UNIVERSITY OF WATERLOO

ECE650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

PROJECT REPORT

Analysis of Vertex Cover Algorithms

Report by:
Alireza LOTFI TAKAMI
Krishna KANTWALA

Instructor:
Dr.Reza BABAEE

March 31, 2020

1 Introduction

In this project, we have implemented three algorithms in order to solve the vertex cover problem for a given input graph. The input graph is a set of vertices and edges between them. It has been shown that finding the minimum number of vertices that cover all of the graph edges is an NP-complete problem. It means solving this problem is too time-consuming. One approach to solving these kinds of problems is using approximation algorithms. The approximation algorithms do not always produce the precise output but they solve the NP-complete problem considerably fast. In this project, we have solved the vertex cover problem using three different algorithms. We named this algorithm CNF-SAT-VC, APPROX-1-VC, and APPROX-2-VC. We have used a multi-threading approach in our implementation using C++ programming language. There are four main threads in this implementation. One thread is the I/O thread and the three other threads are for the three mentioned algorithms. In the subsequent sections, we provide details of each algorithm, then we compare these approaches together.

2 CNF-SAT-VC

This solution is the most precise solution for the vertex cover problem. The output of this algorithm is exactly the minimum number of vertices needed to cover all edges of the input graph. As it was mentioned minimum vertex cover problem is an NP-complete problem. Hence we use a sat solver called Minisat to solve the problem. In order to use Minisat for the vertex cover problem, we needed a polynomial encoding from the vertex cover problem to the Satisfiability problem. we have implemented these encoding using C++ programming language and we pass the output of this encoding as an input to the Minisat tool. The output of our encoding is a Conjunctive Normal Form (CNF) formula. Using the output Minisat we can specify the vertices which needed to be select to solve the minimum vertex cover problem. This approach solves the minimum vertex cover problem but it is too slow and for larger input graph size it takes intolerably long to produce the output.

2.1 Improvement in Encoding

The encoding from the vertex cover problem to sat problem works fine and the output is correct. But it takes too long for graphs with a size of more than fifteen. In order to solve this problem faster, we improved the encoding such that the output of the encoding algorithm is 3-CNF and now we can see the outputs for graphs with a size of thirty. Even using this encoding is not helpful for input graphs with a size of more than thirty. Therefore we need other algorithms to solve the vertex cover problem faster. So, we use two fast approximation algorithms for the vertex cover problem.

3 APPROX-1-VC

In this algorithm, we pick a vertex with the highest degree and add it to our vertex cover and omit it and all of the edges that are connected to it for the input graph. we repeat this till there are no edges in the input graph. Although this algorithm is considerably faster than CNF-SAT-VC and provides the vertex cover solution, its output is not always the minimum number of vertices which is needed to cover all of the edges of the input graph. The size of the computed vertex cover to the size of a minimum-sized vertex cover called approximation ratio.

4 APPROX-2-VC

In this algorithm, we Pick an edge of the input graph and add the two vertices which are connected to it to the vertex cover and remove the vertices and all of the edges which are connected to them from in input graph. We repeat till no edges remain in the graph. Again, the output of this algorithm is sometimes more than the minimum number of vertices which is needed to cover all of the edges of the input graph. So, we need to consider its approximation ratio. In the subsequent sections, we examine the execution time of these three algorithms.

5 Execution Time Analysis

We implemented three algorithms and we mentioned that they have different execution times. In our implementation, we use $pthread\_getcpuclockid()$ function at the end of each thread to measure the execution time of each algorithm.

5.1 CNF-SAT Execution Time

To analyze the execution time of this algorithm we considered the input graphs with a size of [5, 50], but as Figure 1 indicates the maximum input graph size that we manage to analyze its execution time is 30. According to our encoding, the execution time of this algorithm increases exponentially by input graph size. The reason is that the larger input graph leads to a high number of clauses as the output of the encoding algorithm and also these clauses are too large. we coped with the large clauses by changing the encoding in a way to produce 3-CNF instead of k-CNF. Without this improvement, we only could see the outputs for the input graphs until size 15. The number of clauses that our encoding produces is as follows:

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

Where n is the number of input graph vertices. k is the size of vertex cover and $|E|$ is the number of input graph edges. As can be seen from the formula, as n increases the number of clauses in the output of the encoding algorithm increases significantly. As a result the

execution time of CNF-SAT-VC increases. We evaluated the algorithm execution time by taking 10 of each size. The error bar in the plot corresponds with the standard deviation. It can be seen that for larger input graph size the standard deviation increase considerably. The reason for that can be the way the OS schedules the process and threads and also the way the Minisat solver solves the SAT problem.

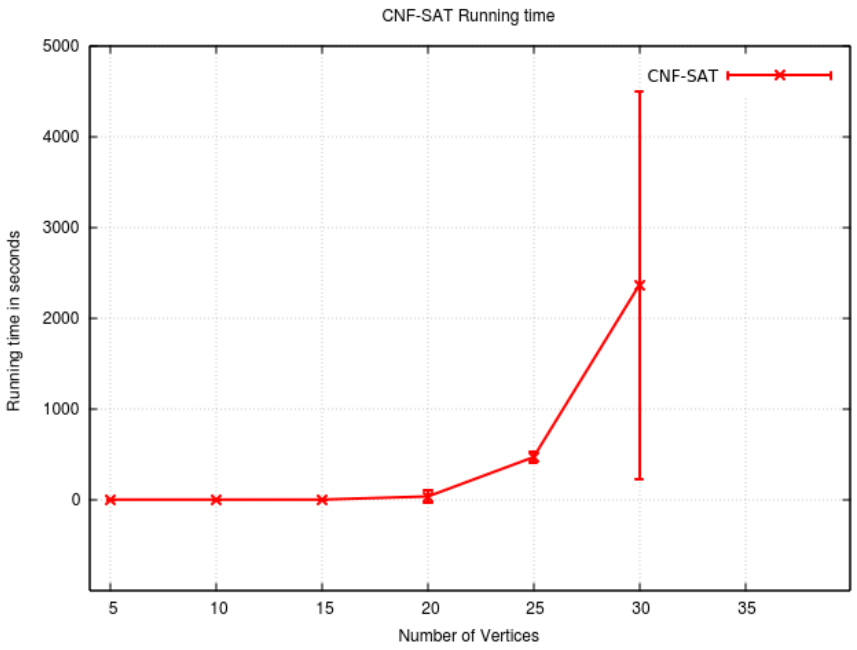

Figure 1: Execution time of CNF-SAN-VC

5.2 Approximation Algorithms Execution Time and Comparison

To analyze the execution time of these algorithms we considered the input graphs with a size of [5, 50]. Despite the optimal solution algorithm, we can easily measure the execution time of this approximation algorithms until 50 for graph input size. As we anticipated and can be seen in Figure 2, our approximation algorithm execution time is much lower than the execution time of the optimal solution. According to our implementation, we expect that the Approx-1-VC execution time is more than the Approx-2-VC execution time in most cases. The reason for this expectation is that Approx-1-VC traverses all vertices of the input graph each time to find the vertex with the maximum degree, But Approx-2-VC just picks an edge each time sequentially. Therefore, we think Approx-1-VC time complexity is more than Approx-2-VC time complexity. In the next section, we are going to discuss an execution time comparison among all of these three algorithms.

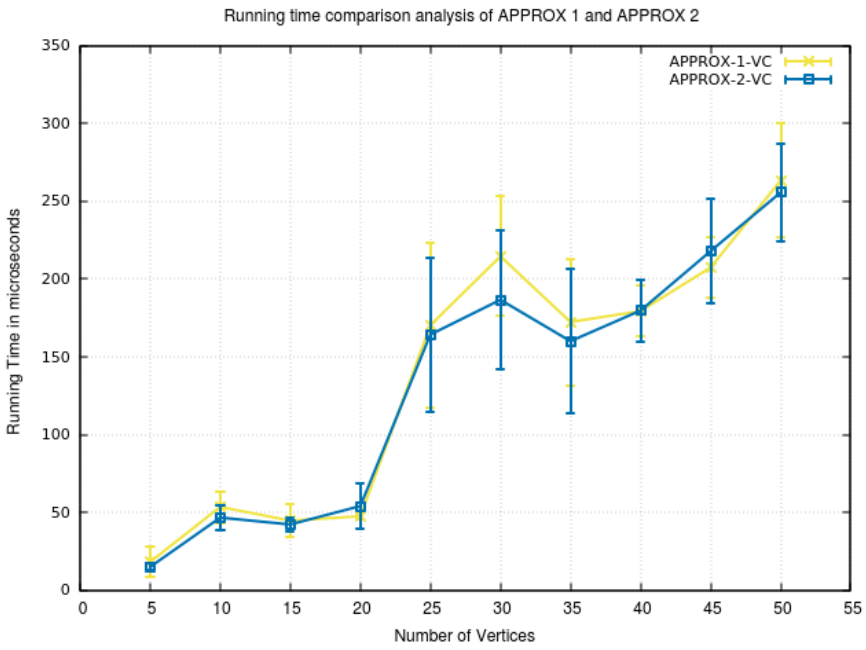

Figure 2: Running time comparison analysis of Approx-1-VC and Approx-2-VC

Figure 3 indicates a visual comparison between the execution time of these three proposed algorithms. In the figure x-axis is the number of vertices and y-axis in $\log_{10}$ scale is the execution time of algorithms in us. The reason that we have a logarithmic y-axis is that the execution time of the optimal solution is too greater than two other algorithms. Hence, on a normal scale, it would be too difficult to see the execution time of Approx-1-VC and Approx2-VC and compare them with optimal execution time on the same plot. It can be seen in Figure 3 that the execution time of the optimal solution is more than Approx-1-VC and Approx-2-VC in all cases. We only depicted the execution time for input graphs with a size of less than 30 and that is because with larger input graph size the execution time of the optimal solution increases exponentially. So it cannot be measured and depicted in the plot.

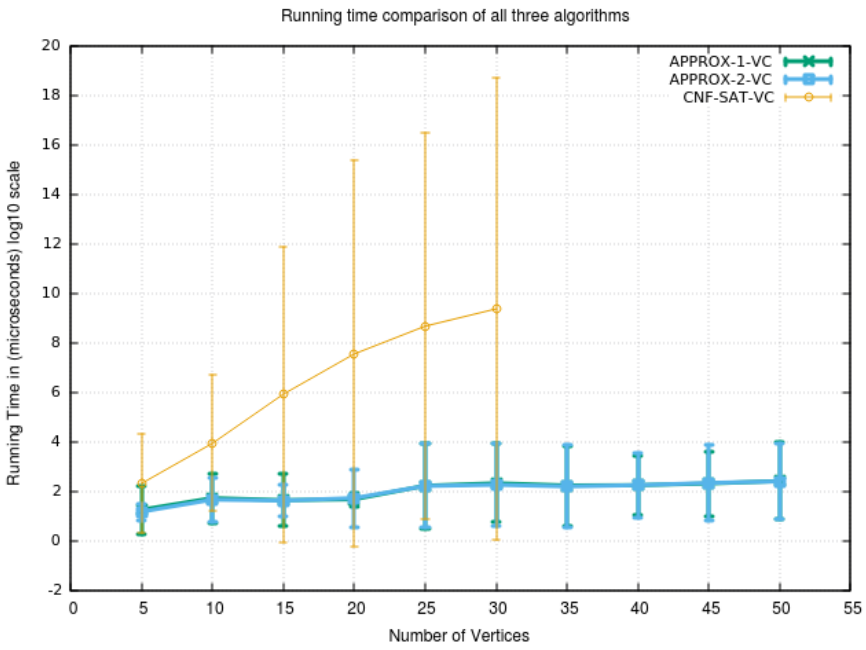

Figure 3: Running time comparison of all three algorithms

6 Approximation Ratio Analysis

As we discussed before, Although the approximation algorithms are considerably faster than the optimal solution, they are not as accurate as that. Hence, we defined approximation ratio for the vertex cover approximation algorithms as follows:

$$ApproximationRation = \frac{computed\_vertex\_cover\_size}{optimal\_vertex\_cover\_size}$$

As can be seen in Figure 4, the Approx-1-VC approximation ratio is lower than the Approx-2-VC approximation ratio. Therefore, Approx-1-VC is more reliable than Approx-2-VC. The standard deviation in Approx-1-VC is negligible and that is because in our implementation each time we pick the vertex with the maximum degree of the input graph.

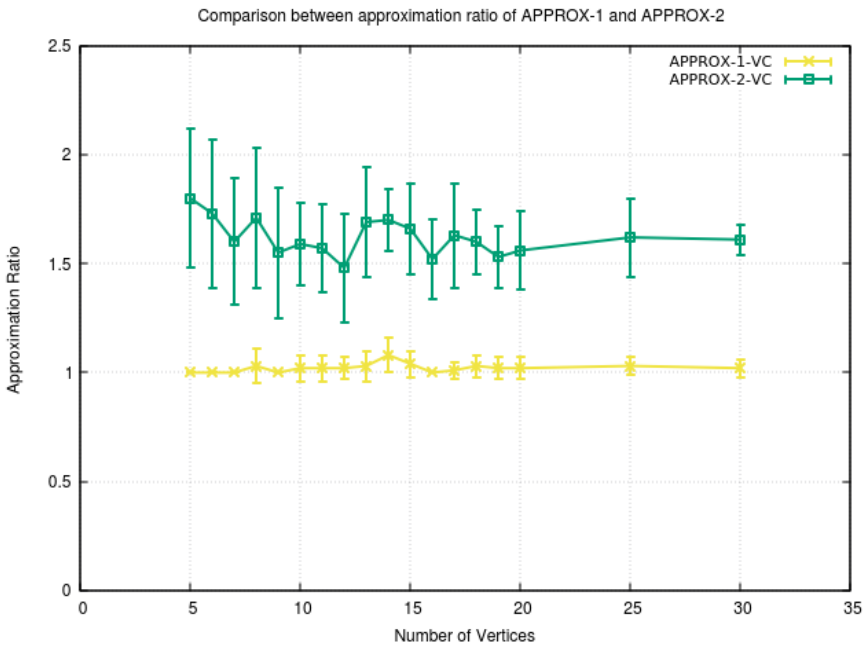

Figure 4: Comparison between approximation ration of Approx-1 and approx-2

7 Conclusion

We can use all of these three algorithms to solve vertex cover problem. But, We have to consider running time and approximation ratio of each of them. Considering efficiency and optimally we have the following options.

- If finding the minimum size vertex cover is the most important thing for us and execution time of the algorithm is not that much important or if the size of the input graph is small, then using CNF-SAT-VC is the best choice. CNF-SAT-VC guarantees finding the minimum vetex cover. For larger input graphs it is not possible to use this algorithm. Hence, we need to use the approximation algorithms.

- A faster solution to solve the vertex cover problem is Approx-1-VC. The execution time of this algorithm is much less than the optimal solution. Its approximation ration is quite low and in many cases its approximation ratio is one. It means in many case it can even produce the optimal solution and in other cases its result is near to optimal solution.

- Approx-2-VC has the best execution time among these three solution. but, its approximation ratio is higher than others and in most cases it cannot produce optimal solution. Hence, If we only care about solving the vertex cover problem as fast as possible and the optimally of solution is not that much important or the input graph size is too large then this algorithm could be the best choice.