Quantitative Analysis of Mini-Vertex Cover Using Various Algorithms

Submitted by

Mamta Mamta - 20867979

Catherine Hannah Thangaian - 20868337

1 Introduction

Our main objective is to help the local police department with their installation of security cameras at traffic intersections. 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. The problem of finding a minimum vertex cover is a classical optimization problem in computer science and is a typical example of an NP-complete problem that has an approximation algorithm.

A vertex cover of a graph G(V, E) is a subset of vertices V such that for every edge (u, v) is a subset of E, at least one of the vertices u or v is in the vertex cover. In this project we discuss different techniques and algorithms used to solve the parameterized Vertex Cover problem. The first approach is to encode vertex cover to CNF-SAT clauses in a graph (using MiniSat). We also have two approximation algorithms Approx-VC-1 and Approx-VC-2 that are used to calculate the vertex cover.

2 Algorithms

2.1 CNF-SAT:

The reduction of vertex cover to CNF-SAT is a polynomial-time reduction. The graph is implemented as a set of literals and clauses to be in CNF form. 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 one another. These ANDed clauses are given to MiniSat SAT solver, which determines if the clauses are satisfied i.e. if vertex cover exists for the current graph.

2.2 Approx-VC-1:

In this approach we pick the vertex which occurring most frequently in the given set of edges i.e. vertex with highest degree. This vertex is added to the vertex cover list and we remove the edges that are attached to this vertex. This process is repeated until no edges remain.

2.3 Approx-VC-2:

In this approach we pick an arbitrary edge (u, v) from the set of edges E and add them to VC, where VC is the vertex cover. It deletes every edge that is connected to these both vertices. We continue this process of selecting an edge and deleting its adjacencies until no edges remain.

To solve the minimum vertex cover we have used 4 threads: one for I/O which is created by the main function and other three threads for each algorithms CNF-SAT, Approx-VC-1 and Approx-VC-2. The main thread i.e. I/O thread reads the input(V,E) given by the user and based on it creates the adjacency list.

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 vertices. With help of this, we compute the Running time and the Approxaimation ratio for each graph

3.1 Running Time:

To measure the running time of an algorithm we make use of a pthread utility "pthread_getcpuclockid()" which gives us the CPU execution time for that thread. Depending upon the number of vertices that is given as input we can calculate the running time for each algorithm and find the maximum CPU time taken by an algorithm.

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.

We calculate the running time and approximation ratio for various values of V (number of vertices), for the graphs generated by graphGen. The graph ranges from V=5 to V=50 with an interval of 5. Then, we compute the

average and standard deviation across those 100 samples for each value of V.

4 Analysis

4.1 Running 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.

In figure 1 we have plotted the running time of CNF-SAT Algorithm, where we plot the mean and standard deviation of running times with increments of 2. In our algorithm for CNF-SAT-VC, the running time increases exponentially with the increase in the number of vertices. Initially, for smaller values of vertices, it takes less time to find satisfiability because less literals and less clauses and for larger values of vertices, larger the number of literals and clauses contributes to the increase in time consumption. This graph will steadily increase exponentially with increase in vertices. We can also clearly see that the standard deviation has increased a lot when the vertex is 15.

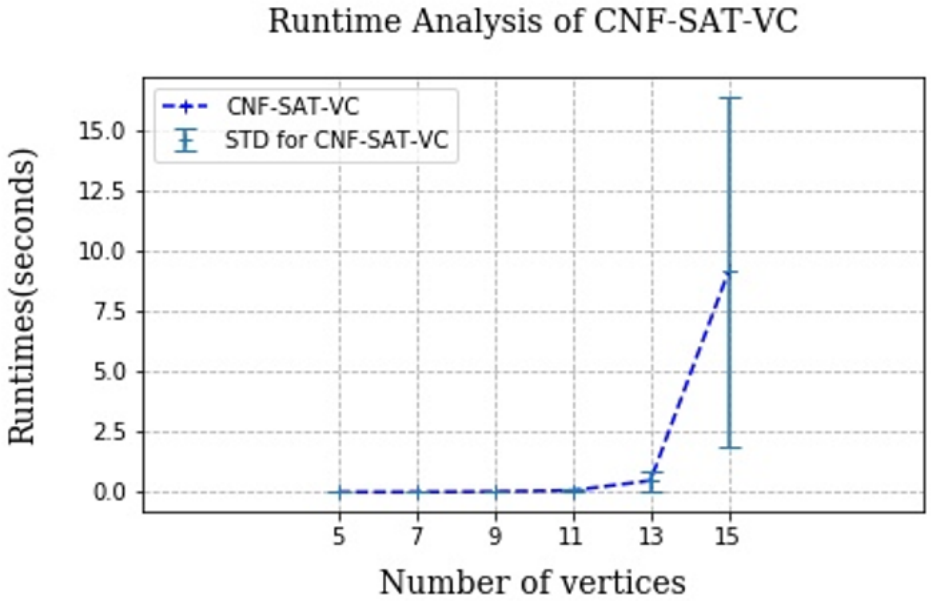

Figure 1: Running time CNF-SAT for the range of vertices [5, 15].

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

In Figure 2, we analysed the running time of APPROX-VC-1 and APPROX-VC-2 by plotting the mean and standard deviation of running times for each value of V in [5,50]. As shown below, the running time of both APPROX-VC-1 and APPROX-VC-2 has increased with the increase in number of vertices. However, the running time of APPROX-VC-1 is observed to be relatively higher compared to APPROX-VC-2. This is because APPROX-VC-1 is used in optimization problems which calculate the maximum degree of the vertex and then finds the vertex cover list by visiting each node in every iteration. Whereas APPROX-VC-2 doesn't find the highest degree vertex and in each iteration it doesn't go back to the node that was visited earlier. In turn the number of nodes to be visited in next iteration is less. Thus, the time complexity of Approx-VC-1 is little higher than that of Approx-VC-2.

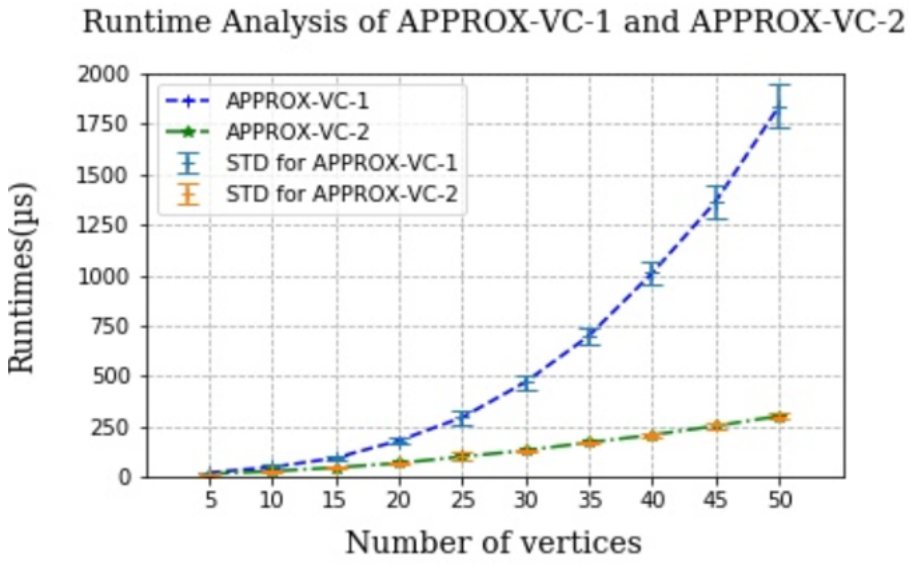

Figure 2: Running times of APPROX-VC-1 and APPROX-VC-2 for the vertices in range [5, 50].

In figure 3 we have plotted the mean and standard deviation of APPROX-VC-1 and APPROX-VC-2 for the vertices range of [5,15].

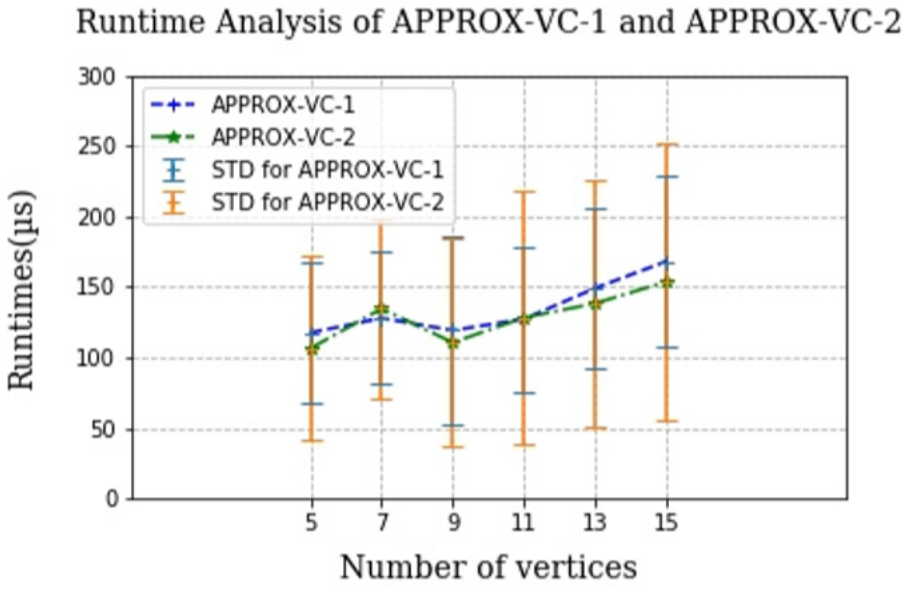

Figure 3: Running times of APPROX-VC-1 and APPROX-VC-2 for the vertices in range [5, 15].

4.3 Approximation Ratio:

The Figure 4 plots the approximation ratios, where approximation ratio is represented on the y axis and the number of vertices V on the x axis. In the project problem statement, we have decided that CNF-SAT-VC is an optimal solution. Hence, we define the approximation ratio by comparing the vertex cover size of CNF-SAT-VC with the vertex cover produced by APRROX-VC-1 and APPROX-VC-2.

The result of this is in terms of the minimum vertex cover and from the graph we understand that the vertex cover generated by APPROX-VC-1 is almost equal to that of CNF-SAT. Since CNF-SAT-VC guarantees to find the minimum number of vertices cover thus the approximation ratio is equal to 1 in all vertex cases here, but this is not the case for APPROX-VC-2. We can see that this algorithm has the worst approximation ratio among the other two because, it doesn't find vertex with the highest degree instead, it takes the arbitrary vertex and vertex cover usually not close to the minimum vertex cover.

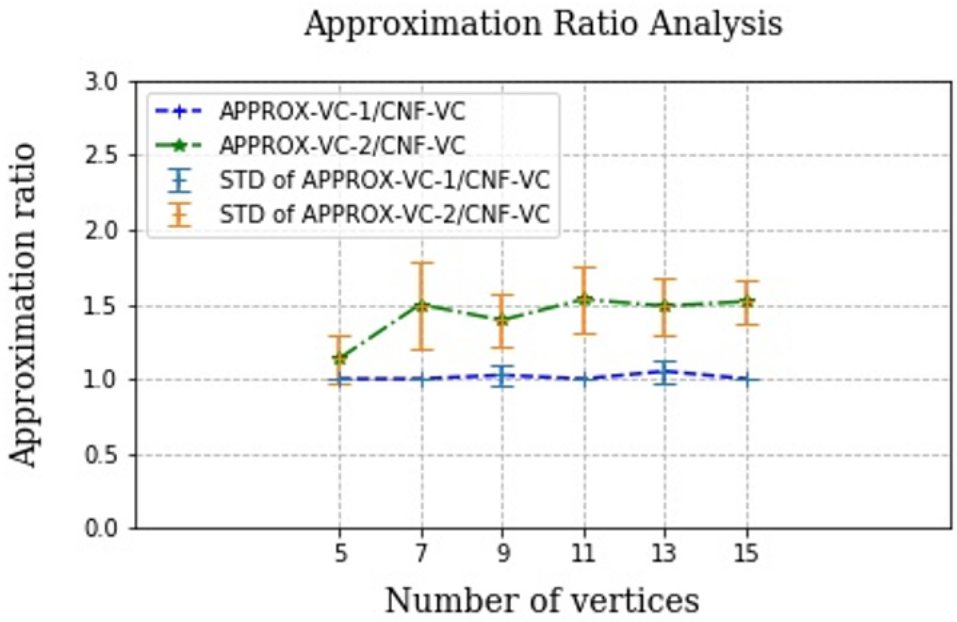

Figure 4: Approximation Ratios of APPROX-VC-1 /CNF-VC and APPROX-VC-2/CNF-VC

5 Conclusion

CNF-SAT-VC couldn't perform better to produce an output for large number of vertices. It is important to note that after vertices V $>$ 15 there was a steap increase in running time. Hence to avoid waiting for its result we have used the timeout of 120 seconds, where the CNF-SAT-VC thread is stopped when it consumes more than 120 seconds to find the output for a given vertex. On testing CNF-SAT-VC, we observed that the timeout was occurring when the vertex is more than V 15. It was noticed that the running times in all the algorithms is increasing with the number of vertices.

In conclusion, when we consider Approximation ratio APPROX-VC-1 algorithm becomes more efficient than the APPROX-VC-2. When running time is the constriction of design APPROX-VC-2 should be the preferred algorithm to compute vertex covers because it takes the least time amongst all the three. But if the main focus is to find minimum vertex then, APPROX-VC-2 is more likely to fail.