Quantitative Analysis of Mini-Vertex Cover Using Various Algorithms

ECE650

Methods and Tools of Software Engineering

Submitted by

Devansh Kansara - 21112551

Harmanjot Singh - 21114621

1 Introduction

In this project, different techniques and algorithms have been used to solve the parameterized Vertex Cover problem, and the corresponding results and performance of different algorithms are presented in the following report. The first approach is to encode vertex cover to CNF-SAT clauses in a graph. Then Mini Sat solver have been used to solve the problem in a more effective way. Two alternate approximation algorithm which are Approx-VC-1 and Approx-VC-2 have been used to calculate the vertex cover just for comparison. These alternate algorithms have been explained in detail in the further sections.

2 Algorithms

2.1 CNF-SAT:

The CNF-SAT problem is a Boolean satisfiability problem, which is the reduction of the vertex cover of this sort is polynomial-time reduction issue. The A4 encoding is used to create the clauses. CNF is a conjunction of disjunction of literals. i.e. the literals are OR with each other and the clauses are AND with each other. These ANDed clauses are given to Mini Sat solver, which determines if the clauses are satisfiable or not. In this case, if a vertex cover exists or not.

2.2 Approx-VC-1:

In this approach, we select a vertex that appears frequently in the set of edges (Vertex of the highest degree). Then this vertex is deleted from the list and all the relevant edges are also removed. This continues until there are no more edges remaining in the edge list.

2.3 Approx-VC-2:

Here, we select a random edge form the edge list, then we add this edge to the VC list, where we assume that the VC is the vertex cover. Then all the edges corresponding with all the vertices on both ends of the edge is removed from the graph. This continues until the remaining edge list is clear.

We used four threads to solve the minimum vertex cover: three for each of the methods CNF-SAT, Approx-VC-1, and Approx-VC-2, and one for I/O produced by the main function. The adjacency list is created by the main thread, often known as the I/O thread, after it has read the user-provided input (V, E).

3 Method Used for Analysis

In order to find the efficiency of these three algorithms for the number of vertices, we make use of two factors namely (1) Running time, and (2) Approximation Ratio. We generate 10 graphs for each vertex. With help of this, we compute the Running time and the Approximation ratio for each graph. For better fairness of the evaluation of the algorithms, the inputs are generated through a graph generator called **GraphGen** which is in /home/agurfink/ece650/graphGen/graphGen on eceubuntu. This tool could generate different edges with same numbers of vertices.

3.1 Running Time:

We have measured the running time of each algorithm threads and calculated their mean and standard deviation. For this we have used, clock() and finish() functions of the <time.h> library in C++.

3.2 Approximation Ratio:

The Approximation Ratio is defined as the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover.

For the graphs produced by graphGen, we compute the approximation ratio and execution time for a range of values of V (number of vertices). The graph has a 5-interval range from V=5 to V=50. Next, for each and every value of V, we calculate the average and standard deviation over that 50 samples.

4 Analysis

4.1Running time of CNF-SAT-VC:

We have plotted three graphs here, where running time is represented on the y-axis and the number of vertices V on the x-axis.

The CNF-SAT Algorithm running time is plotted in Figure 1, where the mean and standard deviation of the running times are displayed with 5-unit increments. The execution time of our CNF-SAT-VC method grows exponentially as the number of vertices rises. At first, finding satisfiability takes less time for smaller values of vertices because there are fewer literals and clauses; however, for greater values of vertices, there are more literals and clauses, which increases the time consumption. As the number of vertices increases, this graph will progressively grow exponentially. Additionally, it is evident to us that when the vertex is 15, the standard deviation has significantly grown.

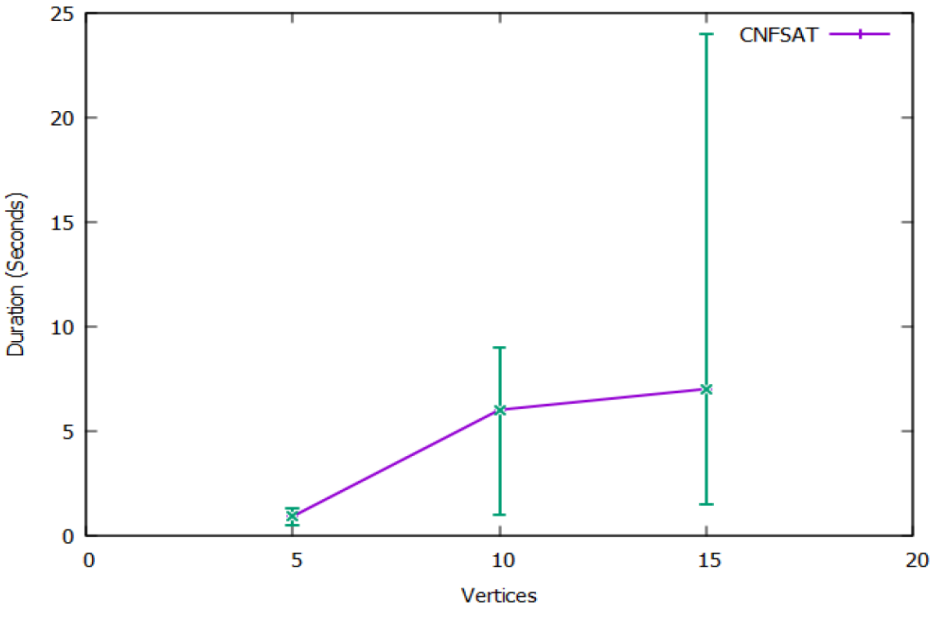

Figure 1: Running time CNF-SAT for the range of vertices [5, 15].

4.2 Running time of APPROX-VC-1 and APPROX-VC-2:

We were able to examine the running times of APPROX-VC-1 and APPROX-VC-2 in Figure 2 by plotting the mean and standard deviation of the running durations for each value of V in [5,50]. As can be seen below, the running times of APPROXVC-1 and APPROX-VC-2 have increased as the number of vertices has increased. However, it should be noted that APPROX-VC-1 runs for a greater amount of time than APPROX-VC-2. This is because optimization issues using APPROXVC-1 find the vertex's greatest degree before going to each node individually to find the vertex cover list. As opposed to this, APPROX-VC-2 does not search for the highest degree vertex or return to the node that was previously visited throughout each iteration. Consequently, there are fewer nodes to visit in the following iteration. As a result, Approx-VC-1's temporal complexity is somewhat greater than Approx-VC-2's.

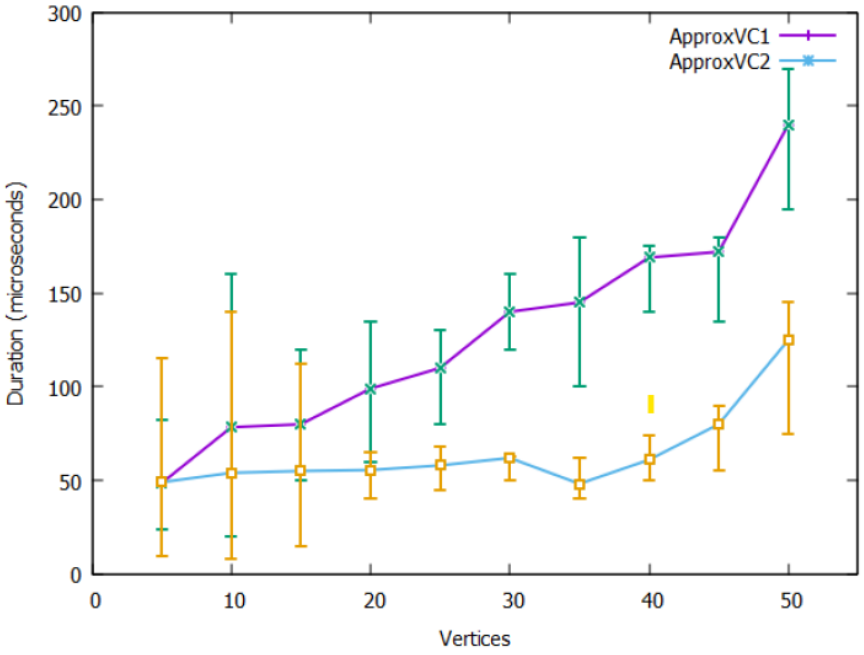

Figure 2: Running times of APPROX-VC-1 and APPROX-VC-2 for the vertices in range [5, 50].

4.3 Approximation Ratio:

In Figure 3, the y-axis represents the **approximation ratios**, while the x-axis corresponds to the number of vertices **V**. In our project problem statement, we consider **CNF-SAT-VC** as the optimal solution. Therefore, we define the approximation ratio by comparing the vertex cover size of **CNF-SAT-VC** with the vertex cover produced by **APPROX-VC-1** and **APPROX-VC-2**.

The results are based on the **minimum vertex cover**, and from the graph, we observe that the vertex cover produced by **APPROX-VC-1** closely matches that of **CNF-SAT**. Since **CNF-SAT-VC** guarantees the minimum number of vertices in the cover, its approximation ratio is always 1 for all vertex cases. However, this is not the case for **APPROX-VC-2**. This algorithm exhibits the worst approximation ratio among the three because it does not prioritize vertices with the highest degree. Instead, it selects arbitrary vertices, resulting in vertex covers that are typically not close to the minimum vertex cover.

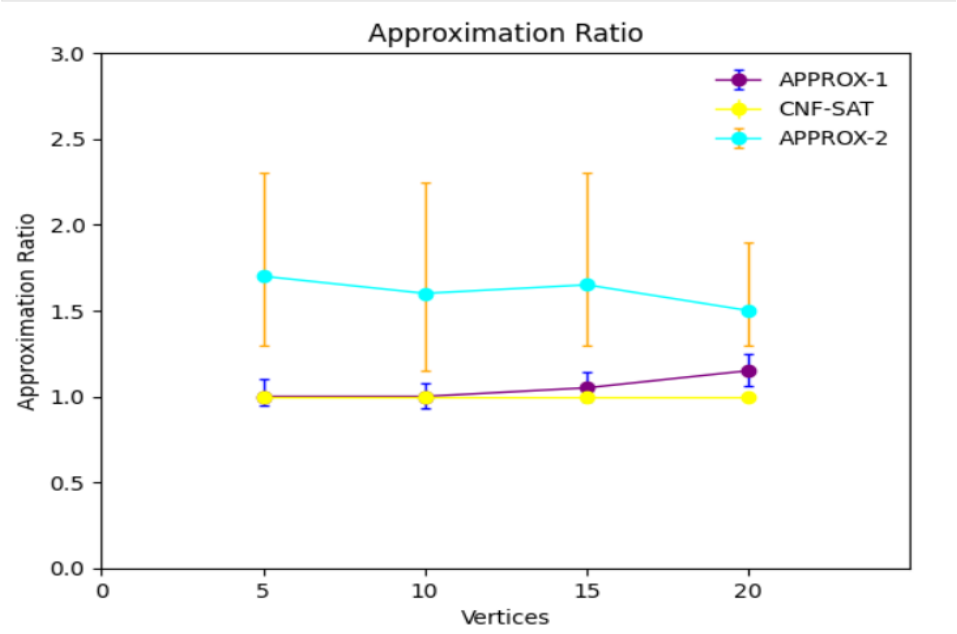

Figure 3: Approximation Ratios of APPROX-VC-1 /CNF-VC and APPROX-VC-2/CNF-VC

5 Conclusion

This document presents the application of three distinct strategies to address the vertex cover problem, followed by an assessment of their effectiveness through considerations of running time and approximation ratio. For vertices less than 15, CNF-SAT-VC is the optimal choice. However, after vertices V > 15, there was a steep increase in the running time of CNF-SAT-VC. Hence, it was noticed that the running time of the algorithm increases with the no. of vertices. And this is true for all the algorithms.

Now when we consider Approximation ratio, APPROX-VC-1 becomes more efficient than APPROX-VC-2. When the running time is the main factor for consideration, APPROX-VC-2 should be preferred algorithm to calculate vertex cover because it takes the least time among all the three. But if the main focus is to find minimum vertex cover irrespective of the running time, APPROX-VC-2 may fail. However, in scenarios where both time efficiency and optimal solution is paramount, APPROX-VC-1 is the best choice.