ECE 650 Final Project: Vertex cover timing analysis

Surya Murugaian-20815071 and
Matthew Pitropov-20480877

December 2019

1 Introduction

The vertex cover problem is such that given an undirected graph, find the minimum subset of vertices that are connected through edges to all nodes within the graph. The problem of computing a minimum vertex cover of any graph is an NP-hard optimization problem. The previous assignment's goal was to utilize MiniSat along with the given encoding to create a sat version of the undirected graph. This project builds upon the code of the prior assignment and adds the addition of two approximations that will be able to run at a reasonable speed for graphs with a high number of vertices and edges. It also adds the requirement to use multiple threads to time the speed of each algorithm.

Once after designing the three algorithms, we analyze the performance of the algorithms using the metrics : (1) Running time and (2) Approximation ratio. Here, approximation ratio is the ratio of the size of the calculated vertex cover to the size of an optimal vertex cover, in this case it is given by the CNF-SAT algorithm.

2 Implementation

Initially, We designed all three algorithms to perform independently finding the vertex cover for a given graph. Next, we designed the thread skeleton using 'pthreads'. There are total of 4 threads that need to run concurrently as mentioned in the problem statement. In the main function, the program creates the I/O thread. The I/O thread parses the user graph input pairs V, E and initializes the graph with the adjacency list. The threads CNF-SAT-VC, APPROX-VC-1 and APPROX-VC-2 then run concurrently making use of the adjacency list created above. The running time of each thread is captured at the end of each thread using the pthread-getcpuclockid () function for getting the CPU time of each algorithm function.

3 Algorithm descriptions

3.1 CNF_SAT_VC

For this algorithm we have used the same encoding given in assignment 4. The graph is implemented as as a set of literals and clauses to be in CNF form. This is passed in MiniSat which will determine if there exists a vertex cover for the current vertex cover size. Each iteration increases the vertex cover size until the minimal cover is found.

3.2 APPROX_VC_1

This algorithm gives a boost in speed compared to our above encoding and is a better approximation than the second approximation. It is a greedy algorithm that removes the node with the most edges in each iteration. We used an unordered set to store the current nodes in the graph along with a map of vectors that we refer to as an adjacency map. Algorithm [1] is included as our pseudo code for APPROX_VC_1.

Algorithm 1 APPROX_VC_1

```
while vertex set > 0 do
    initialize highest node variable
    for node in adjacency map do
        if node has more higher number of edges then
            set new highest node.
        end if
    end for
    if if highest node has zero edges then
        break out of while loop
    end if
    add highest node to vertex cover
    remove highest node from adjacency map
    for node in adjacency map do
        for node in node vector do
            if node is the highest node then
                remove it from the vector
            end if
        end for
    end for
end while
```

3.3 APPROX_VC_2

This is the simplest algorithm listed and runs the fastest, however, it gives the least accurate result. For this algorithm we used a vector of pairs where each pair represents an edge on the graph. A random edge is selected and removed from the graph. Then each edge is checked and removed if either of the nodes in the random edge are in it. Algorithm [2] is included as our pseudo code for APPROX_VC_2.

Algorithm 2 APPROX_VC_2

```
while graph size > 0 do
    u, v = random edge
    add u, v to vertex cover
    for edge in graph do
        if u or v in edge then
            delete edge
        end if
    end for
end while
```

4 Analysis

For the analysis, we focus on two aspects of performance metrics: Running time and Approximation ratio. In case of running time, the objective is to measure, for various values of V (number of vertices), for the graphs generated by graphGen, the running time and approximation ratio. The graph ranges from V=5 to V=50 with an interval of 5.

For each vertex, we generate 10 set of graphs, computing the running time and approximation ratio of each of them. Finally, calculating the mean and standard deviation across those 100 samples for each value of V. For approximation ratio, we consider the optimal vertex cover is given by CNF-SAT-VC algorithm.

4.1 Running time of CNF-SAT-VC

This approach is considered to give optimal vertex cover for an given graph, G = (V, E) is a subset of vertices C $\subseteq$ V such that each edge in E is incident to at least one vertex in C. This approach uses Mini-SAT solver to compute the vertex cover using polynomial time reduction. We have encoded the reduction based on the document 'a4-encoding' provided to the class for assignment-4. CNF-SAT-VC takes the maximum amount of time among all algorithms. For better insight, we have separated the CNF-SAT-VC plot from other two as the time difference is huge.

As we can see in Figure 1, the running time rises as the vertex count increases. As the number of vertices within the graph increases, the running time of the algorithm begins to

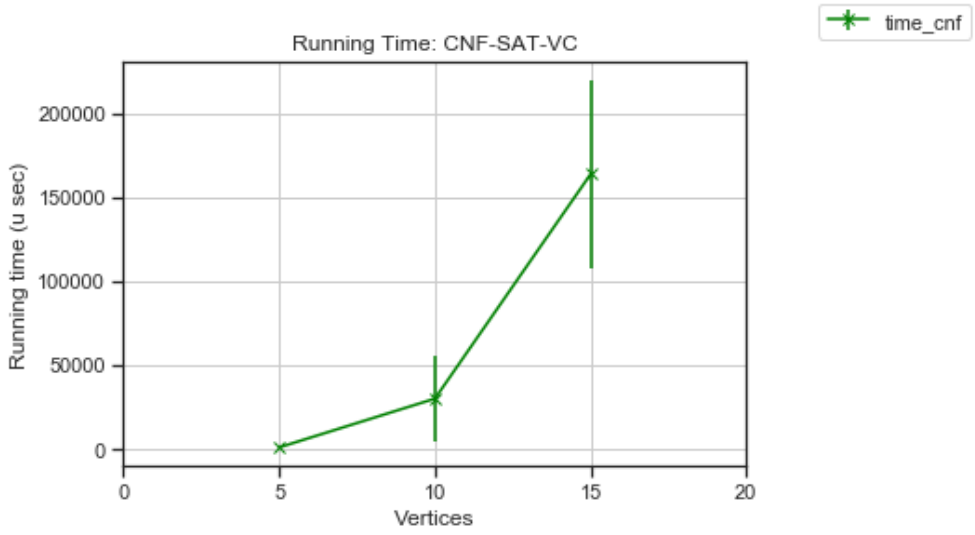

Figure 1: Running time of CNF-SAT-VC

rise rapidly. The graph will steadily rise exponentially with more vertices. To handle the case where the CNF-SAT-VC thread requires an increasingly large amount of time to solve, we introduced timeout of 20 sec, where the thread is stopped if the MiniSat solver consumes more than 20 sec to find the vertex cover for the given V. Eventually, we found that for vertices greater than 15, it gets timed out. This is due to the fact that in our reduction the number of clauses also increased as the vertices increase. This provides more combination of clauses to satisfy.

Also, the error bar that represents the standard deviation is increased greatly for vertex 15. From this we can infer that the running time calculation from them varied greatly for different graphs of the same vertices.

4.2 Running time of APPROX-VC-1 and APPROX-VC-2

In this section, there are two approximation algorithms adopted. As discussed in the section 3, both of these algorithms distinct to each other in the way they pick the vertex cover. In the figure 2,both approximation algorithms have a running time which rises with increase in size of the graph (V). However, we don't see an exponential increase as compared to the SAT solver running time. The running time of APPROX-VC-1 is slightly higher than the running time of APPROX-VC-2, this is due to the fact that APPROX-VC-1 tries to find the highest degree vertex compared to APPROX-VC-2 which tries to pick an edge without restrictions. Thus, time complexity of APPROX-VC-1 is higher than APPROX-VC-2.

4.3 Approximation Ratio

In order to validate these algorithms we consider another important metric called approximation ratio. We have decided that CNF-SAT-VC is an always optimal solution as it checks for

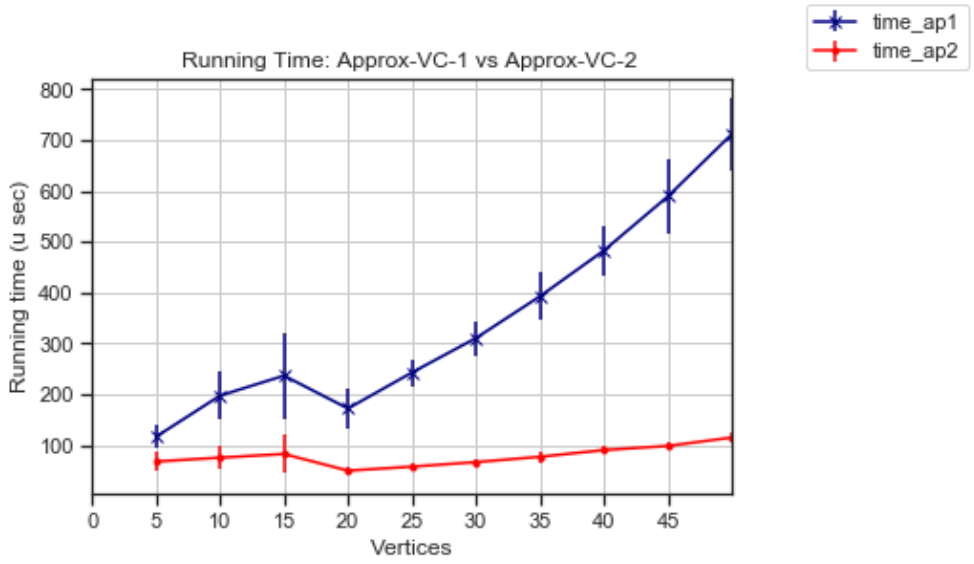

Figure 2: Running time of APPROX-VC-1 vs APPROX-VC-2

all satisfactions. It follows then that the vertex cover of CNF-SAT is the optimal solution and we compare the size of CNF-SAT-VC with the vertex cover produced by APPROX-VC-1 and APPROX-VC-2. So, the approximation ratio is given by,

$$Approx.Ratio = \frac{VetexCover - Size(ApproximationAlgorithm)}{VertexCover - Size(CNF - SAT)}$$

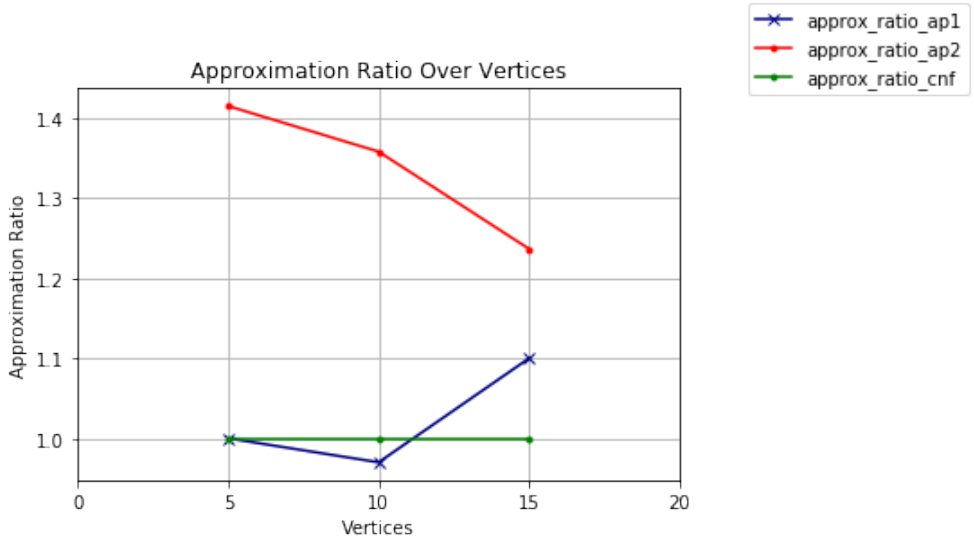

Figure 3: Approximation Ratio

From the figure 3, it can be seen that approximation ratio of APPROX-VC-1 is close to 1, which is the best we can achieve. This implies that the APPROX-VC-1 gives almost optimal solution same given by CNF-SAT. But, we have to note that APPROX-VC-2 is the

fastest algorithm to solve the vertex cover, performing poorly to give optimal solution. The solution given by APPROX-VC-2 is twice the size of optimal solution. Thus, APPROX-VC-1 resembles more of optimal algorithm CNF-SAT for the purpose of vertex cover solution.

5 Conclusion

In conclusion, as a team we were able to successfully implement the two specified approximation algorithms to solve for the vertex cover of a graph. Afterwards, during our analysis of the three algorithms we determined that CNF-SAT-VC was unable to finish it's computation within a reasonable time for large graphs. The first approximation was determined to be more accurate than the second due to selecting the vertex with the highest edge. This is proven through the approximation ratio. One final note is that the first approximation is not perfect and will not always find the optimal solution. The next step for a future project could be improving the CNF-SAT-VC algorithm to improve it's run time speed.