UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 650 - Methods and Tools for Software Engineering

Final Project Report

INSTRUCTOR:
Reza Babaee Cheshmeahmadrezaee

TEAM MEMBERS:
Katayoun Hossein Abadi (20915759)
Erfan Yazdpour (20924364)

April 23, 2021

Contents

1 Introduction

In this report, a comparison between three different algorithms for solving the minimum vertex cover problem is discussed on basis of their running time and approximation ratio. This problem has been classified as an optimization problem under the NP-Hard problems. Given an undirected graph G(V,E), where "V" is the number of vertices and "E" is a set of the paired vertex that is connected by an edge in the graph "G", the vertex cover problem objective is to find the minimum-sized subset of the vertices from graph G in which all edges are covered such that at least one endpoint of every edge of graph G can be reached.

The first algorithm, which will be named "CNF-SAT-VC" in this report, is based on a polynomial -time reduction to CNF-SAT and the use of a SAT solver.

The second algorithm picks a vertex of highest degree (most incident edges), adds it to the vertex cover and throw away all edge's incident on that vertex, and repeats the same instruction till no edges remain. In this report, we will call this algorithm "APPROX-VC-1".

The last algorithm picks an edge <u,v>, and adds both "u" and "v" to the vertex cover. Then it will throw away all edges attached to u and v and repeat the same above steps till no edges remain. In this report, this algorithm will be called "APPROX-VC-2".

2 Execution Environment

The program was tested on the University of Waterloo ECE Linux server. The machine's CPU model is Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, and the OS is Ubuntu. Also, the Linux kernel version is 5.4.0-70-generic.

3 Coding Considerations

The developed program gets input using the standard input and outputs the result using the standard output.

The input format is:

$$> V number\_of\_vertices$$

$$> E\{< vertex\_id, vertex\_id >, < vertex\_id, vertex\_id >, \dots\}$$

Multiple considerations have been applied in the process of implementing algorithms and developing the program.

3.1 Storage Structures

The program has been developed according to the Object-Oriented-Programming concept to increase the cohesion and flexibility of the program. As a result, there is a “Node” Class to keep track of nodes’ behavior and also a “Graph” class to include graph behaviors and functionalities on a set of node objects. Edges are stored as a relation between different instances of the Node class.

3.2 Threads

There are 4 different threads other than the main thread of the program: one thread per algorithm implementation, and one thread for IO-related functionalities. The most important consideration in implementing algorithms in three different threads was to make sure that the only thing that each thread needs to handle is the algorithm itself and there won’t be any overhead processes like unnecessary data manipulation, calculating runtime or approximation ratio, sorting the result, and formatting the output. This is the most important factor that was considered while developing the program since we want a comparison between these three algorithms on basis of runtime which would be seriously affected if this factor was not considered. Some points need to be mentioned about each thread:

- CNF-SAT Thread (Heuristic choice algorithm for "K" using divide and conquer method)
  This algorithm works by choosing a value for the number of vertices and then checks if there is a valid subset using that number or not. The common logic is to choose K from 0 (no vertex) to the total number of vertices that the graph has (the input for V). Regarding the fact that applying this common logic increases the runtime of this thread and this effect can be more tangible for graphs for higher V values, we applied a heuristic choice for assigning different values to K. We have used a concept that came from the divide and conquers problem-solving concept.
  All possible values for K are from 1 (not 0 because we handled that one case differently) to V. Our heuristic choice algorithm will divide this set into two nearly equal parts by assigning K with the middle value in the set:

K = floor (((min value of the set) + (max value of the set)) / 2)

Then the reduction part and SAT-solver will be run with that value of the variable K.

- If the value of K was valid then the result will be stored in a vector after clearing the vector data, and the heuristic choice algorithm will change the max value to K+1 so that we can search for a new K lower than our current answer (searching in the lower group).
- If the value of K was invalid, then the heuristic choice algorithm will change the min value to K+1 so that we can search for a new K higher than our current answer (searching in the upper group).

The process for assigning a new value to K and rerunning the rest of the CNF-SAT algorithm will be the same until we reach the point that the min value gets bigger than the max value.

This heuristic choice algorithm is always eliminating half of the possibilities by eliminating half of a given set of numbers and trying to find a better answer (in this case a valid lower value) until there will not be anything to eliminate.

The common algorithm of choosing the value for "K" takes a linear time (O(n) time since the complexity is O(n)) whereas our heuristic choice algorithm complexity is O(log(n)) so it takes less time to execute as can be seen in Figure 1.

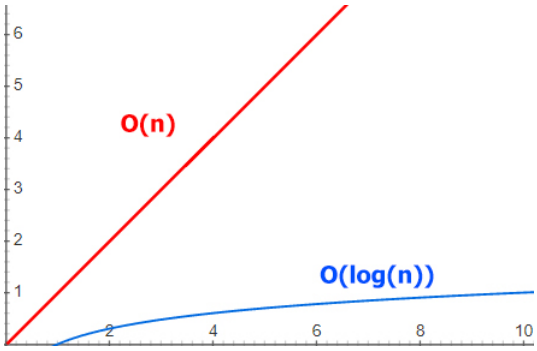

Figure 1: O(n) vs O(log(n))

- APPROX-1 Thread

  This thread is handling the APPROX-1 algorithm without doing anything else and store the result in a global vector.

- APPROX-2 Thread (Heuristic choice for using (u,v) using the greedy method)

The common way to implement this algorithm is to choose a completely random (u,v) but this algorithm can be optimized by using a heuristic choice for (u,v) using the greedy method. We implement this heuristic choice algorithm in a way that it will select a valid vertex with the highest degree for u and accordingly select the other vertex which has the highest degree among vertices that are connected to "u" with an edge as "v".

- IO Thread

  The IO thread is only responsible for sorting and formatting all results in the instructed way and printing the final output.

- Main Thread

  The main thread is responsible for running other threads as well as executing the calculation mode.

3.3 Calculation Mode

The program has been developed to not only execute three different algorithms for G(V,E), also there is a calculation mode that can be activated using a command-line argument while calling the program executable:

$$> ./prjece650 - calc$$

Activating this function will activate some functions to calculate all necessary statistics such as mean and deviation for runtime and approximation ratio for each set of inputs with a similar number of vertices.

3.4 Setting a Timeout reach point for the CNF-SAT thread

As mentioned in the project handout, the CNF-SAT approach is difficult to scale for a large number of vertices. Even though we tried to optimize the algorithm by using a heuristic choice for the value of K and made it more scalable for a larger number of vertices, it still needs a time limit to control this thread for some large cases. To achieve this goal, we experimented on this thread using different values for the number of vertices and found out that we need to control the time for cases with more than 18 vertices since it took a long time and there was no answer for such cases. We did not want to limit the thread nor the program using a criterion on the number of vertices instead, we have tried to find an optimal time to wait before terminating a

thread by running 10 test cases for each value of "V" from 15 to 20. Based on the data shown in Figure 2, it can be concluded that waiting for 10 seconds before terminating a thread is a logical choice considering the deviation for our input test sets and runtime values.

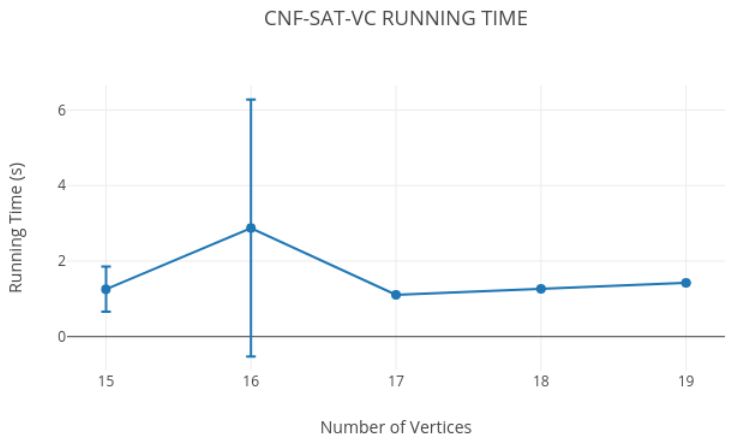

Figure 2: CNF-SAT-VC RUNNING TIME

After terminating the CNF-SAT thread if the calculation mode is on, the runtime and approximation ratio for this approach will be equal to -1 and there will not be any approximate ratio for two other approaches because there is nothing to compare those results to. We developed the calculation mode in a way to eliminate these terminated threads in calculating the mean and deviation for graphs with a similar number of vertices so the -1 value for the mentioned cases will not affect any statistic data.

4 Analysis

As mentioned earlier, the analysis for these three algorithms is based on two factors: the runtime of the thread that is responsible for executing each algorithm, the approximation ratio for the correctness of the output of each algorithm. We have run the program in the calculation mode from 5 vertices to 50 vertices with the increment of 5 and for each number of vertices, we have produced 10 different graphs as our test input.

4.1 Running Time

- CNF-SAT

  The highest runtime belongs to this approach comparing to other implemented algorithms.

The running time of CNF-SAT has a direct relationship with the number of vertices. Since we set the runtime reach point to 10 seconds for each thread, this algorithm is not able to calculate the desired output for inputs that their calculation took more than 10 seconds. The result is plotted as can be seen below:

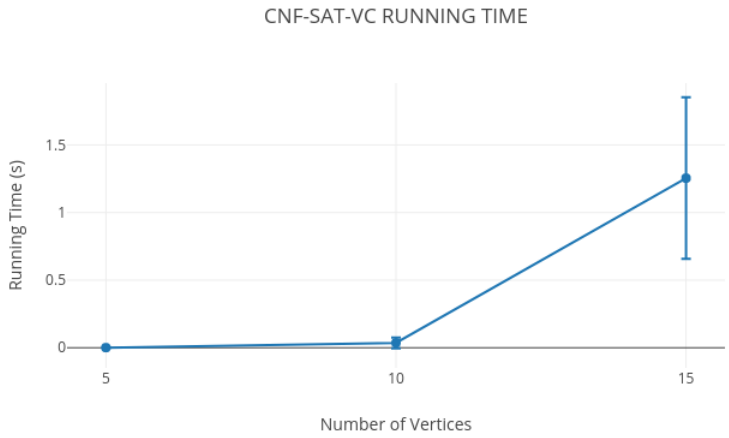

Figure 3: CNF-SAT-VC RUNNING TIME

After running different test cases we found out that the CNF-SAT approach is taking less running time for the lower number of vertices but as this number increases, the increase of running time becomes exponential to the point that the program reaches its time limit for inputs with more than 19 vertices since polynomial-time reduction used in the SAT-Solver is related to the number of vertices.

The execution time complexity for this algorithm considering the heuristic choice algorithm suggested applied by us is:

$$\log(n) * \left(k + n\binom{k}{2} + k\binom{n}{2} + |E|\right)$$

In the above equation, "n" is the number of vertices for a given graph. It can be understood that the number of clauses increases exponentially with the increase in the number of vertices. In Our case, this exponential rise has happened for test graphs with vertices greater than 10.

The deviation for test graphs with 5 to 10 vertices cannot be seen properly since the huge running time for test graphs with 15 vertices compress the visibility of the standard deviation for the program outputs. It can be seen that the standard deviation will be

amplified when we increase the number of vertices although the complexity of the graph can play a role in changing this deviation as well the effect cannot change the final result at all. We can see that there is a small standard deviation for test cases with lower vertex number when we draw our graph for graphs with only 5 to 10 vertices:

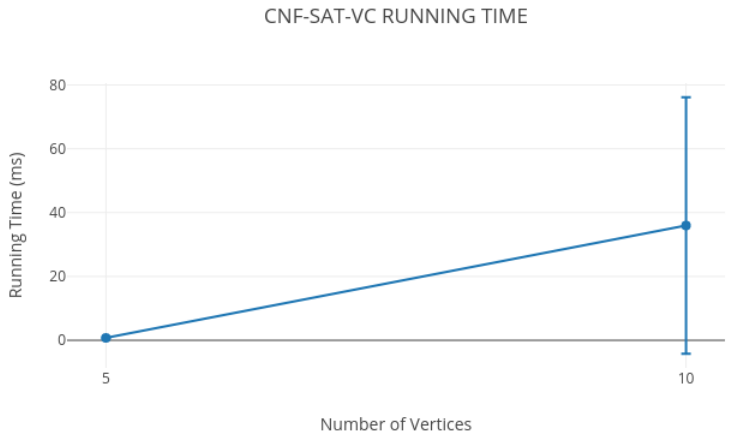

Figure 4: CNF-SAT Running time for graphs with only 5 to 10 vertices

- APPROX-1 and APPROX-2

  The graph below reveals information about the difference between the running time of two other approaches: APPROX1_VC and APPROX2_VC. The overall trend for both approaches shows that by increasing the number of vertices the running time is increasing accordingly.

  The graph also illustrates that for the lower number of vertices the APPROX1_VC approach is a faster solution but when the number of vertices passes 20 the rise of this approach becomes higher than the other approach to the point that eventually for graphs with more than 20 vertices this approach results in a higher running time than the APPROX2_VC. However, we can see that the difference between these two approaches is not significant and that is because of our heuristic choice algorithm that has been applied instead of choosing a complete random vertex. Also, the standard deviation gets amplified by increasing the number of vertices.

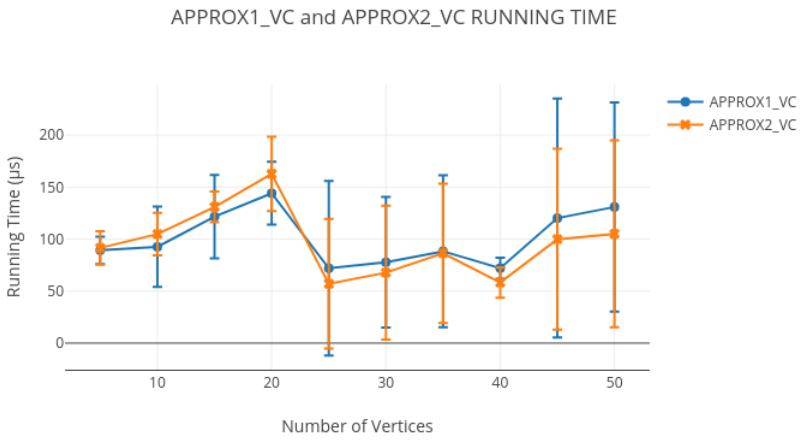

Figure 5: APPROX1_VC and APPROX2_VC RUNNING TIME

- Full comparison

  The graph below provides the difference between the running time of all three algorithms. Although the overall trend for all three algorithms shows that the greater the number of vertices is, the bigger the running time will be, it should be mentioned that the growth for the CNF-SAT runtime is exponential after increasing the number of vertices to more than 10 while the increase for two other algorithms is moderate.

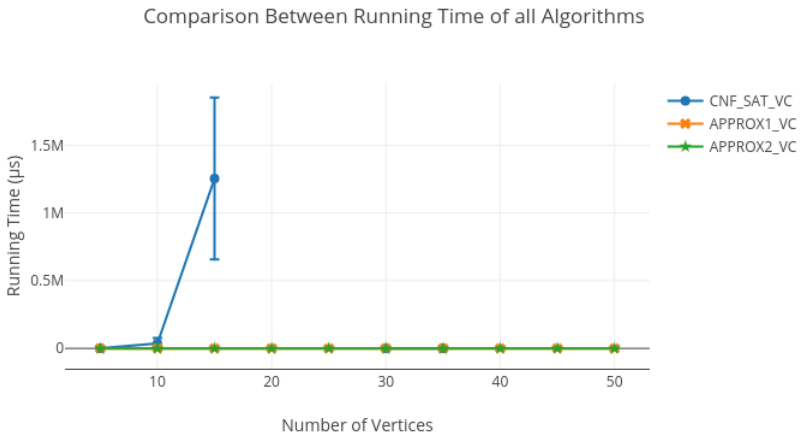

Figure 6: Comparison Between Running Time of all Algorithms

4.2 Approximation Ratio

To calculate the approximation ratio, we need a parameter as a reference to compare different outputs. Since CNF-SAT-VC's output always has an equal or a smaller number of vertices than two other approaches, it is wise to consider it as an optimal solution and to choose the result of

the CNF-SAT-VC approach as the reference for approximation ratio calculation. As a result, the formula is:

$$Approximation\_Ratio = \frac{Size\_of\_computed\_vertex\_cover}{Size\_of\_optimal\_vertex\_cover} = \frac{Size\_of\_algorithm\_Output}{Size\_of\_CNF\_SAT\_VC\_Output}$$

Regarding this formula, the approximation ratio (AR) for CNF-SAT-VC is always equal to 1. Also, we can understand that the smaller the AR, the more optimal will be the output of our algorithm. As the graph below suggests, the AR for APPROX1_VC is exactly 1 for graphs with 5 vertices and the deviation is nearly 0 and also, it is around 1 for graphs with less than 10 vertices, with a slight upward trend. For test graphs with higher than 10 vertices, the trend is changed to downward but still higher than 1 in most cases. The same trend has happened to the AR for APPROX2_VC but the changes are much more noticeable.

As it can be seen on the graph, APPROX1_VC is more optimal than APPROX2_VC and the deviation standard for APPROX2_VC is greater than APPROX1_VC which means that the chance of having a bad result in terms of generating an optimal result is higher for APPROX2_VC.

Also, this should be mentioned that these two algorithms are experiencing the same overall trend and the reason for it is that the first step of both algorithms is the same (choosing a vertex with the highest degree) as we tried to choose vertices using our heuristic algorithm instead of choosing randomly in APPROX2_VC.

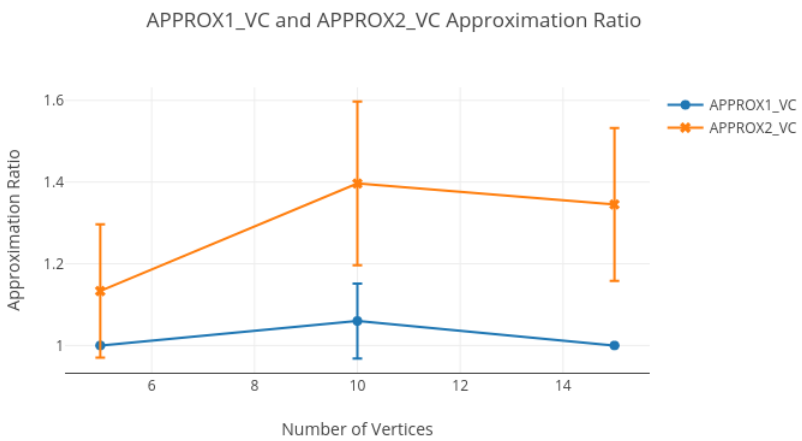

Figure 7: APPROX1_VC and APPROX2_VC Approximation Ratio

5 Conclusion

All these three algorithms can be used to solve the minimum vertex cover problem but the difference in running time and generating an optimal result is noticeable when we try to increase the number of vertices. In conclusion, we can say:

- In general, APPROX1_VC is a better choice to solve large scale vertex cover problem because:
  - The runtime for this approach is less than that of CNF-SAT-VC.
  - The approximation ratio for such cases is near 1 and there is a chance that it might generate the optimal result as well.
- In terms of getting the optimal result, the CNF-SAT-VC approach is the best choice, but the running time can be a big problem for the higher number of vertices and the runtime for cases with more than 19 vertices is too high to the extent that we can consider this algorithm insufficient for such cases.
- In terms of having a fair running time, the choice would be APPROX2-VC for cases with a higher number of vertices but the approximation ratio was a lot greater than APPROX1_VC hence, there is a high chance that this algorithm cannot generate the optimal result.