Running Time and Approximation Ratio Analysis of CNF-SAT Solver With Approximation Algorithms for the Vertex Cover Problem

Ganesh Rajasekar,MEng ECE

ECE650 Final Course Project

Email id: g3rajase@uwaterloo.ca,Student ID:20793591

Abstract

In this report as part of the final project, I have analyzed the NP-Complete problem of finding Vertex cover of a given graph using different approaches. We take use of two approximation algorithms and a 3-CNF-SAT Solver to compute the results and compare their running time and approximation ratio and provide a quantitative analysis of each approach to the solve the problem.

Keywords: Vertexcover Problem, CNF-SAT Solver, NP-Completeness

1. Introduction

We define Vertex cover of a graph as the a set of vertices in a graph such that each edge is incident to at least one vertex of the set. This problem of computing such minimum vertex cover of any graph is an NP-hard optimization problem. In this project we firstly solve the minimum vertex cover problem using a 3-CNF-SAT Solver by polynomial reduction of the problem. Then I have developed two approximation algorithms to find the minimum vertex cover. The Minisat Solver used in the first case always provides us with an efficient and correct minimum sized vertex cover but the approximation algorithms may not always provide us the minimum sized vertex cover. The following sections will present the Implementation,results with graph and explain in detail the running time and the approximation ratio in various approaches.

2. Implementation

Firstly to solve the vertex cover problem using 3-CNF SAT-Solver we use the polynomial reduction from Assignment4 material. I have utilized multi-threaded programming in which 4 threads are used- One for I/O, One for CNF-SAT solver and the other two for two approximation algorithms. The results obtained were cumulatively collected and analyzed using Microsoft Excel and the analysis of the results is discussed in the below sections.

2.1. Hardware Utilized

The Entire Experiment was run on a laptop with an Intel(R) Core(TM) i5-3320M CPU @2.60GHZ With 8GB RAM and a 64-Bit Ubuntu Operating System. The performance and sub-factors influencing it might vary with a different computer configuration.

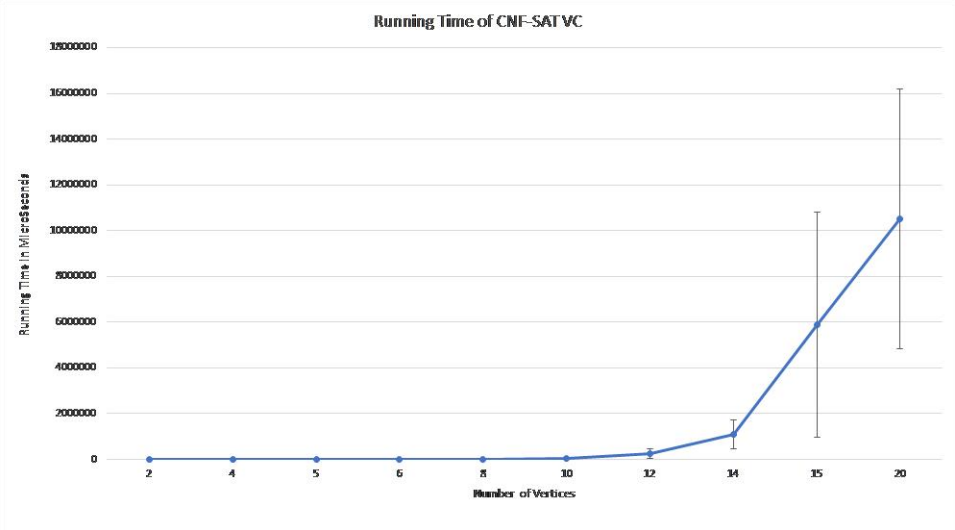

Figure 1: Approximation Ratio of APP1-VC and APP2-VC with CNF-SAT-VC

2.2. Running Time of CNF-SAT-VC

Upon analyzing all the graphs for running time the CNF-SAT-Solver takes the maximum amount of time among all approaches. In order to explain the running time I show two set of graphs. One for SAT-solver (Figure 1),Running time of both Approximation 1 and Approximation 2 Algorithms(Figure 2).I have plotted the average running time of graphs with vertices [2,..20] using Microsoft Excel Line Plot.For calculating the running

time pthread function was utilized. Each graph of size V from [2 to 20] was run for ten times and then ten such graphs for each size of V was run and the mean and standard deviation was calculated. Hence we have around 100 samples for each value of V from 2-20.

As we see from the graph the CNF-SAT solver performs very fast on smaller sized vertices but on increasing the size of vertices the graph begins to rise rapidly growing and is exponential growth from vertex above 13-15.The graph will steadily rise exponentially as vertices size is more. It is to be noted that the local CPU timed out in certain cases where more than vertex 20 was provided. This is due to the fact that in our reduction the number of clauses is also increased. We have more combination of clauses to satisfy and our SAT solver needs to check individually if they satisfy or not.

We can also see the Error bars, which represent the standard deviation of the data is increased greatly for vertices above 15. This means that the running time calculation for them varied greatly for different graphs of the same vertices. This unpredictable behaviour is seen in the error bar.

2.3. Running Time of APP1 VC and APP2 VC

There are two approximation algorithms adopted. The first one APP1 VC methodology involves always picking the highest degree vertex and remove all the edges incident on that vertex. Keep doing this until all edges related to the graph is removed.The second one APP2 VC methodology involves picking an edge(u,v) and remove all the edges incident on the u,v in the graph. We keep picking edges until all edges are removed in the graph.

We find from Figure 2 that both approximation algorithms have a running time which increases with the size of the vertices. However we don't see an exponential increase as compared to the SAT solver running time. The Running time of both algorithms is still polynomial with the size of the Input(Vertices). We see that APP1 VC takes slightly more time than APP2 VC. However the performance of APP1 VC can be further improved with the implementation of a Heap data structure which always provides us with the maximum degree of vertices in constant time.

The SAT solver however needs to look for all possible combination to find a solution. However here we don't do that in the case of these two algorithms. The time complexity of APP1 VC can be represented as $O(V^2)$. As I need to check for each value of V to check for highest degree it takes V^2 comparisons. For this purpose I use an adjacency matrix-using vector of vectors in cpp and keep checking the adjacency matrix for highest degree. So every time I iterate over the entire matrix(column) to find the highest

degree and then loop over the particular row value of v to find its neighbors and delete the incident edges. So in worst case its V^2 comparisons. Since the adjacent matrix uses a space of V^2 the space efficiency is also $O(V^2)$.

In the case of APP2 VC I utilize an array and I keep picking an edge and delete all the incident edges on the (u,v) I picked, Here I make V comparisons in the worst case and iterate over the Edges(E). So running time can be written as O(V*E) which again is polynomial to size of the input. The space complexity is also the same as APP1 algorithm. The APP2 algorithm seems to work faster as we removed edges incident on two different vertices each time we pick an edge it reduces the graph much faster than APP1 algorithm. However this method may not give an optimal solution every time which will be discussed in the next section.

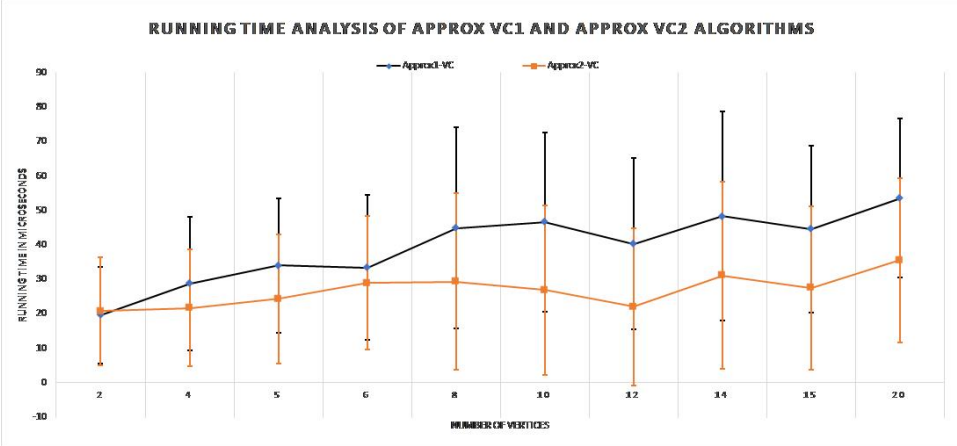

Figure 2: Running Time Analysis of APP1-VC and APP2-VC

3. Approximation Ratio

In order to compare the correctness of these algorithms we take into consideration the approximation ratio. We know for sure that the CNF-SAT VC is always an optimal solution as it checks for all possible solutions and returns a solution. So we take the vertex cover by CNF-SAT as the optimal solution and compare the size of this vertex cover with the vertex cover produced by APP1 and APP2 algorithms. Hence we can rewrite

Approximation Ratio =

$$\frac{SizeOfVertexCover(ApproximationAlgorithm)}{SizeOfVertexCoverSize(CNF-SAT)}$$

We see from figure 3 that the approximation ratio for APP1 VC seems to be very good. The APP1 VC seems to have close to 1 approximation raio which is the best we can achieve. This implies that the APP1 VC gives almost the optimal solution in polynomial time the same given by CNF-SAT in exponential time. However the APP2 VC which is the fastest algorithm to solve our vertex cover performs poorly in giving a optimal solution. We see that in worst case the solution given by APP2 VC is almost twice the optimal solution which is bad in terms of correctness. However we see that in certain cases the APP2 VC has spikes, this is due to the fact that maybe the graph was very well connected that removing an edge removed most of the edges and thus the solution was near the optimal solution. Hence we conclude that APP1 VC has the best approximation ratio over APP2 VC.

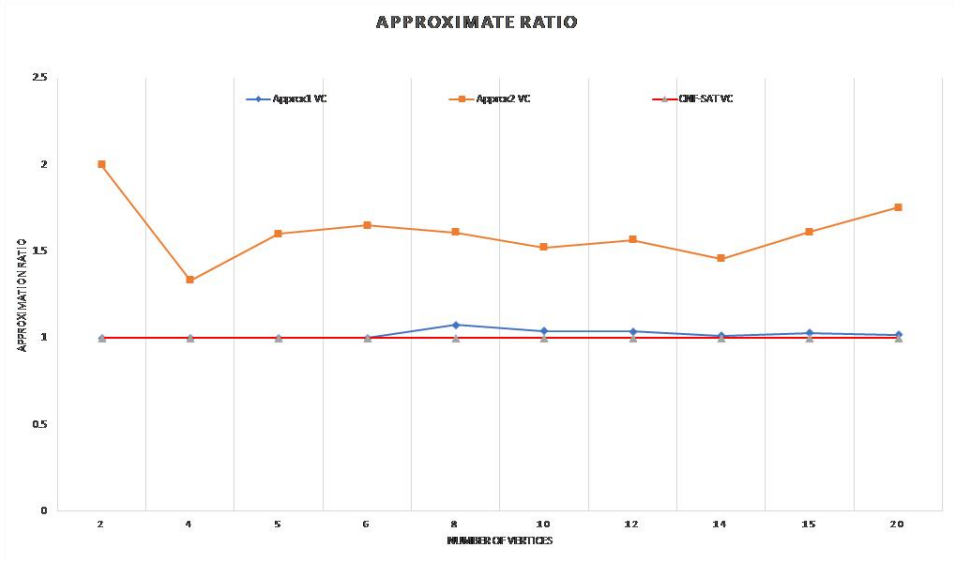

Figure 3: Approximation Ratio of APP1-VC and APP2-VC with CNF-SAT-VC

4. Conclusion

In conclusion we can say for sure that the running time of CNF-SAT solver is exponential with the size of the input. Though it always provides us with the optimal solution it is not efficient in terms of running time when running for larger inputs. When we compare it with the two approximation algorithms we see that one of the algorithms APP1 VC seems to produce almost the same optimal solution while running at a polynomial time. In

such cases we do prefer the approximation algorithms over CNF-SAT but the other approximation algorithm APP2 VC though runs in polynomial time and also faster than APP1 VC does not always provide the desired optimal vertex cover solution. Hence in an expected case we would be tempted to pick APP1 VC to solve the vertex cover problem as it is better in terms of running time but if we need a highly efficient and correct output we should prefer the CNF-SAT VC methodology.