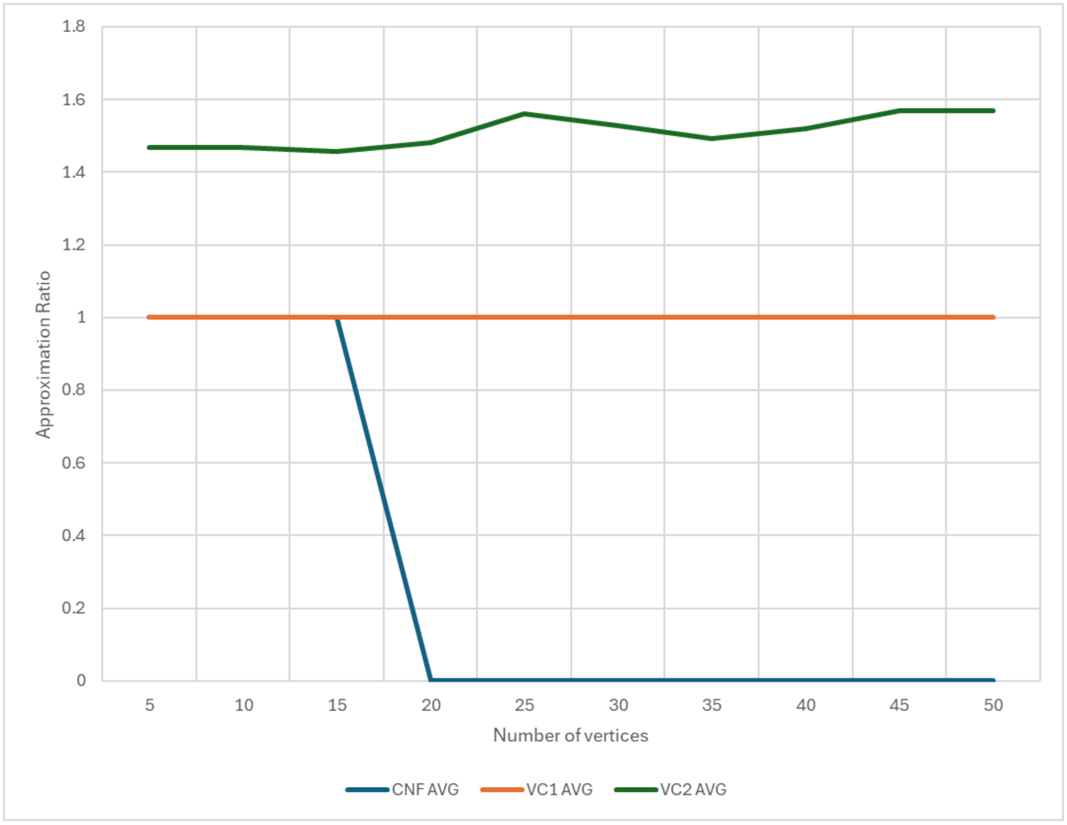

Figure 3: Approximation Ratio versus Number of Vertices

We compare the size of the vertex cover of the algorithms with the size of the minimum vertex cover calculated. This means smaller ratio would be better, i.e., smaller ratio would be a better, closer to optimal, solution.

From figure 3 it can be seen that CNF-SAT-VC has all the ratio values equal to one upto 15 vertices. This is to be expected as that is the benchmark to which the other algorithms are compared. The reason for the approximation ratio of CNF-SAT-VC drops to 0 after 15 vertices is that it times out. On timing out, the vertex cover is not considered. It is also observed that the algorithm APPROX-VC-1 has values closer to one as compared to the APPROX-VC-2 algorithm, this would mean that APPROX-VC-1 algorithm finds better vertex covers as compared to APPROX-VC-2. Furthermore, since the ratio values for APPROX-VC-1 are so close to one, it would mean that it gives the optimal vertex cover most times.

Finally, on looking at both the graphs, it can be seen that APPROX-VC-1 give outputs very close to optimal outputs while running significantly faster than CNF-SAT-VC. Thus, making it a better choice for an algorithm for larger values of vertices.

4 Optimization Techniques

One way to optimize the algorithms is by optimizing the time taken to select the ideal value of size of the vertex cover. This means to select the ideal (*smallest*) value for k. This can be done using various search methods. One way to find the smallest value would be using binary search to get the smallest value of k.

In order to use binary search, we start with an upper and a lower limit for the value of k. The bounds of the value of k could be -1 for lower bound and equal to the number of nodes in the graph for the upper bound. Using binary search, we get the mid-point of the two bounds and then check if minimum vertex cover for that value of k exists. If the minimum vertex cover does exist, the upper bound is changed to the mid-point, else the lower bound is changed to the mid value. This process is repeated till the optimal value of k is found.

Using binary search makes the runtime complexity of finding the optimal value of k logarithmic. Binary search involves narrowing down the search space in every iteration, by dividing it in half based on the results of comparisons, giving us $O(log(n))$ complexity.

5 Conclusion

In conclusion, the project aims to help the local police department with their installation of security cameras at traffic intersections. The project helps determine the minimum number of cameras required and their placements on the intersections in order to cover all the roads. This project implements three different algorithms (CNF-SAT, APPROX-VC-1 and APPROX-VC-2) to determine the minimum vertex cover. Finally analysis and comparison of these algorithms is performed based on their running time and approximation ratio.