Efficiency Analysis & Comparison of Various Vertex Cover Algorithms

ECE 650: Methods & Tools for Software Engineering

Fall – 2017

Report by:

ABHISHEK THAPLIYAL – 20690827

MANSUKH SINGH SEERHA - 20690558

1 INTRODUCTION

In this report, comparison of three vertex cover algorithms are represented and discussed. The goal of each algorithm is to deduce the minimum sized vertex cover of a given graph (V,E). Where, V is the total number of nodes and E denotes the edges which are formed between the nodes. The first Algorithm, which is called CNF-SAT, relies on the conversion of polynomial-time reduction form vertex cover to CNF-SAT (Conjunctive normal form) problem. The second one is called Approx-1, it works by picking up the most effective vertex (with most incident edges) and remove all its connections in the edges list, it will continue this operation until no edges remain. Lastly, for the third and final algorithm, called Approx-2, it works by picking a random edge (instead of a vertex) and remove all its connections from the edges list, this will continue until no edges are left.

We are comparing these algorithms on the basis of their running time and approximation ratio. To see the running time as well as other measurements, the user is required to add "*-calc*" as a command line argument when running the executable file (i.e., "**./ece650-prj -calc**"). The results were plotted and represented using *matplotlib* package tool in python script.

2 ANALYSIS

2.1 EXPERIMENT DETAILS

The analysis program is based on multi-threading. A total of four threads are used, namely I/O thread (for input), and three more threads (one from each of the vertex cover algorithm). The I/O thread is responsible for distributing input to the other threads. The running time is computed at the end of each thread, *pthread_getcpuclockid ()* function was used for getting the CPU time of each of the algorithm function. The program was tested on a local Linux machine. The machine is an Intel Core i5 Multicore-processor, 2.7GHz, 8G of RAM, and its OS is Ubuntu.

The analysis of efficiency for all the three algorithms were done on the basis of two factors (1) Running Time (which is the CPU clock time for each algorithm). (2) Approximation Ratio (which is the ratio of the size of computed vertex cover to the size of optimal/minimum vertex cover). Here, CNF-SAT output is considered as an optimal approach than the other outputs.

2.2 RUNNING TIME ANALYSIS:

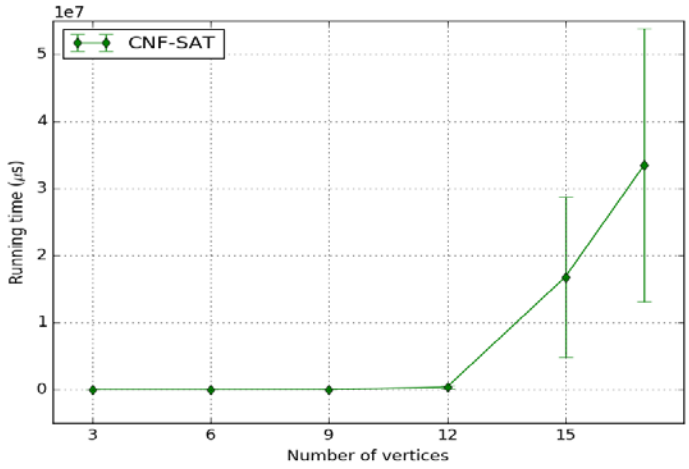
2.2.1 RUNNING TIME ANALYSIS FOR CNF-SAT:

As per our analysis, CNF-SAT algorithm takes the maximum running time, among all three algorithms, to calculate the vertex cover. We have taken readings from 3 vertices to 17 vertices with the increment of 3 (i.e. 3,6,9,12,15 till 17). Since, CNF-SAT takes a lot of time to calculate vertex cover for higher vertices, therefore, we have restrained our analysis up to 17 vertices.

Alternatively, CNF-SAT algorithm takes very less running time when number of vertices are less, specifically less than 12 but it shows exponential rise in the running time when number of vertices becomes greater than 12. This is because polynomial time reduction used in sat solver is related to the number of vertices, for any graph the number of clauses passed to the sat solver is given by:

$$\boldsymbol{k} + \boldsymbol{n}\binom{\boldsymbol{k}}{2} + \boldsymbol{k}\binom{\boldsymbol{n}}{2} + |\boldsymbol{E}|$$

Here, n is the number of vertices for a given graph. So as per the above equation the number of clauses increases exponentially with the the increase in the number of vertices and hence we have an exponential increase in the running time of SAT algorithm after 12 vertices.

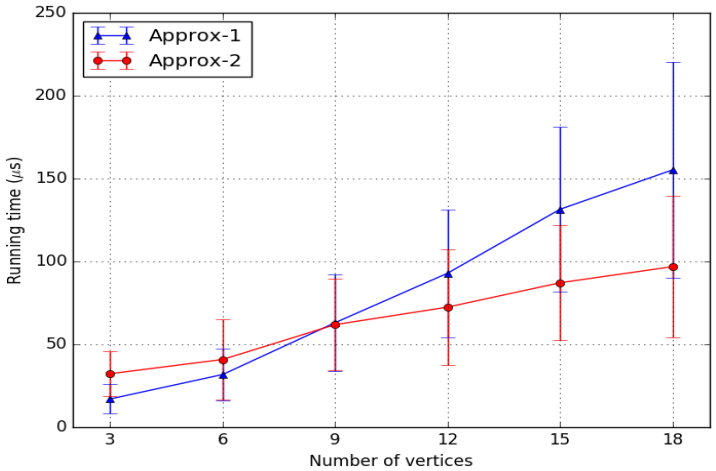


GRAPH 1: CNF-SAT RUNNING TIME ANALYSIS IN INTERVAL [3,17].

Now coming to the standard deviation, the graph however does not show the standard deviation for smaller vertices, since the huge running time on higher vertices (more than 12) compresses the observation of the standard deviation of running time on graph for small vertices number. But, there is small standard deviation for lower number of vertices too. And this standard deviation gets amplified when the number of vertex increases for graph.

2.2.2 RUNNING TIME ANALYSIS FOR APPROX-1 AND APPROX-2 ALGORITHMS:

Graph 2 shows the difference between the running time taken by two algorithms, Approx-1 and Approx-2. The scale of vertices is taken from [3,18] with the increment of 3. From the graph we can clearly say that Approx-1 one takes more time than Approx-2 for higher number of vertices. And the reason behind this is the time complexity of the Approx-1 is greater than that of Approx-2.



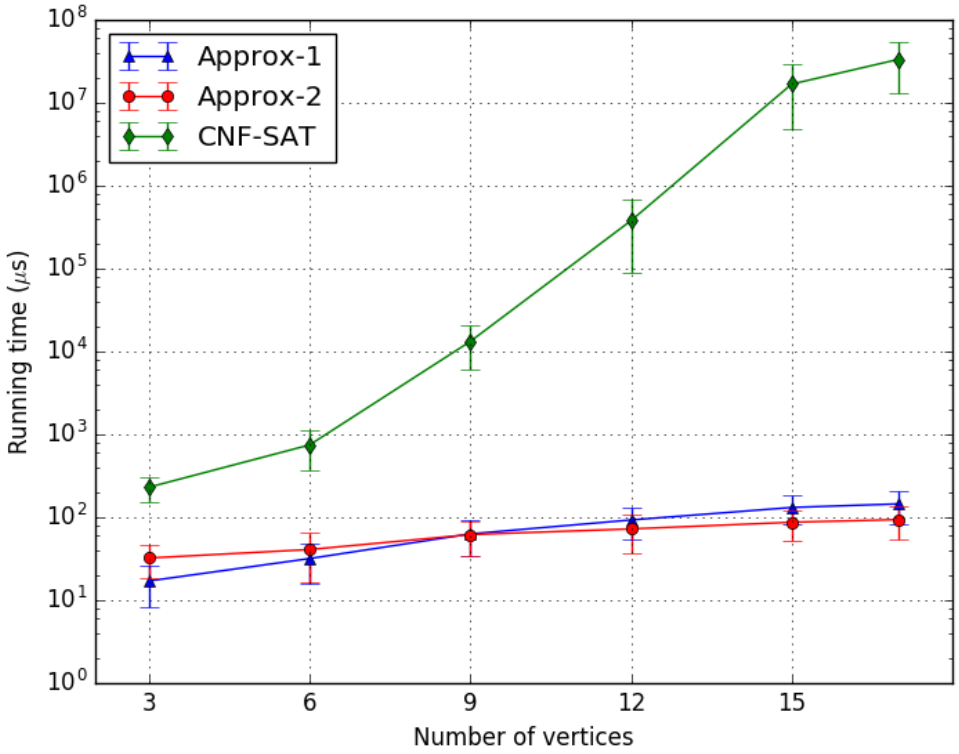
GRAPH 2 : COMPARISION ANALYSIS BETWEEN RUNNING TIME OF APPROX-1 & APPROX-2 IN INTERVAL [3,18].

On the other hand, for less number of vertices (3 and 6), Approx-1, was found to be relatively faster. The reason behind this can be attributed to the randomness involved in Approx-2. Since, each time APPROX-2 will pick a random edge and depending on that particular edge, the running time as well as output will vary. While, in Approx-1, for smaller number of vertices, it had to go through less number of loops as the graph for those vertices will be small. So it will take considerably less time.

However, for higher vertices (i.e., [9,18]), Approx-2 was found to be faster then Approx-1. As mentioned before, this can be justified by considering the randomness for Approx-2 and more loops that Approx-1 had to go through.

2.2.3 RUNNING TIME ANALYSIS – COMPARISION:

For comparison of all 3 algorithms, we have taken log scale on Y axis (i.e. the running time) and X axis as the number of vertices from [3,17] with the increment of 3 till 17. Now from graph 3, we can observe that CNF-SAT takes much more time in comparison to Approx-1 and Approx-2. Moreover, CNF-SAT follows an exponential increase trend (as mentioned before) in running time with respect to increase in the number of vertices. While, on the other hand, this is not the case for other two algorithms. Also, in CNF-SAT, it has to traverse the whole solution space and have to look for all possible assignments to the given CNF in order to satisfy every clause to find the minimum vertex cover. The number of clauses and hence time taken by SAT solver increases exponentially with the increase in vertices, it is given by the same aforementioned formula (i.e. $\boldsymbol{k + n\binom{k}{2} + k\binom{n}{2} + |E|}$). While in the case of other two algorithms, they have less time complexity as compared to CNF-SAT. Hence, they will never show exponential rise in their running time as oppose to CNF-SAT.



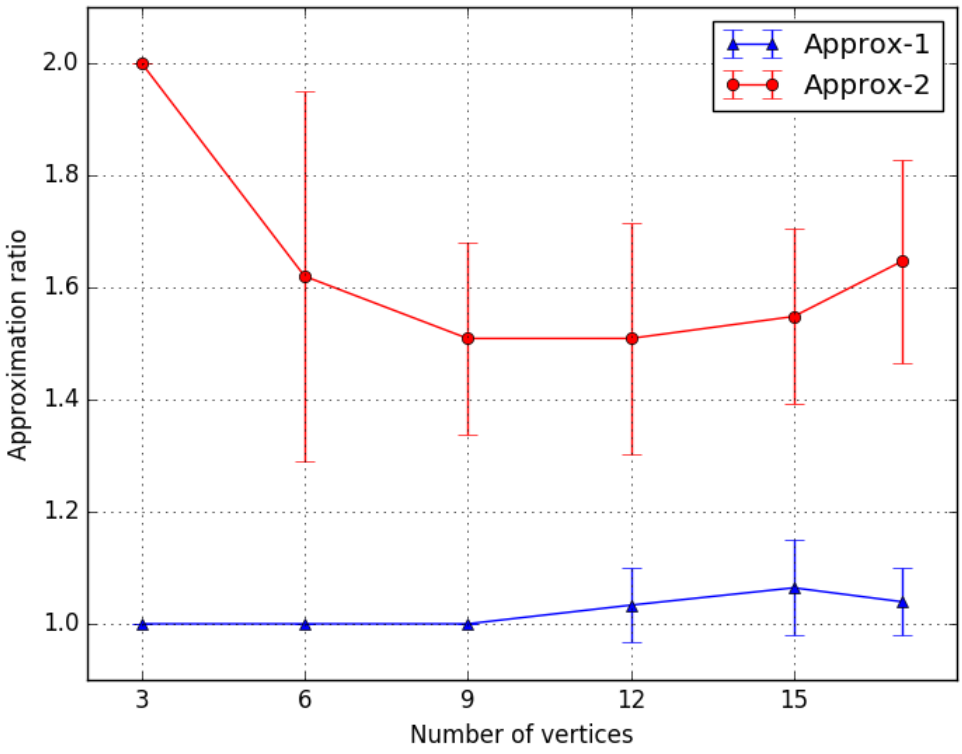
GRAPH 3: COMPARISION ANALYSIS BETWEEN RUNNING TIME OF ALL THE THREE ALGORITHMS IN INTERVAL [3,17],WITH LOG-SCALE ON Y AXIS.

2.3 APPROXIMATION RATIO ANALYSIS:

For calculating the approximation ratio, we have used the formula:

$$\textbf{\textit{Approximation Ratio}} = \frac{\textbf{\textit{Size of Computed Vertex Cover}}}{\textbf{\textit{Size of Optimal Vertex Cover}}}$$

So smaller the approximation ratio, more optimal will be the output of the algorithm. As we can see from graph 4 the approximation ratio for most of the vertices (lesser than 9) is 1 for Approx-1. On the other hand, the approximation ratio is quite high for Approx-2. This is because the Approx-1 algorithm picks up vertices, while Approx-2 picks up whole of the edge. So in Approx-1, for most of the time, it gives us the minimum vertex cover (especially for lesser number of vertices) which was found to be equivalent to the output of CNF-SAT. While, it shows a slight increase in approximation ratio for larger vertices but not as much as observed for Approx-2. Approx-2, for most of the cases, will provide vertex covers which will always be greater than the minimal vertex covers given by the CNF-SAT algorithm. The standard deviation in Approx-2, corresponds to the randomness aspect of the algorithm, and may vary after each run.



GRAPH 4 COMPARISION ANALYSIS BETWEEN APPROXIMATION RATIO OF APPROX-1 & APPROX-2 IN INTERVAL [3,17].

3 CONCLUSION

In a nut shell, as deduced from the above plotted graphs, it is safe to say that all the three algorithms can be used to solve the vertex cover problem. But, on the basis of running time and approximation ratio, their efficiency may vary.

- In terms of running time, Approx-2 is better for higher number of vertices, but it fails to get minimum vertex cover. So it is a better way to get vertex cover but not the minimum vertex cover.

- In general, Approx-1 is a better approach to solve large scale vertex cover problem, the running time is less than that of CNF-SAT, and it can sometime get minimum vertex cover result. Also, when compared with Approx-2, Approx-1 was found to run faster when it was computing vertex cover for lower number of vertices (3 and 6).

- In terms of approximation ratio, CNF-SAT definitely gets the minimum vertex cover, but it requires more time than the other two approaches, Approx-1 and Approx-2. So it is a better way to solve small scale vertex cover problems, especially for cases when number of vertices are less than 18.