Analysis of Different Algorithms for solving Minimum Vertex Cover Problem

ECE-650 Project

Report Submitted by:

Jaydeep Singh-20911330
Harkirat Singh Saini-20922368

1.Abstract

A vertex cover of a graph G is a set of vertices, V, such that every edge in G has at least one of vertex in V as an endpoint. A minimum vertex cover is defined as a vertex cover of smallest possible size. The problem of finding a minimum vertex cover is a classical optimization problem in computer science and is an NP-hard problem. In this report we do an analysis of different algorithms for solving the minimum vertex cover problem by comparing their running time and approximation ratio.

2.Introduction

In this project, we use the following algorithms to solve the minimum vertex cover problem:

- **CNF-SAT-VC**
  In this approach, we do a polynomial time reduction of the given vertex-cover problem to CNF-SAT. Thus, we transform the given instance of vertex cover problem to a formula propositional logic formula F, and we provide F as an input to a SAT solver. The true/false answer from the SAT solver is the answer to the instance of vertex cover problem.

- **APPROX-VC-1**
  In this algorithm, we follow a greedy approach where we pick a vertex of highest degree (most incident edges). Then we add it to our vertex cover set and throw away all the edges incident on that vertex. These steps are repeated till no edges remain. The final set of vertices in the vertex cover when there are no edges in the graph is the answer to given problem.

- **APPROX-VC-2**
  In this approach, we randomly pick an edge <*u, v*> and add both u and v to our vertex cover. Then, we throw away all edges attached to u and v. This process is repeated till no edges remain. The set of vertices in our vertex cover after all the edges are removed is the answer to given instance of the problem.

In this report, we consider CNF-SAT-VC to be an optimal solution and consider it as a benchmark for analysis of other two approximation algorithms. The report analyses the performance of these algorithms by comparing the running time and approximation ratio.

---

3. Analysis Methodology

The experiment has been performed using a multithreaded C++ program. The program runs 4 concurrent threads, one for the I/O and three threads for each of the algorithms. The I/O thread takes a set of vertices and edges as the input and converts the graph to its adjacency list representation which is further used by all the algorithms. Following are the main points of the implementation:

- **SAT-solver**
  The experiment uses a custom implementation of Minisat solver available at https://github.com/agurfinkel/minisat. This minisat solver is used by the SAT-CNF-VC approach to solve the given input problem by converting it into a propositional logic formula.

- **Timeout Limit**
  In this experiment, we have set 2 minutes (120 seconds) as the timeout limit for the output. Note that Although CNF-SAT-VC gives us the optimal solution, but the algorithm does not scale well for higher values of V. And In our case, for values of V greater than 15, the CNF-SAT-VC takes a large amount of time to compute the result. Thus a timeout limit is necessary in these cases so as to avoid indefinite waiting and the program displays "TimedOut" as output of CNF-SAT-VC.

- **Input Data**
  Input Data for the experiment were the graphs generated by Random graphGen (implementation present at uwaterloo eceubuntu server).10 graphs each for various values of V(no. of vertices), Vϵ [5,15] with increments of 2 were generated for the input data. Each graph was randomly generated with n umber of vertices |V| and a random set of edges.

- **Running Time**
  To measure the running time of algorithms, getcpuclockid function inside the thread utility is used and the running time of each execution is noted. This function gives the average CPU running time for a particular thread. The results are then plotted by computing the average and standard deviation of the running time of the algorithms for each value of |V|.

- **Approximation Ratio**
  The approximation ratio (or approximation factor) of an algorithm is the ratio between the result obtained by the algorithm and the optimal cost or profit. Since our experiment assumes that CNF-SAT-VC is an optimal algorithm, the approximation ratio for our experiment is defined as :

$$Approximation\ Ratio = \frac{SizeOfVertexCover(ApproximationAlgorithm)}{SizeOfVertexCover(CNF-SAT-VC)}$$

  The average and standard deviation of the approximation ratio of various algorithms obtained across various executions are then plotted on a graph and analysed.

4. Experiment Result and Analysis

4.1 Running Time

The following graph shows the average running time and the standard deviation of the CNF-SAT-VC algorithm for different number of vertices. The graph is plotted for different values of |V| in the interval [5,15] in increments of 2. This is because the algorithm does not scale well for larger graphs and thus returns time-out for graphs with vertices value greater than 15. This is because as the no. of vertices increase, the number of literals and clauses in the propositional formula increase, which lead to an increase in the complexity and the computation time.

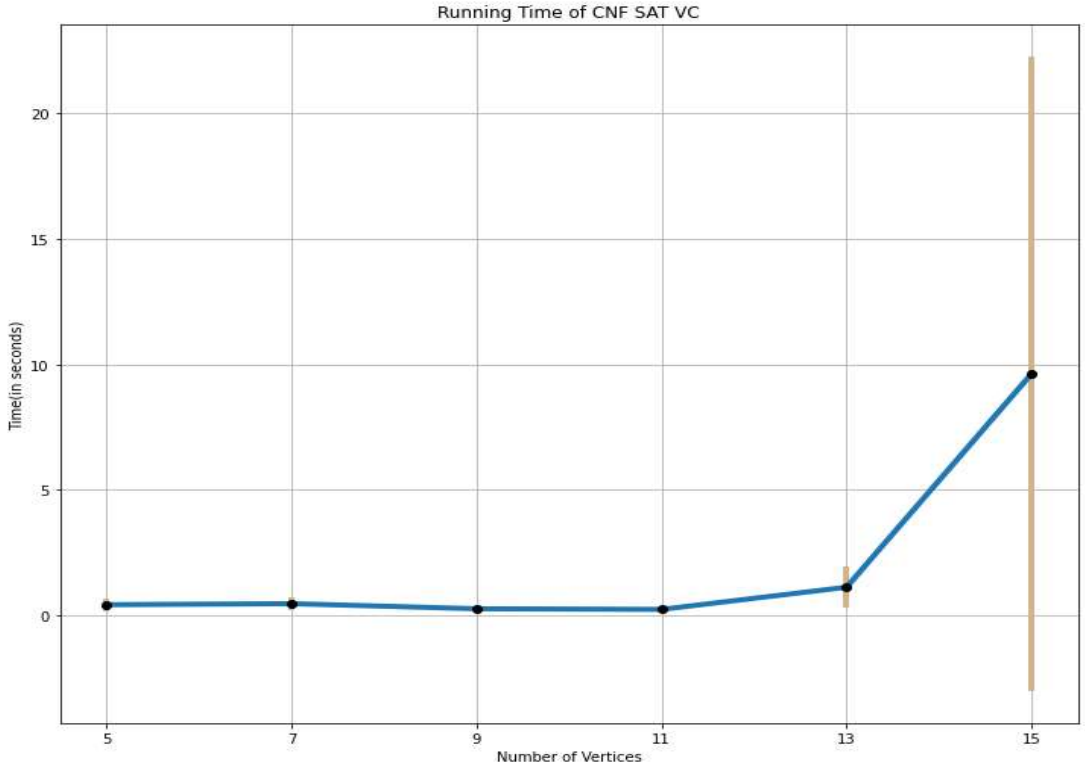

Figure 1: Relationship between running time and number of vertices for CNF-SAT-VC algorithm

It is observed that the CNF-SAT works fast for the cases when number of vertices are small. And the amount of time taken by the algorithm increases exponentially with the increase in the no. of vertices in the graph. There is a steep increase in the time taken for the algorithm to execute as the number of vertices increase from 13 to 15 and on further increase in the vertices, the algorithm gives a timeout. Also note that as the vertices increase, the algorithm gives varied results for inputs with similar number of vertices. This is indicated by a large standard deviation at V=15.

Similarly, the results obtained for the other two algorithms for the values of V are plotted. The following figure depicts the results:

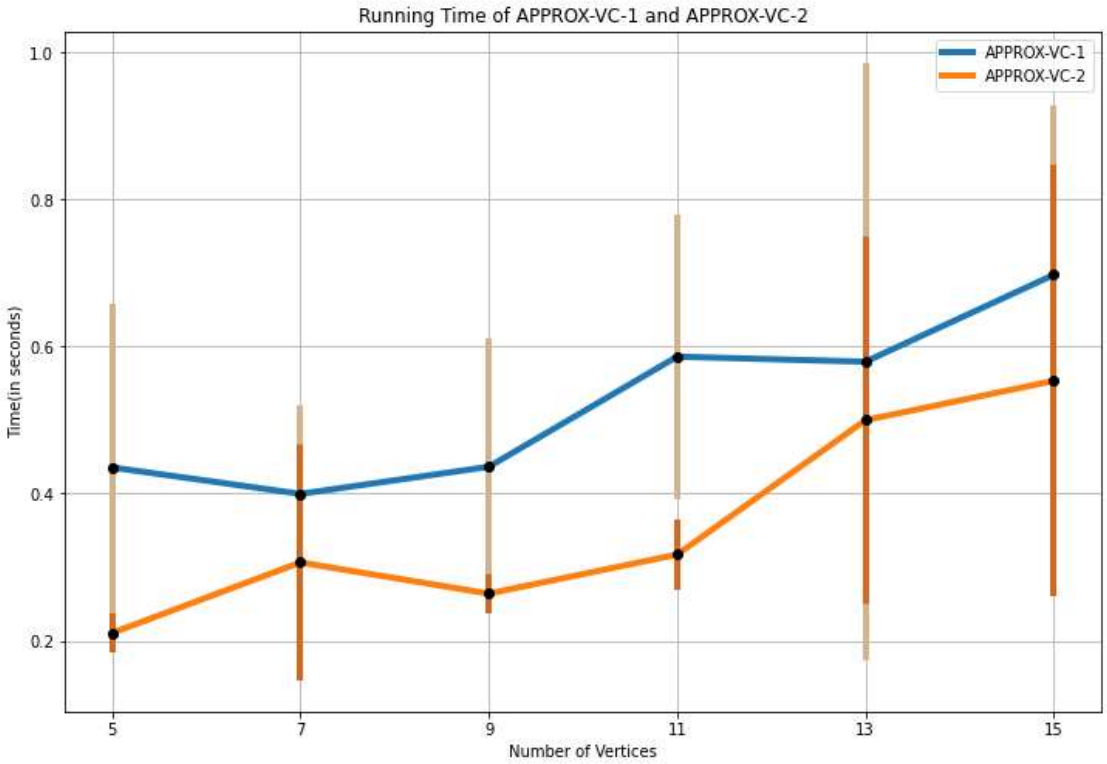

Figure 2: Comparison of the running time of APPROX-VC-1 and APPROX-VC2

Summarising the observations:

- It is observed that both the approximation algorithms APPROX-VC-1 and APPROX-VC-2 perform much faster than the optimal SAT algorithm. This is because SAT approach converts the given problem into propositional literals and clauses and then looks for all possible combinations to find a solution. Whereas both the approximation algorithms solve the problem in polynomial time.

- On an average, the running time of VC-2 is lesser than the running time for VC-1 algorithm. This is because VC-1 involves finding the vertex with the highest degree and then selectively removing particular edges, whereas VC-2 randomly selects the edges and removes them, thus involves lesser computation.

- Both the algorithms, show a linear increase in the running time with an increase in the number of vertices. Note that the increase in the running time is not exponential as it was the case with CNF-SAT approach and these algorithms return results even for V higher than 15. But here in the graph, we are plotting graphs only for values of V <=15 as CNF-SAT gives timeout after that and thus appropriate comparison cannot be done.

4.2 Approximation Ratio

We compare the algorithms on the basis of approximation ratio, considering the CNF-SAT as the optimal result. The following figure shows the comparison of both the algorithms on the basis approximation ratio:

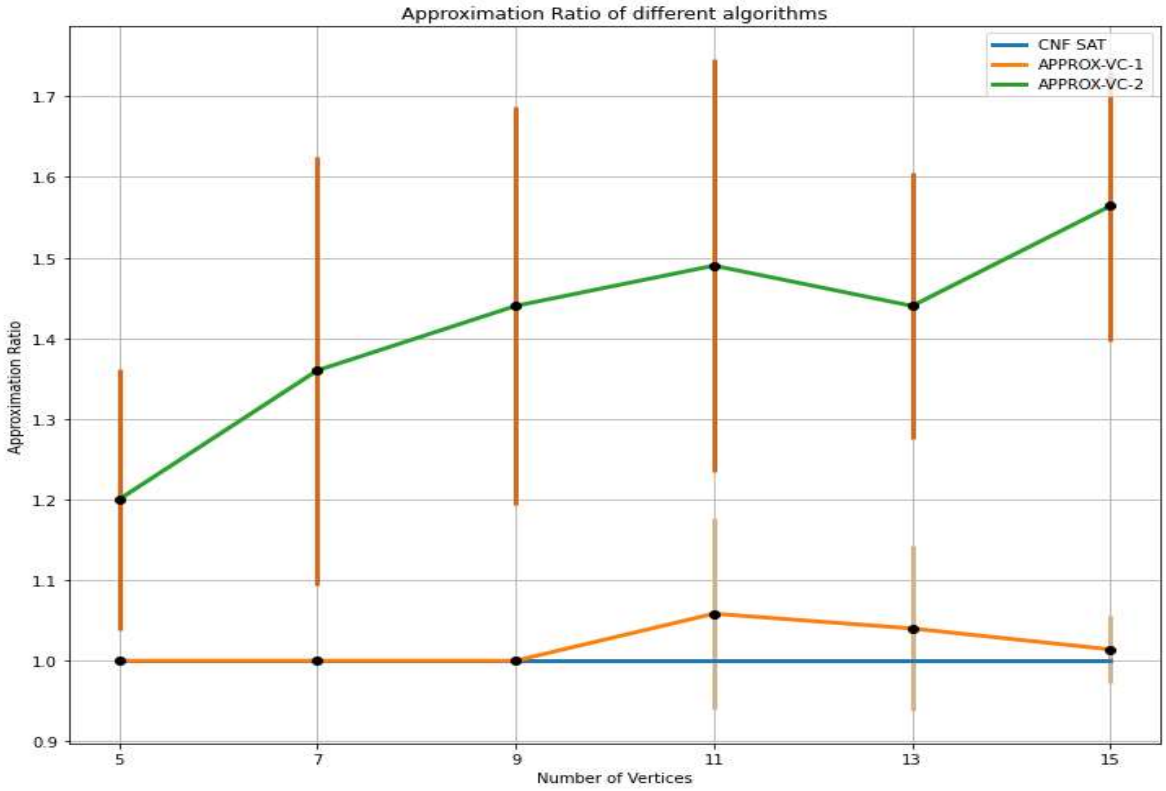

Figure 3: Comparison of the Approximation-ratio of APPROX-VC-1 and APPROX-VC2

Summarising the observations:

- As CNF-SAT is our optimal algorithm, it has the approximation ratio of 1 for all the cases, as indicated by the blue line.

- The results of algorithm VC-1 are very similar to the optimal benchmark performance of CNF-SAT. This is evident from the approximation ratio of VC-1 which is very close to 1 for different number of vertices. Thus, it can be inferred that VC-1 gives us a near-optimal solution in polynomial time for the optimal solution provided by CNF-SAT in exponential time.

- An interesting point to note is that although VC-2 algorithm is much faster than VC-1, but it has a higher approximation ratio. This means that VC-2 algorithm is not able to give an optimal solution in majority of the cases.

5. Conclusion

Analysing the different approaches to solve the vertex cover problem, we observe that CNF-SAT always provides us an optimal solution in all the cases, but the drawback of this approach is that it runs in exponential time and it is unable to provide an answer for graphs with large number of vertices.

The Approach APPROX-VC2 is the fastest of all the three methods, but it fails to give an optimal solution in majority of the cases. On the other hand, although APPROX-VC1 is slower than VC-2, it is able to give us an optimal solution most of the time.

Thus, we can conclude CNF-SAT is the best solution for solving vertex-cover-problem if there are no constraints on the execution time. But if there are time-limit constraints to solve the problem, then APPROX-VC1 is a good alternative as it provides near-optimal solution in polynomial time.