Final Course Project

ECE 650 Methods and Tools for Software Engineering

Diego Cepeda 20763269
Devon Deane 20788525

INTRODUCTION

In the following report we will present the outcome of the test made on three different algorithms that attempt to solve the vertex cover problem. The three algorithms employed include:

- SAT: Utilizes the MiniSAT Satisfiability solver by converting problems to CNF form.
- VC_1: Chooses vertex of the highest degree, adds it to the vertex cover and removes all edges associated with that vertex. Repeats until no edges remain.
- VC_2: Chooses any vertex, adds both edge vertices to the vertex cover and removes all edges associated with that edge. Repeats until no edges remain in the graph.

ANALYSIS AND GRAPHS

All tests were done using a multithreaded program where no memory is being shared amongst threads and locks are implemented to ensure appropriate execution takes place. These were done on the same system with the same inputs in order to ensure consistency amongst results.

Two types of tests were applied in this project. First we did running time tests using "pthread getcpuclockid()" in order to get each thread's execution time. Secondly, we compared the minimum vertex cover generated by each algorithm. We did five tests for each number of vertices per graph in ascending order from five to twenty in increments of five.

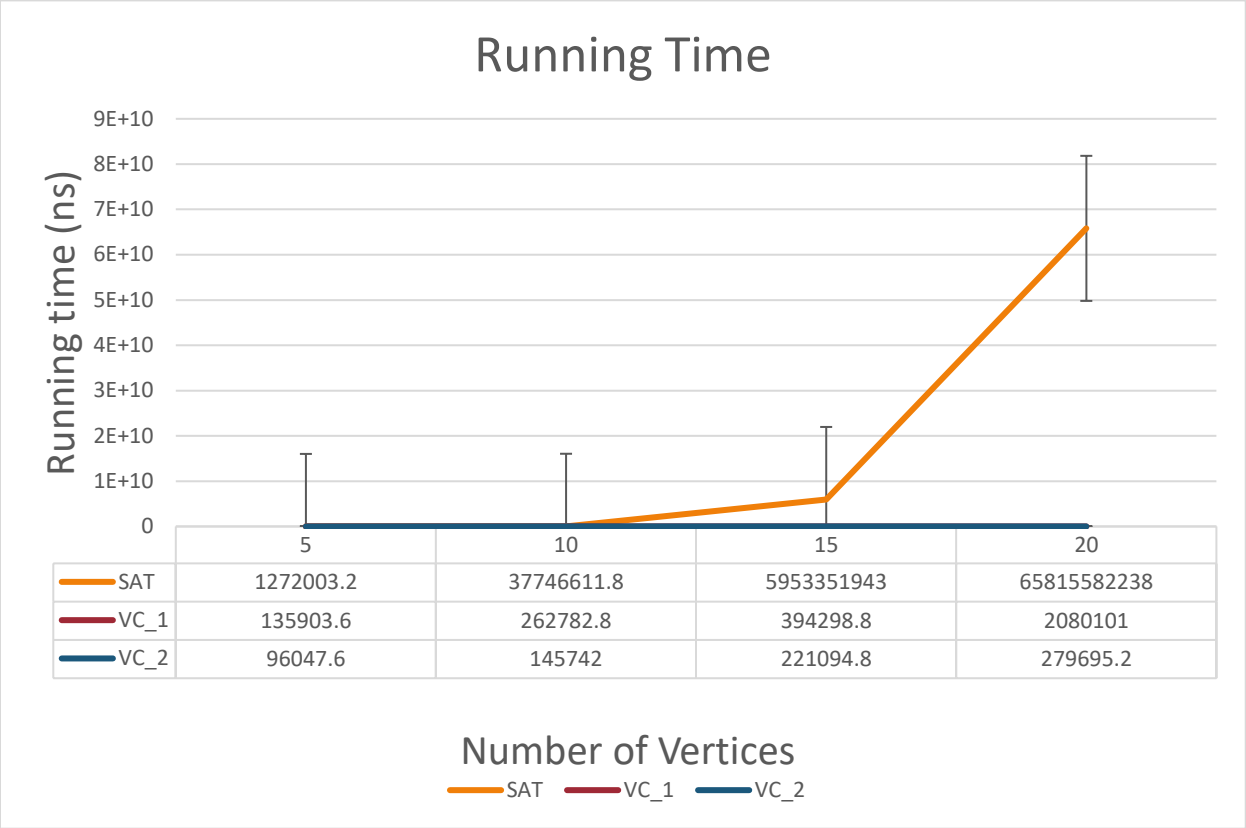

Figure 1

Figure 1 above depicts the average running time for the three algorithms. As shown by the graph, the SAT algorithm is significantly slower than the other two algorithms. At the value of 15 vertices there is a sharp increase in the total running time required for this algorithm to generate the minimum vertex cover for the graph provided. This is due to the fact that the SAT searches for the correct solution as opposed to an approximation which is sough by the other two algorithms. As a result, the more vertices and edges introduced into the graph, the number of calculations to be performed to solve the vertex cover problem begins to grow exponentially.

We also noticed that at the value of 20 vertices there was difficultly computing the minimum vertex cover given the set of edges of the graphGen. That is, when we surpassed a set of greater than twenty-three edges, most of the time, we were unable to get a solution within a reasonable time. By reasonable time we are referring to approximately ten minutes in duration.

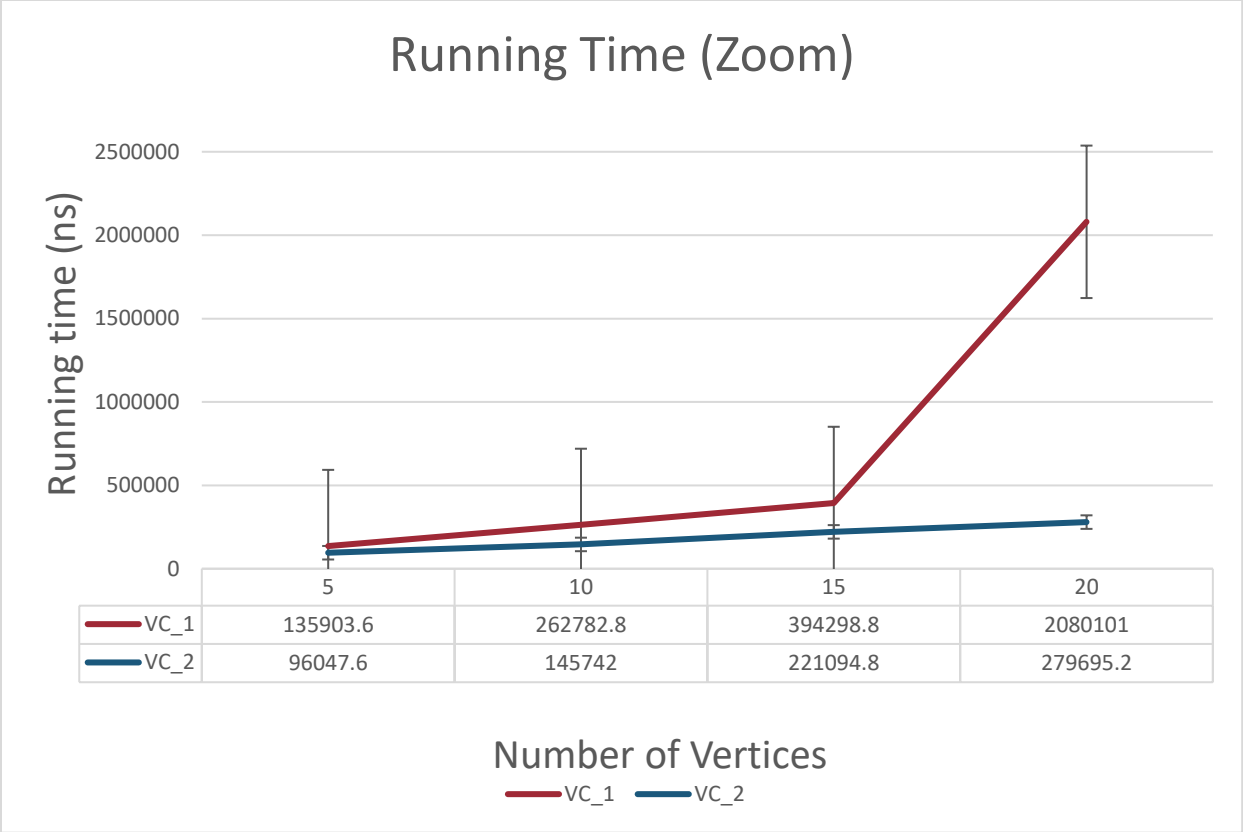

Figure 2

In figure two we can better see the performance difference between the two approximation algorithms. At the 15 vertex mark there is an evident spike for the VC-1 approximation algorithm versus the VC-2 algorithm. This occurs due to the fact that VC-1 algorithm is required to tabulate the most incident vertex during each and every iteration of the algorithm. Unlike VC-1, VC-2 simply selects an edge at the beginning of the iteration irrespective of whether or not the vertices in that edge have the highest degree. This computation requires an iteration of the graph over each edge which remains in the graph at the given time. Thus, as the number of vertices begins to grow, the amount of time required for VC-1 to complete its computations also grows relative to the VC-2 algorithm. Particularly, the growth relative to VC-2 starts to tend towards exponential as the number of vertices in the graph begins to increase. This is most evident in the range between 15 and 20 vertices shown on the graph.

It should also be noted that due to the implementation approach of VC-2, the number of iterations is constantly based on the number of edges in the graph. This is based on the fact that it selects a new edge in constant time for each iteration and then removes all edges associated with that edge in linear time. Thus, the overall time complexity of the VC-2 algorithm remains linear according to the input given by the graph.

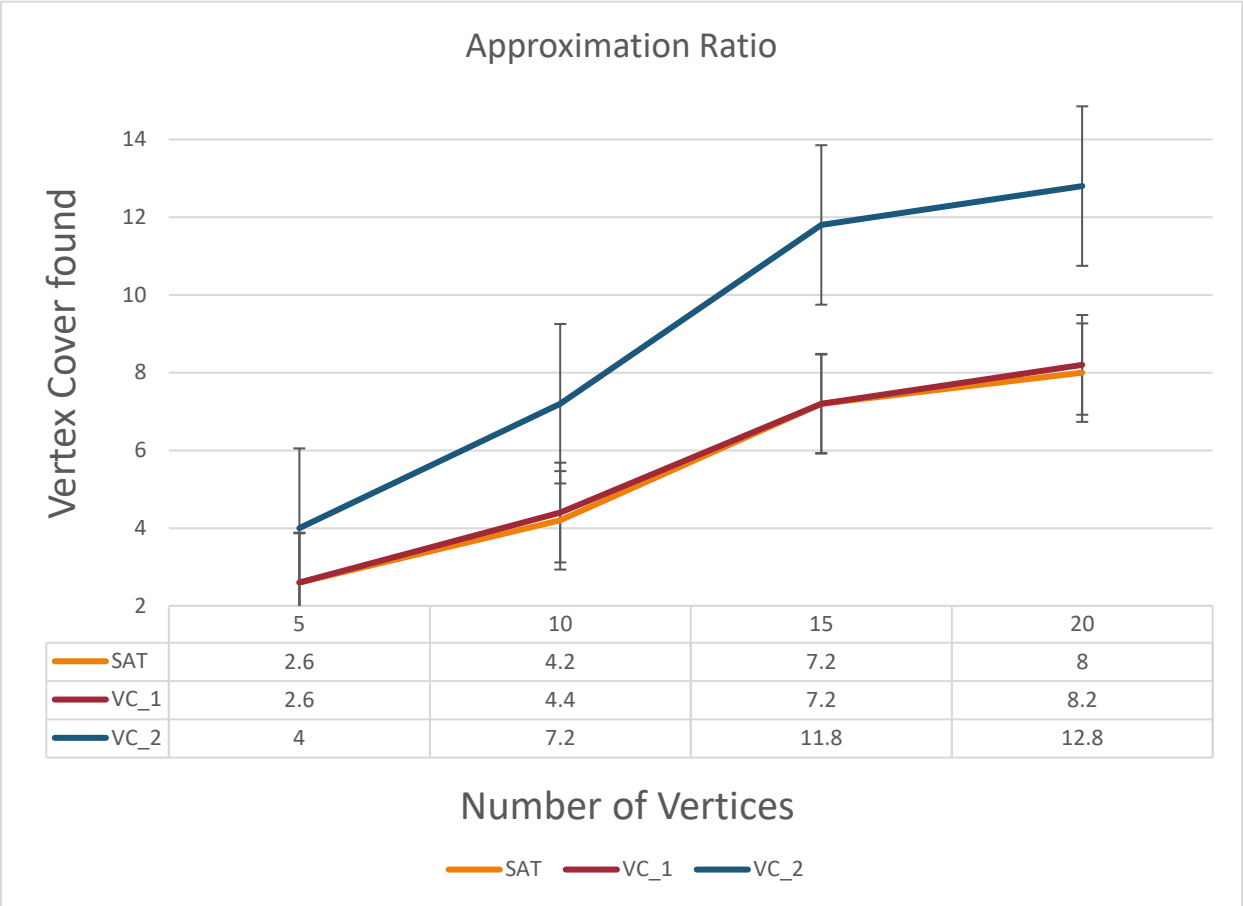

Figure 3

As can be seen from the graph, the overall best approximation algorithm is VC-1. VC-1 has shown to be significantly more accurate than VC-2 as it relates to finding the minimum vertex cover when provided the same graph. With smaller graphs, VC-1 even gives the same or comparable vertex cover solutions to that of the SAT algorithm which always gets the minimum vertex cover of the graph after its computations.

It should be noted however that though VC-2 does not produce a vertex cover as minimal as either the VC-1 or SAT algorithm, it does manage to give a respectable vertex cover given the amount of time it takes to complete its computations as shown in figure 2 above.

CONCLUSION

Overall, the two most important takeaways between the two approximation algorithms can be stated as follows. VC-1 does an impressive job in capturing similar or equal to the minimum vertex cover on smaller graphs alike the SAT algorithm. More impressive is that it is able to reach its conclusions in only a fraction of the time taken by the SAT algorithm while producing a very respectable vertex cover.

An important takeaway for the VC-2 algorithm is that though it does not produce a vertex cover comparable to either SAT or VC-1 algorithm in most cases, it is on average the fastest of the three algorithms at generating a vertex cover for a given graph. Particularly, as the graph size begins to grow, the speed at which VC-2 is able to produce a vertex cover becomes marginal even when compared to the very fast VC-1 algorithm.