# Analysing different vertex cover algorithms using running time and approximation ratios

Saem Abdulhamid Sheth (20807869)
Priya Saini (20752999)

ECE650, Fall 2019

## 1 Introduction

The vertex cover in the graph theory can be defined as the set of vertices such that every edge of the graph incident to at least one vertex of the vertex cover set. The vertex cover problem belongs to NP class of optimization problems. The minimum vertex cover set can be defined as the set of a minimum number of vertices needed to cover all the edges in the graph. Here, three different approaches were used to obtain the vertex cover set of graphs. We also calculated the approximation ratio and approximate time in obtaining the vertex cover problem for comparing all the three approaches.

This analysis shows the performance comparisons between three different approaches to find the vertex cover set. The three approaches are,

1. CNF-SAT solver: A CNF-SAT solver finds the vertex cover of the given graph by polynomial-time reduction of the problem. In the CNF-SAT algorithm, MINISAT was used. MINISAT is developed for generating the boolean variables and thereby generating the boolean expressions. Apart from this MINISAT is also used to determine whether the boolean expressions are satisfied, given an expression in terms of AND, OR, NOT, parentheses and boolean variables. The CNF-SAT solver developed for this project always gives the minimum size vertex cover. This algorithm always gives the most efficient output, in terms of the size of the output vertex cover set.

2. APPROX-VC-1: In this algorithm, the vertex having the highest edges attached to it (having the highest degree) is picked, all the edges attached to it is removed and that vertex is added in the vertex cover set. This procedure is repeated until all the edges in the given graph are removed.

3. APPROX-VC-2: In this algorithm, a random edge is selected at first, the endpoints of that edge is then added to the vertex cover set and all the edges attached with these two endpoint vertices of the edge is removed. This procedure is repeated until all the edges are removed from the graph.

In this project, the multi-threaded approach was used for implementation of the vertex cover problem solution. Four threads corresponding to 1) For input/output 2) for CNF-SAT solver algorithm 3) for APPROX-VC-1 algorithm and 4) APPROX-VC-2 algorithm for finding vertex cover were used. The obtained results of this program were then analyzed and compared by using Microsoft Excel software.

The detailed analysis of the results is shown in the next section.

# 2 Analysis

The performance and accuracy of these three algorithms are measured in terms of,

1. Running time of the algorithm

2. Approximation ratio

The whole program was performed on AMD A8 processor having 4GB RAM, CPU clock of 2 GHz and 64 bit UBUNTU operating system. The performances of these algorithms might vary with computers having different configurations.

## 2.1 Running time of the algorithm

Amongst the three algorithms, the CNF-SAT algorithm was taking maximum time and had a huge difference in running time comparatively. Therefore, we took an approach of analyzing CNF-SAT and the other two algorithms

separately for better analysis. As for all the algorithms, it was noticeable that with an increase in the number of vertices the running time increases, this is due to the fact that with increasing number of vertices CPU needs more time to process the data.

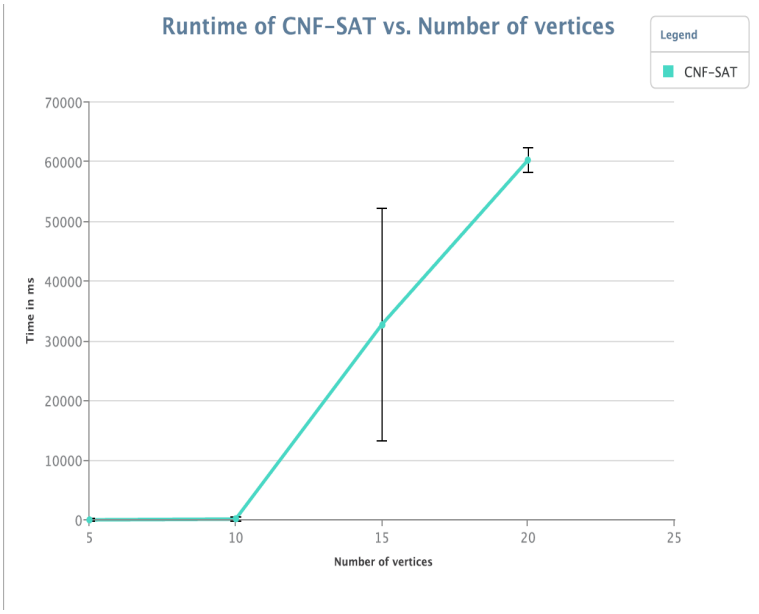


Figure 1: Runtime of CNF-SAT vs Number of vertices

We have generated the above graph by giving the inputs of 10 graphs of each 5,10 and 15 vertices, and 2 graphs of 20 vertices. Each of these graphs were run for 10 times. Hence we have 100 samples for each value of V of 5,10 and 15, and 20 samples of graph having 20 vertices. We have used Microsoft Excel for generating this graph as a result of various inputs having different combinations of vertices and edges.

From the graph, it can clearly be observed that CNF-SAT performs very fast for the inputs having a smaller number of vertices. But with an increase in the number of vertices, the time for computing vertex cover increases exponentially. This is due to the fact that if we increase the number of input vertices, the number of clauses in our reduction problem will increase. With increase in the number of clauses the SAT solver takes more time in checking whether the clause is satisfied or not, this, in turn, justifies the exponential behavior of the running time of the SAT solver algorithm. The

running time of SAT solver increases to a great extent after input vertices increase from 15. In addition to this, the error bars in the above graph show standard deviation, which indicates that with an increase in the number of input vertices the running time of the same graph varies greatly for the graphs having the same number of vertices. The running time complexity of this algorithm is $O(V^4)$.

The following graph shows the comparison between APPROX-VC-1 algorithm and APPROX-VC-2 algorithm.

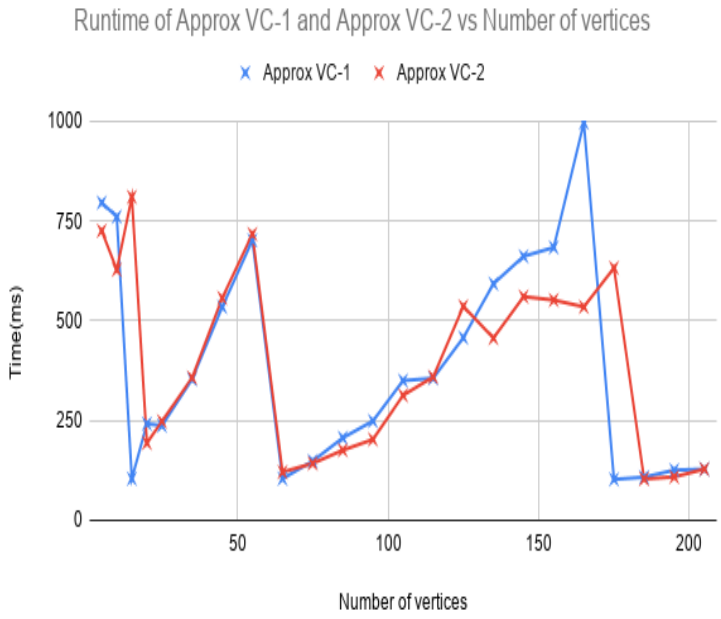


Figure 2: Runtime of Approx VC1 and Approx VC2 vs Number of vertices

Above graph is obtained by taking V from [5, 205] with a gap of 5 vertices. From the above graph, it can clearly be observed that the computational time of APPROX-VC-2 is a little lesser than APPROX-VC-1. This is because calculating the highest degree vertex in input would always take more time than picking any random edge. The APPROX-VC-1 algorithm has a higher computational cost than APPROX-VC-2 and thus we can state that the complexity of APPROX-VC-1 is higher than APPROX-VC-2. We can also say that APPROX-VC-1 is a greedy approach as we select the highest degree vertex every time for calculating the vertex cover. The running time of

APPROX-VC-1 is  O(V2 ) while the running time of APPROX-VC-1 is O(V*E).

## 2.2 Approximation ratio

In order to compare the efficiencies of these algorithms, we have used approximation ratio as our second approach for the analysis. The approximation ratio was calculated by using the following formula,

$$Approximation\ Ratio = \frac{(Size\ of\ calculated\ vertex\ cover)}{Size\ of\ optimal\ vertex\ cover\ (CNF\ SAT)} \quad (1)$$

Using equation (1), the approximation ratio will always be equal to 1 for CNF-SAT algorithm as the algorithm will always give minimum size vertex cover. The graph below shows the comparison amongst different algorithms in terms of approximation ratio.

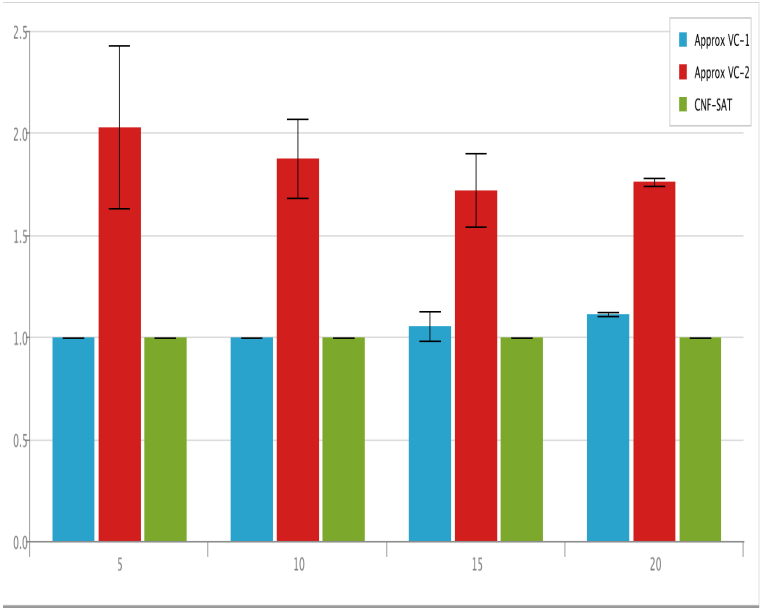


Figure 3: Approximation Ratio analysis

As we discussed earlier that algorithm APPROX-VC-2 takes the minimum time for finding the vertex cover but it has the worst performance in terms of approximation ratio as it always gives the most number of vertices.

This is because the algorithm APPROX-VC-2 will always pick any edge and remove all the edges associated with the endpoints of that edge without considering any other criteria, because of this there is a high possibility that it would have more vertices in vertex cover than APPROX-VC-1 algorithm would have. So the probability of getting a minimum vertex cover set (an optimal solution) is more in the APPROX-VC-1 algorithm as compared to the APPROX-VC-2 algorithm.

# 3 Conclusion

In this project, we studied the three different algorithms for calculating the vertex cover problem for a given input graph. After studying the performances of these algorithms from Running time and Approximation Ratio point of view, it can be stated that the running time of all the algorithms increases with the number of input vertices. If we consider the running time for measuring the performance then CNF-SAT performs worst as its running time increases exponentially with an increase in the number of input vertices, while the APPROX-VC-2 algorithm performs the best and gives the best running time efficiency. In contrast to this, if we consider the approximation ratio for measuring the efficiencies then the APPROX-VC-2 algorithm gives the worst efficiency, and the APPROX-VC-1 algorithm has more possibilities to give the optimal solution in polynomial time than the APPROX-VC-2 algorithm. However, the CNF-SAT algorithm gives the best efficiency as it always calculates an optimal solution (a minimum vertex cover set) but it can not be scaled to calculate large number of input vertices in reasonable time.