ECE 650

Methods and Tools of Software Engineering

Report
Final project

by Adam Kulas (20302000)

and Simon Winkler (20806571)

1. Introduction

The aim of this project was to compare the performance of three different algorithms which solve the minimum-vertex-cover problem.

These three algorithms are:

- Approx-VC-1 (as described in the instructions)
- Approx-VC-2 (as described in the instructions)
- CNF-SAT-VC (as described in the instructions)

These algorithms have been compared by analyzing both their respective runtime and their approximation ratio (size of min-vertices-set vs. size of optimal min-vertices-set).

The input graphs which have been used were generated by a graph generator written by Arie Gurfinkel. In order to measure the performance, the graphs were generated with an increasing number of vertices, as follows:

- 10 graphs with 5 vertices	(run each graph 10 times)
- 10 graphs with 10 vertices	(run each graph 10 times)
- 10 graphs with 15 vertices	(run each graph 10 times)
- 3 graphs with 17 vertices	(run each graph 3 times)
- 2 graphs with 20 vertices	(run each graph 3 times)

The V=17 and the V=20 graphs were run only 3 times due to the high runtime of the CNF-SAT-VC algorithm.

2. Analysis

Each of the algorithm's execution time was benchmarked by using the clock_gettime function. Each thread was timed using its respective cpuclockid obtained by the function found in the pthread library. The performance results of the algorithms can be seen in figure 1 below.

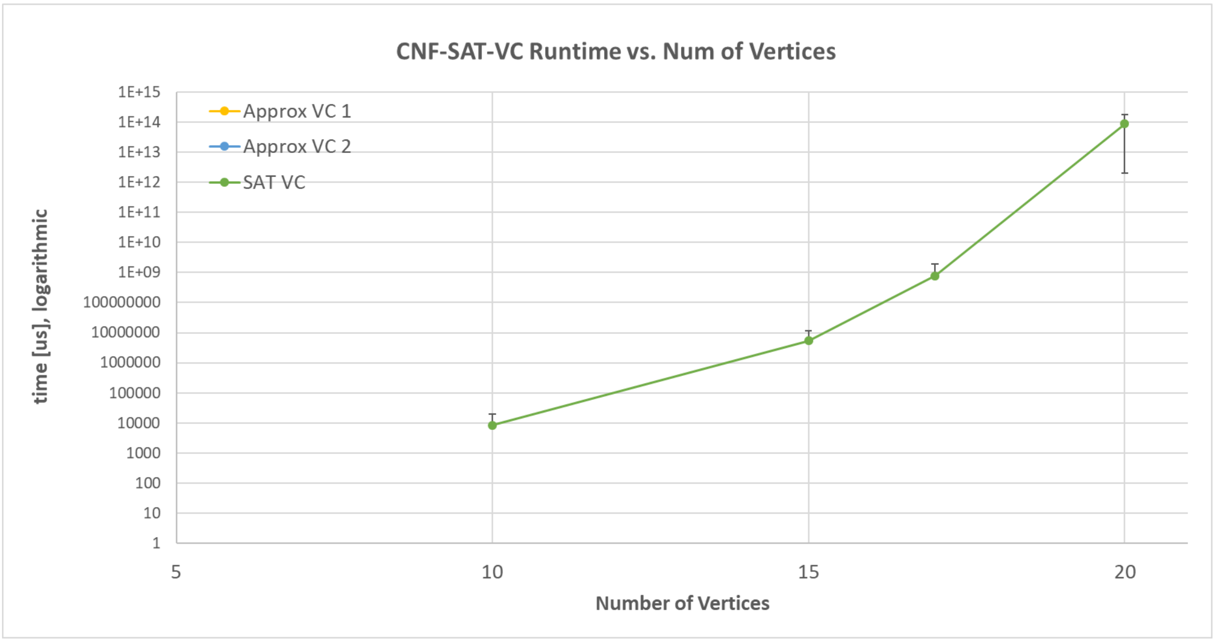

Figure 1: CNF-SAT-VC performance with Graphs containing differing number of vertices

As it can be seen in figure 1, both Approx-VC-1 and Approx-VC-2 can deal with graphs of up to 15 vertices in less than one microsecond. The same holds for CNF-SAT-VC, when the graphs with 5 vertices are being input. However, CNF-SAT-VC starts to need significantly more time as the number of vertices increases. In order to properly show this trend, the graph is shown with a logarithmic y-axis. The logarithm axis results in the error bars distorting.

In order to properly show how the runtime increases with larger input graphs, the approximate vertex cover algorithms were benchmarked using graphs up to 2000 vertices. The CNF-SAT-VC algorithm was excluded from this test. The results of the test can be seen below in Figure 2.

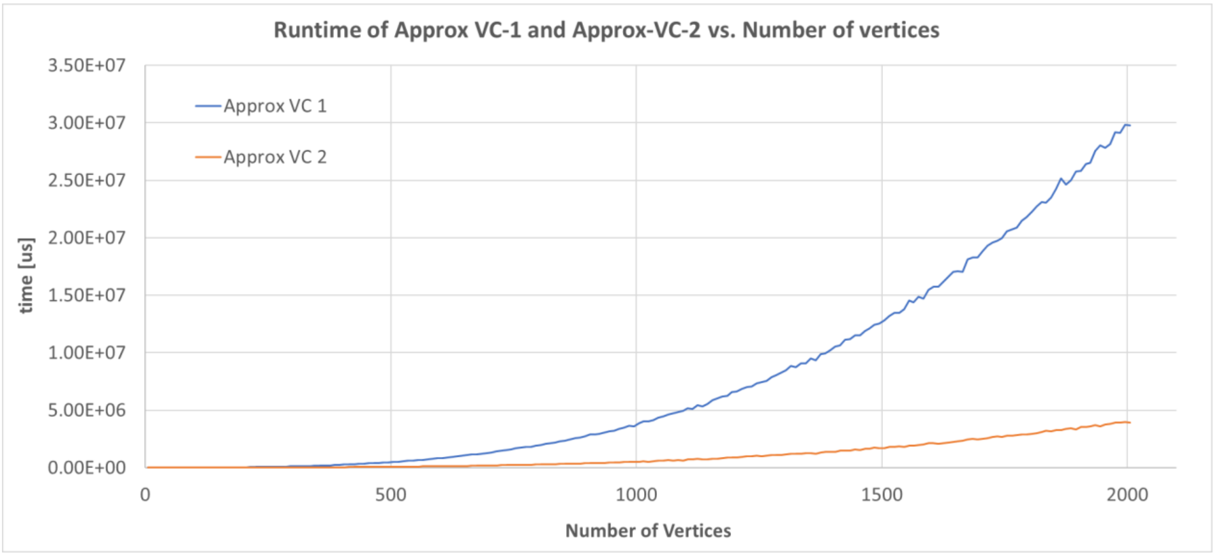

Figure 2: Runtime of approximate vertex cover algorithm with large input graph sizes

From the graph above, the performance of the two algorithms is very similar until around a vertex size of 300. It can then be seen that the two algorithms diverge due to the runtime complexities of the algorithms themselves. When a graph of 2000 vertices is input the APPROX-VC-1 algorithm computes its result in approximately 30 seconds while the APPROX-VC-2 algorithm takes under 5 seconds.

Another key performance benchmark is the approximation ratio between the approximation algorithms and the optimal minimum vertex cover obtained via the CNF-SAT-VC algorithm. The approximation ratio was computed by taking the size of the output vertex cover and dividing it by the size of the optimal minimum vertex cover. The results are visualized in the bar chart shown below in figure 3.

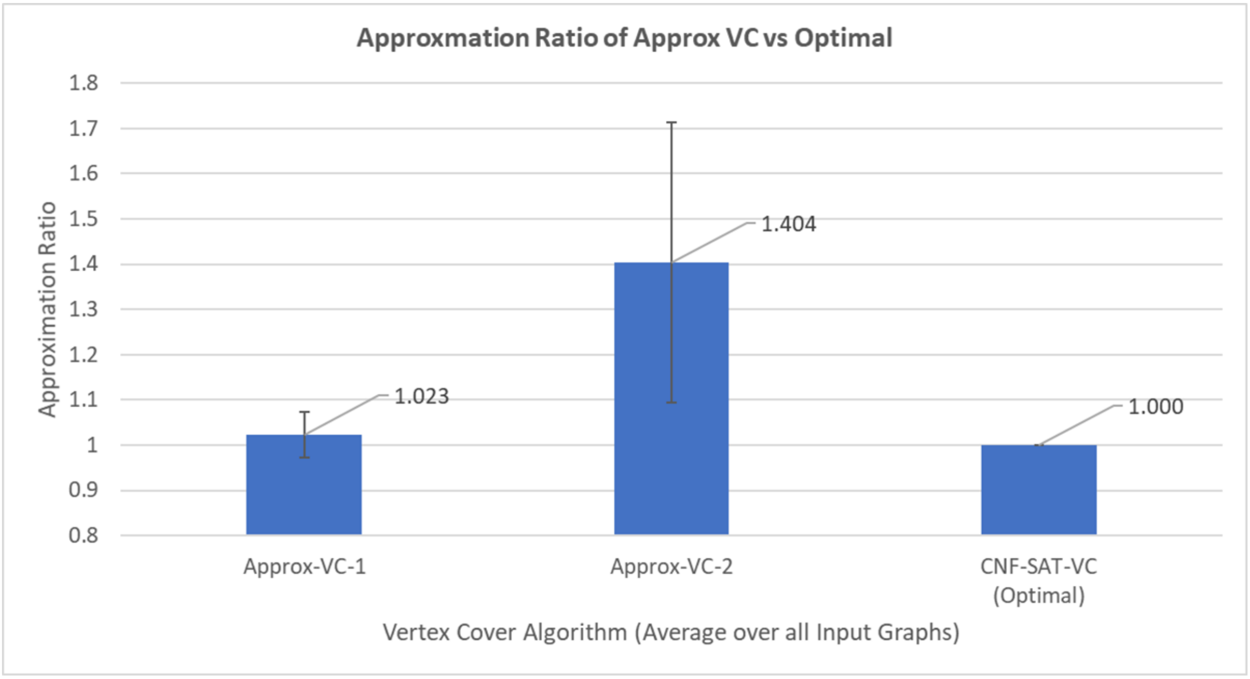

Figure 3: Approxmation ratio of approximation algorithms versus optimal minimum vertex cover

The values seen above in figure 3 are the mean approxmation ratios over all input graphs mention in section 1. The APPROX-VC-1 algorithm is highly performant in returning a vertex cover near to the optimal value obtained by the sat solver. The APPROX-VC-2 algorithm did not perform well at reaching the optimal solution. On average the size of the minimum vertex cover was 40% larger than the optimal solution. Its standard deviation can be seen in the error bars and show the results fluctuate greatly between different inputs.

To achieve greater introspection to the results achieved by the approximation algorithms, the approxmation ratios were also displayed for each set of graphs with the same number of vertices. This can be seen in figure 4 below.

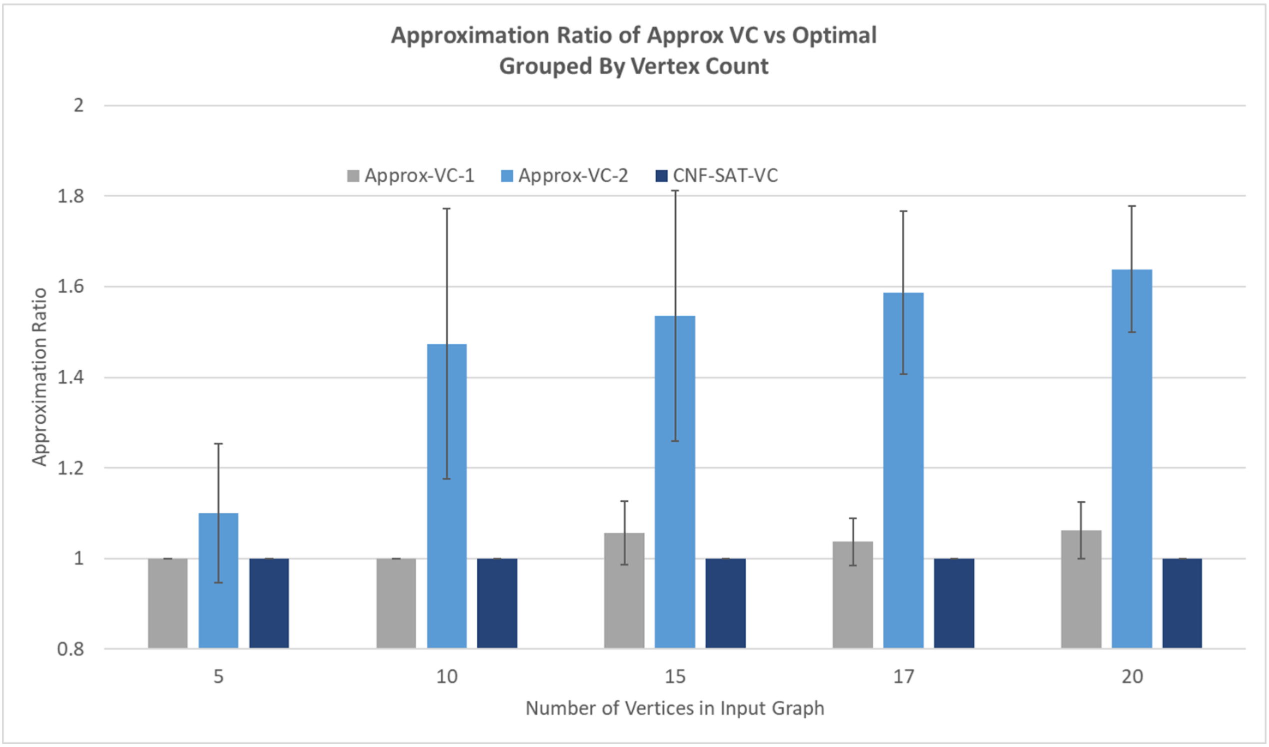

Figure 4: Approximation ratio grouped by input graph size

3. Results

The performance of each approach varies from one another. Each run was performed using the same hardware to keep the results consistent. In terms of runtime APPROX-VC-2 ran the quickest based on the results gathered from conducted trials. When the algorithm implementation is analyzed its run time is $\Theta(|V|)$. The algorithm simply iterates through all edges of the input graph to construct the approximate vertex cover. Each iteration it must remove all vertices connected to it. The graph was represented using an adjacency matrix so the process of removing vertices is simply setting a row and column value to “0”.

A similar approach can be used to describe the runtime performance of the APPROX-VC-2 algorithm. The algorithm must first find the vertex with the highest degree. It then adds that vertex to its minimum vertex cover and removes all attached edges. To find the highest degree vertex it must go to each vertex and check how many connections it has to all other vertices. This runs in complexity $\Theta(|V|^2)$. Figure 2 illustrates this conclusion.

When running the sat solver to determine the optimal min vertex cover, each proposed size for the vertex cover, “k”, was entered starting from 1 until the sat solver returned true. Originally, a binary search implementation was attempted but it was found to perform worse than a linear search. This is because the runtime of the algorithm is evenly distributed over all values of k. It was found that the sat solver took significantly more time at values near to the optimal result. For this reason, a binary search may end up computing multiple values around the optimal and performing drastically worse.

It was also interesting to see how different graph inputs were much more difficult for the sat solver to return a result. For example, three 17 vertex graphs were used. One of the graphs took over 40 minutes to compute while the other two took no more than 2 seconds.

The optimality of the solutions also varied between the approaches. APPROX-VC-2 gave results furthest from the optimal solution. This could be because the algorithm blindly chooses edges to add to the vertex cover. With this strategy it is purely chance that dictates the optimality of the results. APPROX-VC-1 implements a clever heuristic where the max degree vertices are added first. This allows a much closer result to the optimal solution because high degree vertices are likely to touch many edges and contribute greatly to the vertex cover. Obviously the sat solver returned optimal results and was used as the control and benchmark for the approximation ratio results. It is important to note that for larger graphs the CNF-SAT-VC algorithm quickly becomes intractable due to its exponential nature. Depending on the application, it would be good to consider an approximation method when graph inputs become larger than 20 vertices.