Performance Analysis of Various Vertex Cover Algorithms

ECE 650: Methods and Tools for Software Engineering

MENG- Electrical and Computer Engineering
Nitheesha Reddy Penta Reddy: 20811504
Ankita Kapoor: 20811694

1 INTRODUCTION

The aim of this project is to analyze and compare the performance of the three different algorithms for the vertex cover problem. The goal of each algorithm is to deduce the minimum vertex cover of the given graph G(V,E) where V denotes the number of vertices and E represents the edges between the nodes. The first Algorithm that is CNF-SAT, relies on the conversion of polynomial-time reduction form vertex cover to CNF-SAT i.e. Conjunctive normal form problem. The second technique named Approx-1 picks up the vertex with the highest number of incident edges i.e. the most effective vertex and removes all its connecting edges. This will go on until there is no remaining edge. The third and the last technique named Approx-2 picks up the edge and adds both the connecting vertices to the vertex cover. Then it removes all the connecting edges till no edge remains.

So for the comparison, we used two efficiency measurement units:

1) Running Time - Time taken by each algorithm to calculate the minimum vertex cover.

2) Approximation Ratio - Ratio of the size of the computed vertex cover to the size of optimal minimum vertex cover.

To see the measurements on the console, the user has to add a "-calc" command-line argument with the name of the executable file while running it. The graphs representing the results were plotted using Gnuplot.

2 ANALYSIS

2.1 Experiment Details

The analysis program is based on multi-threading technique. Four threads were used in total which are as follows:

- I/O thread: Used for fetching the input from the user in the form of vertices and edges and distributing that to the other threads.
- CNF thread: Uses the CNF-SAT algorithm to find the minimum vertex cover.
- APPROX1 thread: Uses the approx1 algorithm to calculate the optimal minimum vertex.
- APPROX2 thread: Uses the approx2 algorithm to find the optimal minimum vertex.

The running time of each thread was calculated at the end of each thread using the function of C++ i.e. pthread_getcpuclockid(). The program was tested on the ecelinux4.uwaterloo.ca server. The machine on which the analysis was carried out is Intel Core i5 Multicore-processor, 2.7GHz, 8G of RAM, and its OS is Ubuntu.

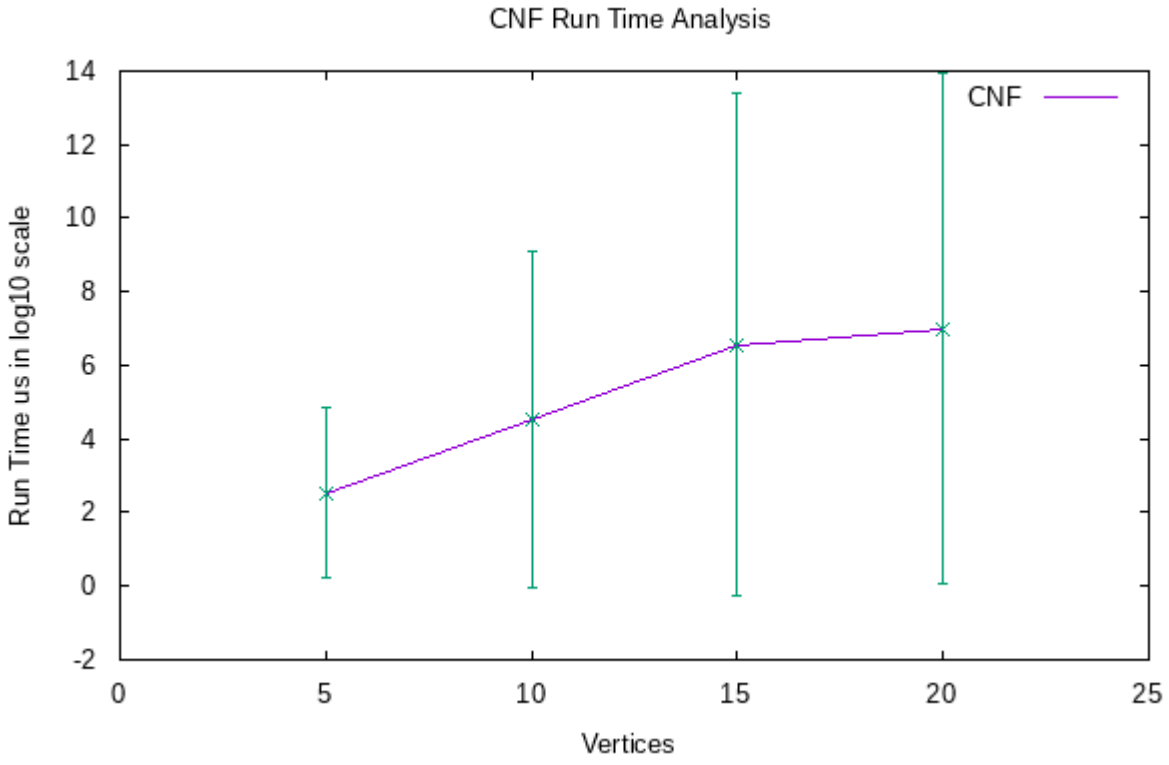

Figure 1: Run time analysis of CNF-SAT in interval [5,20]

2.2 Run Time Analysis

2.2.a CNF-SAT Run Time Analysis

According to our analysis, the CNF-SAT algorithm takes the maximum running time for computing vertex cover when compared to all the other algorithms. We have taken the vertices as the increment of 5 as the input within the interval of [5,20]. We have restricted our analysis up to 20 vertices because CNF-SAT takes a lot of time to calculate vertex cover for larger vertices.
On the other side, the CNF-SAT algorithm takes very less running time for the fewer vertices, especially less than 10 but it shows an exponential rise in the running time when number of vertices becomes greater than that. The reason is that polynomial-time reduction used in sat solver calculates the number of clauses based on the vertices of the input graph and is given by:

$$k + n(k/2) + k(n/2) + |E| \text{ where } n = |V| \text{ in } G \quad (1)$$

Figure 1 shows the CNF-SAT running time analysis plot of vertices ranging from 5 to 20. So, as per the above equation, the number of clauses increases exponentially with the increase in the number of vertices. Therefore, we have an exponential increase in the running time of the algorithm after 10 vertices. For the standard deviation, this graph does not represent standard deviation for smaller vertices because running on larger vertices suppresses the running time on smaller vertices. Although there is a very small standard deviation for smaller vertices that can only get enlarged when there is more number of vertices in the graph.

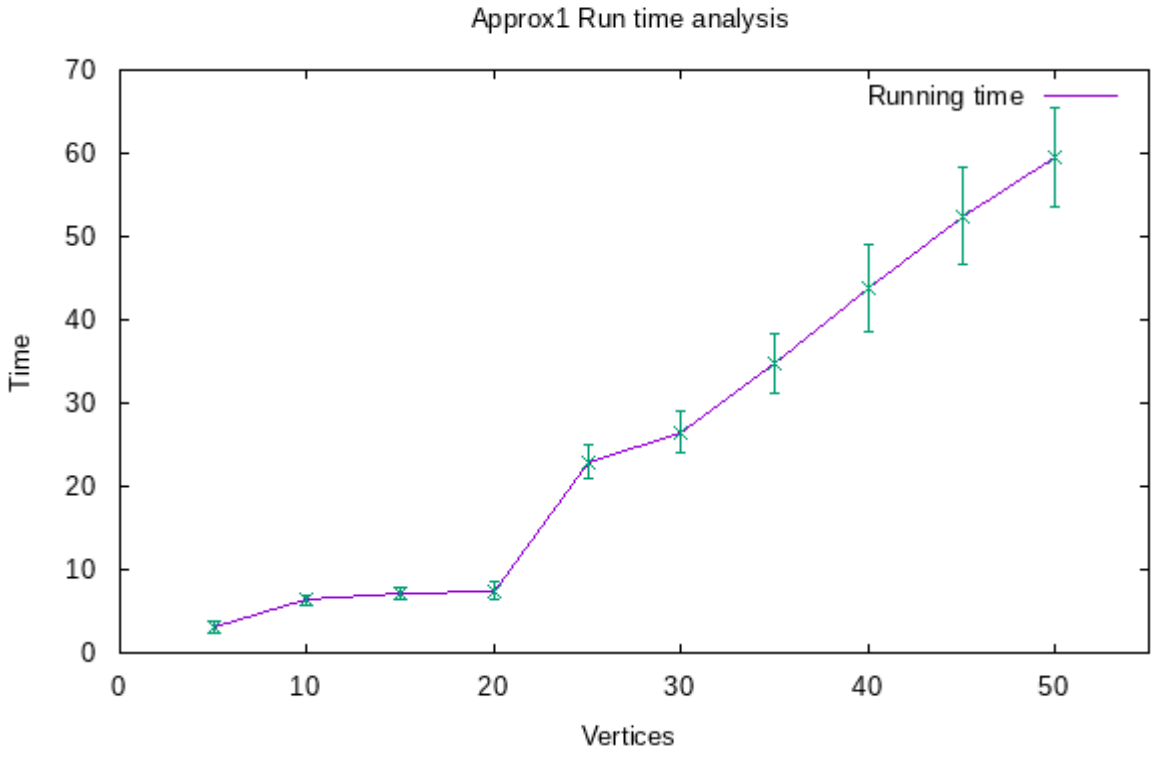

Figure 2: Run time analysis of Approx-1 in interval [5,50]

2.2.b Approx-1 Run Time Analysis

The above Figure 2 represents the running time analysis of the Approx-1 algorithm as a function of the number of vertices. The vertices are taken in the range from [5,50] with an increment of 5. Though this algorithm is an approximation algorithm that may or not give the minimum vertex cover but provides us with the optimal solution. It can be clearly seen from Figure 5 that Approx-1 takes less time as compared to the CNF-SAT for the same inputs. So this means that Approx-1 will provide us with an efficient output but CNF-SAT will give us the correct output.

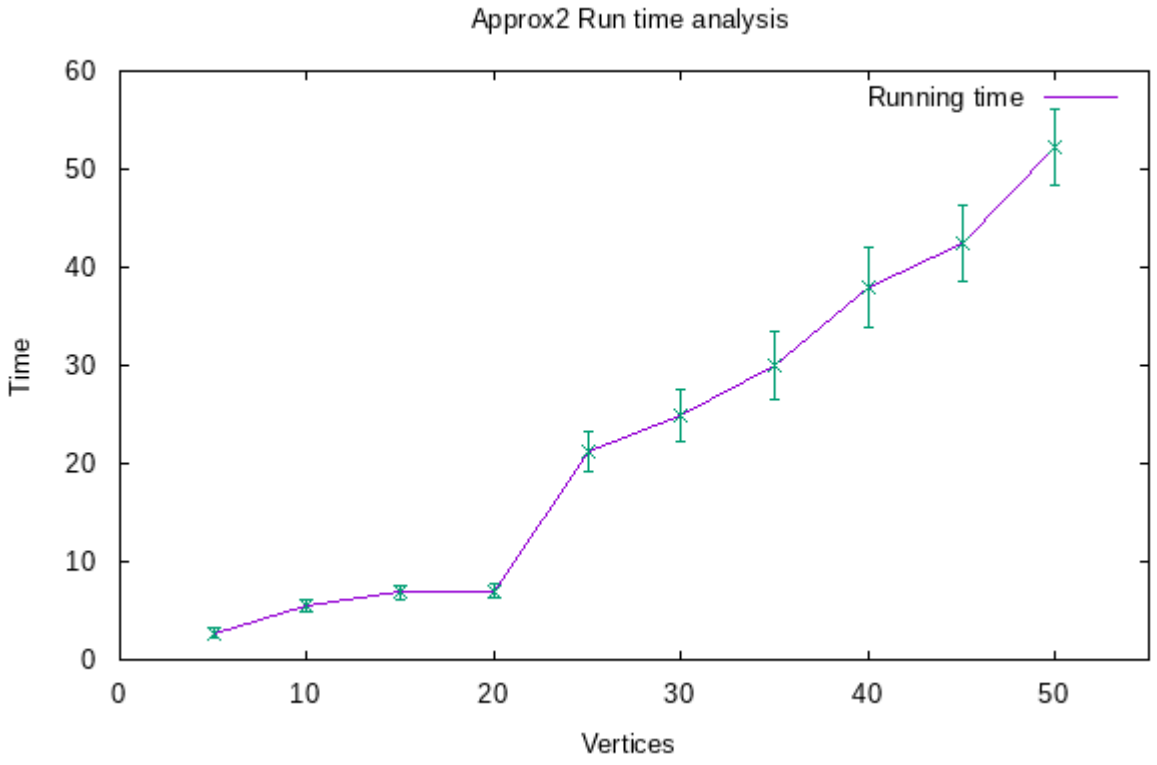

Figure 3: Run time analysis of Approx-2 in interval [5,50].

2.2.c Approx-2 Run Time Analysis

The running time analysis of the Approx-2 algorithm as a function of the number of vertices is depicted in Figure 3. Vertices within the interval of [5,50] are taken with an increment of 5. This algorithm is also an approximation algorithm similar to Approx-1. Also, we can see from Figure 5 that Approx-2 takes lesser time than CNF-SAT algorithm and Figure 4 clearly depicts that Approx-2 takes less time for computation as compared to Approx-1 algorithm for the same inputs even for the vertices after 20. There is some contraction here that theoretically Approx-1 should perform better than Approx-2 because Approx-2 will pick edge at a time whose complexity is $O(|V|^2)$ than the vertices in the case of Approx-1 which is $O(|V|)$. This happens due to our implementation of Approx-1. What we do in that is after picking a vertex of the highest degree we remove it from graph and adjust degrees of all vertices ($O(|E|)$) and then find the vertex with highest degree ($O(|V|)$). Due to which additional overheads come into place and make Approx-1 less efficient than Approx-2 in our project.

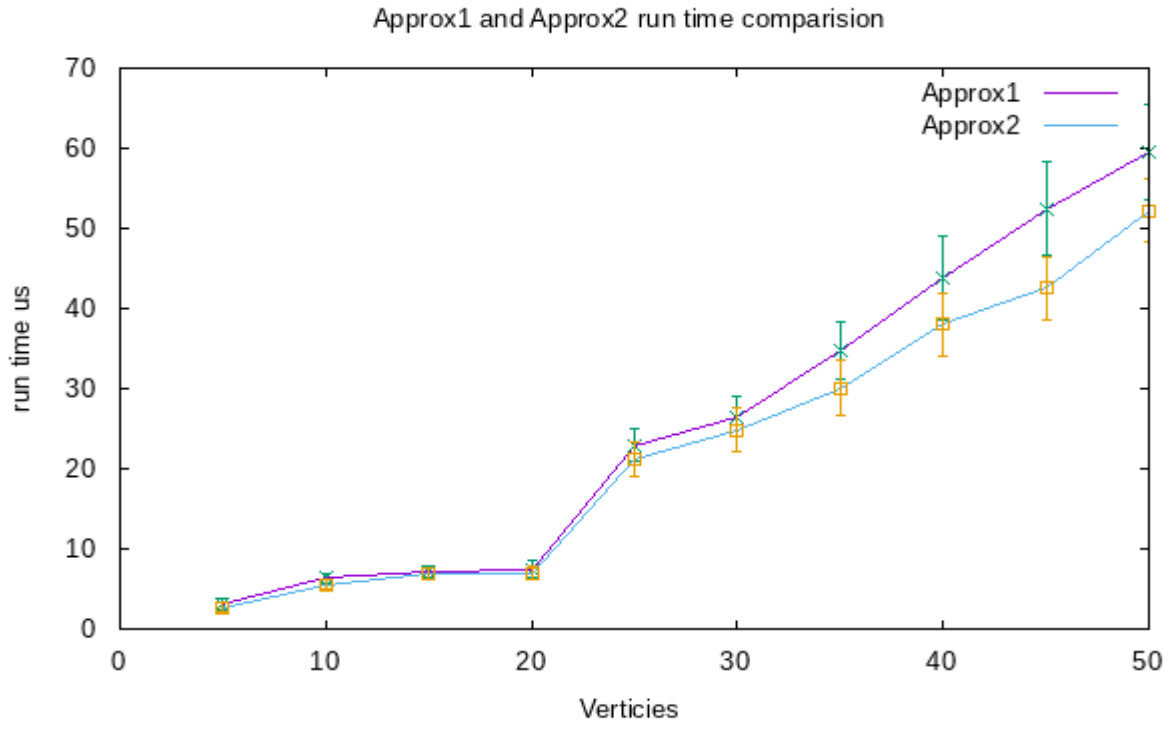

Figure 4: Run time comaparison of Approx-1 and Approx-2 in interval [5,50]

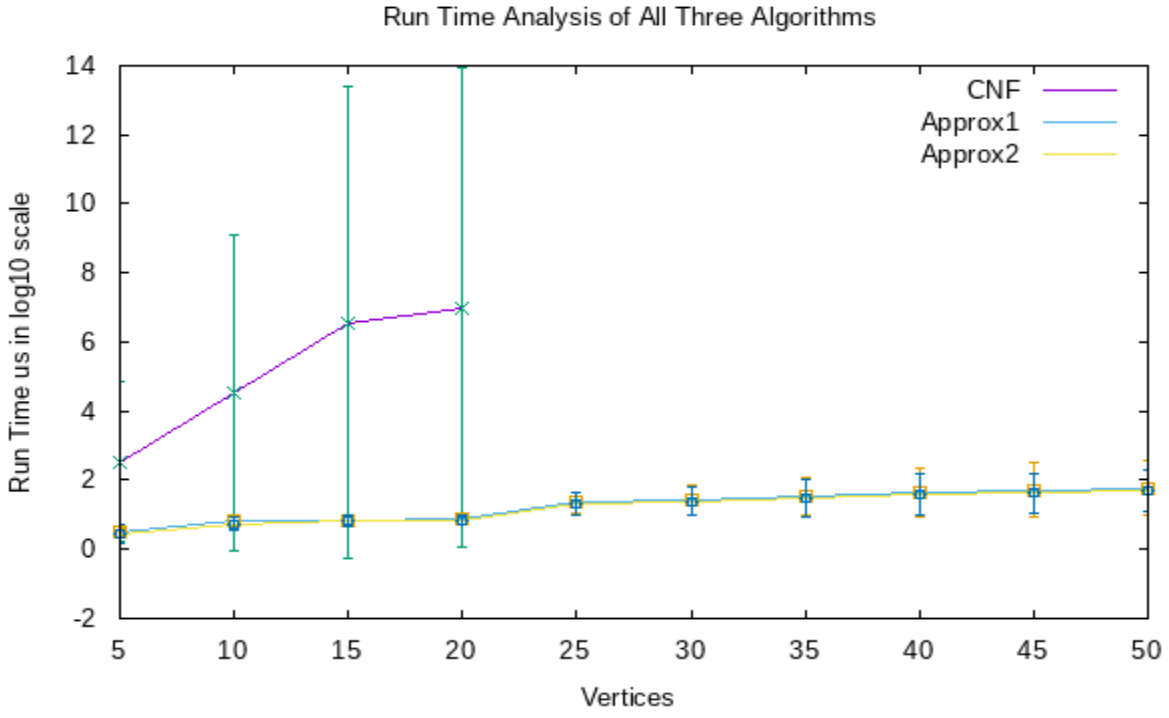

Figure 5: Run time comparison of all three algorithms in interval [5,50] (exception CNF is in [5,20]) with $\log_{10}$-scale on Y-axis

2.1.d Comparison of the three Algorithms

For the comparison of the 3 different algorithms, we plotted the running time which is taken as a log scale on the Y-axis with respect to the number of vertices in the interval of [5,0] with the increment of 5 on the X-axis. From Figure 5, we can see that CNF-SAT takes much more time as compared to Approx-1 and Approx-2. Also, CNF-SAT follows an exponential increase of running time with the increase in the number of vertices but this is not the case for other two algorithms. In order to satisfy every clause for minimum vertex cover, the CNF-SAT traverses the whole solution space and searches for all possible assignments to the given CNF. The number of clauses and time taken by SAT solver increases in an exponential trend with the increase in vertices. While on the other hand, both Approx-1 and Approx-2 algorithms have less time complexity than the CNF-SAT. Therefore, they will never show an exponential rise in their running time as the CNF-SAT.

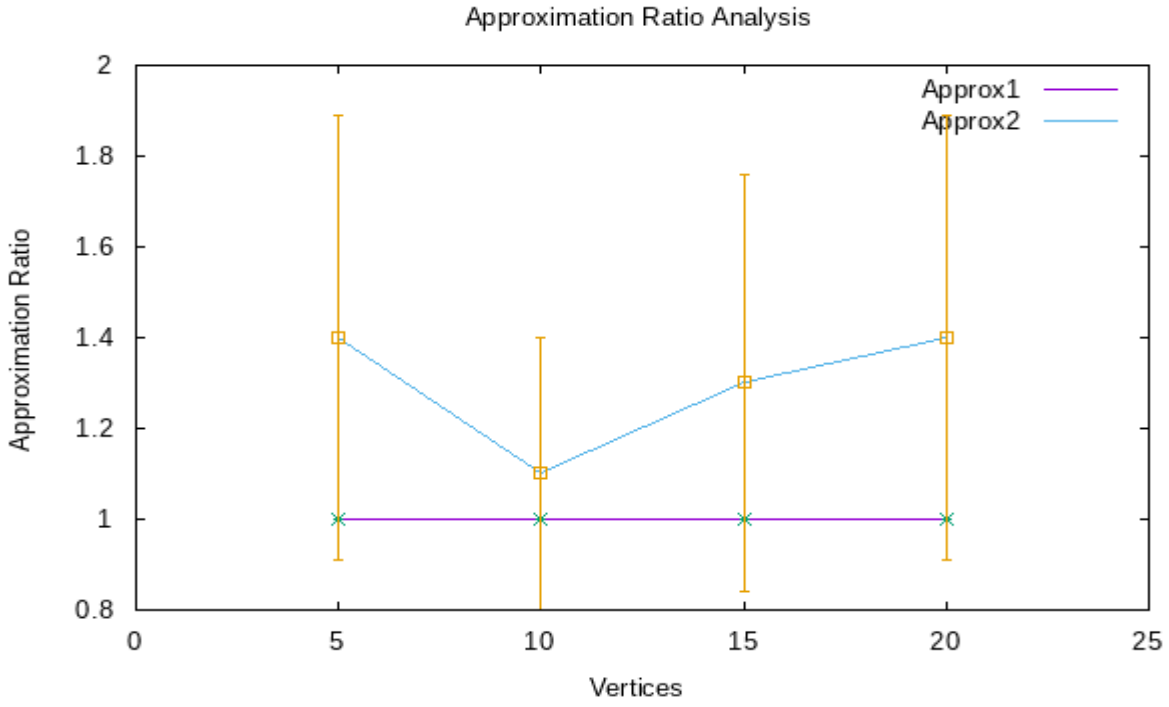

Figure 6: Comparing Approximation Ratio of Approx-1 and Approx-2 in interval [5,20].

2.3 Approximate Ratio Analysis

Approximation ratio can be defined as follows:

$$\text{Approximation Ratio} = \frac{\text{Size of Computed Vertex Cover}}{\text{Size of Minimum Vertex Cover}} \quad (2)$$

So from equation 2, we can say that the smaller the approximation ratio, the more optimal will be the output of the algorithm. Also, Figure 6 clearly depicts that the approximation ratio for all of the vertices in Approx-1 is 1. On the other hand, the approximation ratio is around 2 for Approx-2 which is higher than the Approx-1. This reason behind this output is that the Approx-1 algorithm picks up vertices with the highest degree whereas Approx-2 picks up one edge at a time. So in Approx-1 often gives us the minimum vertex cover especially for smaller vertices which was found to be equivalent to the output of CNF-SAT.

Vertices	MeanAPR Aprx-1	MeanAPR Aprx-2	SDevAPR Aprx-1	SDevAPR Aprx-2
5	1	1.4	0	0.489
10	1	1.1	0	0.3
15	1	1.3	0	0.458
20	1	1.4	0	0.489

Moreover, Approx-2 usually provides vertex covers which is greater than the minimal vertex covers provided by the CNF-SAT algorithm. The standard deviation in Approx-2 implies to the randomness aspect of the algorithm and may vary after each run.

3 CONCLUSION

From the above-plotted graphs, it can be concluded that all the three algorithms can be used to solve the vertex cover problem but keeping in mind that their efficiency may vary on the basis of running time and approximation ratio.

- In terms of running time, Approx-2 is better as compared to CNF-SAT for higher number of vertices, but it fails to get minimum vertex cover as we have seen that the approximate ratio for Approx-2 was quite high. So it is a better way for computation of vertex cover but not for finding the minimum vertex cover.

- In terms of approximation ratio, CNF-SAT definitely gets the minimum vertex cover, but it fails for the larger inputs because of the running time which rises exponentially. So it requires more time than the other two algorithms. Therefore, it is better for small scale vertex problems.

- In general, Approx-1 is a better approach to solve large scale vertex cover problems as its running time is less than CNF-SAT and Approx-2 and it sometimes provides the optimal minimum vertex cover.