ECE 650 Group Project Report

By G. Richard, T. Liu
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada

Dec 4, 2019

1 Introduction

The vertex cover problem is a very famous NP-hard problem in the realm of optimization. Finding a minimum vertex cover is very difficult and time consuming to compute but finding alternative approaches are easier. Due to this, many approximation algorithms have been proposed in the past. These algorithms must have a polynomial time complexity and must return a relatively valid solution. In this report, two such algorithms are explored and compared against the results of a modern SAT-solver based on running time and quality of the their respective outputs. The algorithm descriptions, analysis method, results and conclusions are presented in sections 2, 3, 4 & 5, respectively.

2 Algorithm Descriptions

2.1 SAT Solver Algorithm

The SAT-solver used as part of this investigation is called MiniSAT, designed by Niklas Eén and Niklas Sörensson. MiniSAT is a very popular open source SAT solver that has risen to fame in the early 2000s due to its simple, well documented implementation. It contains certain features such as incremental SAT-solving and a well-defined interface for general boolean constraints. MiniSAT is considered a modern SAT-solver since it implements the David-Putnam-Logemann-Loveland (DPLL) procedure, backtracks by conflict analysis and clause learning and propagates boolean constraints using watched literals.[1]

Since SAT solvers require its input to be in conjunctive normal form (CNF), the vertex cover problem needed to be encoded to fit the input criteria for MiniSAT. The vertex cover problem was reduced to CNF form as follows:

Given a pair (G,k), where $G=(V,E)$, let $|V|=n$. Create $n*k$ atomic propositions, denoted $x_{i,j}$, where $i\in[1,n]$ and $j\in[1,k]$. A vertex cover of size k is a list of k vertices. Let $x_{i,j}$ be true if and only if vertex i of V is the jth vertex in that list. The clauses are as follows:

- At least one vertex in the ith vertex in the vertex cover:
  $$\forall i\in[1,k],(x_{1,i}\vee x_{2,i}\vee\cdots\vee x_{n,i})$$
- No one vertex can appear twice in a vertex cover:
  $$\forall m\in[1,n],\forall p,q\in[1,k] \text{ with } p<q,(\neg x_{m,p}\vee\neg x_{m,q})$$
- No more than one vertex appears in the mth position of the vertex cover:
  $$\forall m\in[1,k],\forall p,q\in[1,n] \text{ with } p<q,(\neg x_{p,m}\vee\neg x_{q,m})$$
- Every edge is incident to at least one vertex in the cover:
  $$\forall\langle i,j\rangle\in E,(x_{i,1}\vee x_{i,2}\vee\cdots\vee x_{i,k}\vee x_{j,1}\vee x_{j,2}\vee\cdots\vee x_{j,k})$$

Based on the above encoding, it was determined that the SAT solver took a significant amount of time to reach an optimal solution for graphs with number of vertices equal to or greater than 20. Due to this, a 2 minute SAT-solver timeout function was implement to exit the program if an optimal solution was unable to be achieved within the specified amount of time.

[1] For a more in-depth description of miniSAT, visit http://minisat.se/downloads/MiniSat.pdf

2.2 Approximation 1 Algorithm

The first approximating minimum vertex cover solver does the following:

- Pick a vertex with the highest amount of edges connected to it
- Add vertex to the vertex cover
- Remove all edges incident to this vertex from the edge list
- Repeat process until no edges remain

This algorithm will be denoted as APPROX-VC-1 for the rest of this report.

2.3 Approximation 2 Algorithm

The second approximating minimum vertex cover solver does the following:

- Pick an edge $\langle u, v \rangle$ from the graph edge list
- Add both u and v to the vertex cover
- Remove all edges incident to vertices u and v
- Repeat process until no edges remain

This algorithm will be denoted as APPROX-VC-2 for the rest of this report.

3 Analysis Method

The three algorithms were implemented to run concurrently using the POSIX Pthread Library in C++. A random graph generator was used to generate 10 different undirected graphs ranging in number of vertices from 5 to 50, incremented by 5. That is, graphs with 5, 10, 15,...,50 vertices were generated. Each of the 10 graphs in each vertex number category was run 20 times concurrently with all three algorithms.

The three algorithms were analyzed based on two criteria:

- Total running time
- Approximation ratio as compared to the output of CNF-SAT-VC (CNF-SAT-VC is guaranteed to be optimal)

For each set of graphs with the same number of vertices, the average and standard deviation was calculated for the two criteria previously mentioned. The results of this analysis is presented below.

4 Results and Analysis

4.1 Running Time

The total running time comparison between the three algorithms explored can be seen in Figure 1 below. The plotted points represent the average running time for all the graphs of the

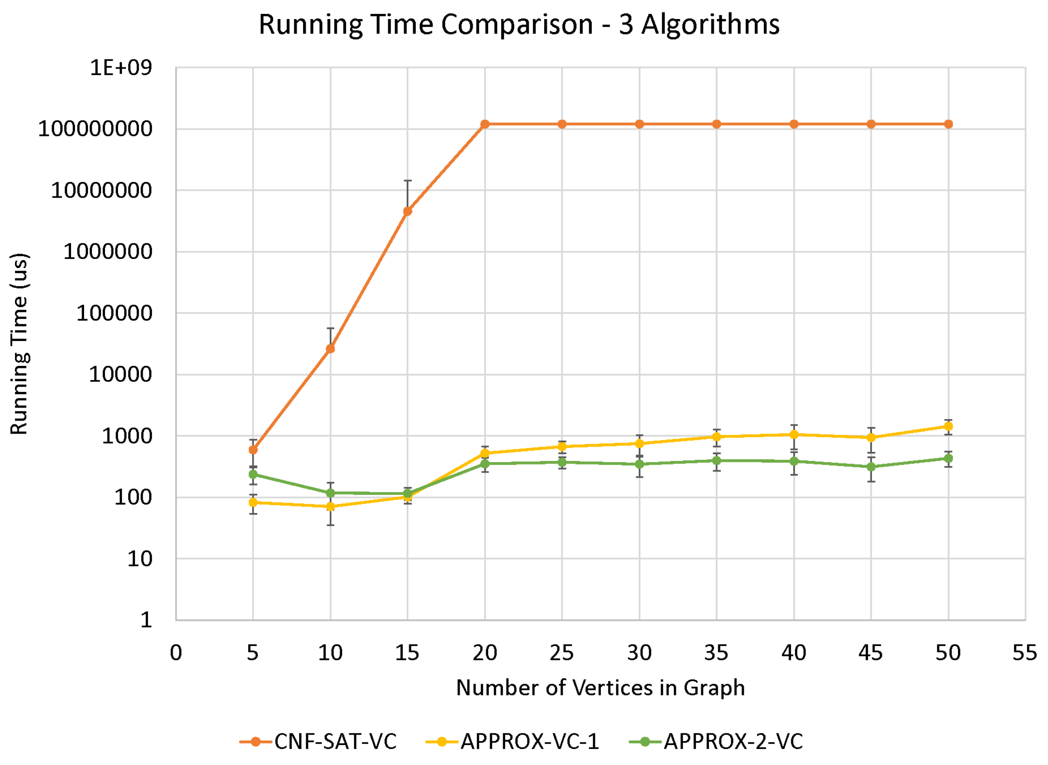

Figure 1: Running time comparison between CNF-SAT-VC, APPROX-VC-1 and APPROX-VC-2

same size and the error bars represent the standard deviation of the running time data for the respective graph sizes.

As it can be seen in Figure 1, the encoding used for MiniSAT exponentially increases the total running time with increasing number of vertices in the graph (logarithmic y-axis). At 20 vertices and above, the algorithm reached the 2 minute timeout (ie. 120,000,000 microseconds). The significant increase in running time with increasing graph size for CNF-SAT-VC is likely due to the encoding process implementing multiple nested for loops of complexity $O(n^3)$ which is of a significantly higher complexity than the two approximation algorithms. The running time of APPROX-VC-1 and APPROX-VC-2 are very comparable, with both having running times below 1 millisecond with graphs up to 50 vertices in size. Figure 2 below shows a closer running time comparison between the two approximation algorithms.

Figure 2 shows that APPROX-VC-1 executes faster than APPROX-VC-2 for graph sizes less than 20 vertices while above this range, APPROX-VC-2 seems to complete its process more quickly. The standard deviation increases for both algorithms with increasing graph size.
APPROX-VC-1 seems to consistently have a higher standard deviation, with significantly larger deviations occurring at graph sizes larger than or equal to 30 vertices. The trending increase in running time for APPROX-VC-1 is likely due to the algorithm having to determine the vertex with the largest number of incident edges before moving on to the edge removal step on that one vertex, determined to be of complexity $\Theta(|V^2|)$. APPROX-VC-2 simply chooses an edge at random, which is a faster process, and removes the incident edges to both vertices related to the chosen edge. The complexity of APPROX-VC-2 is determined to be of $\Theta(|V|)$. The large standard deviation for APPROX-VC-1 with larger graphs is likely due to the variety of graphs

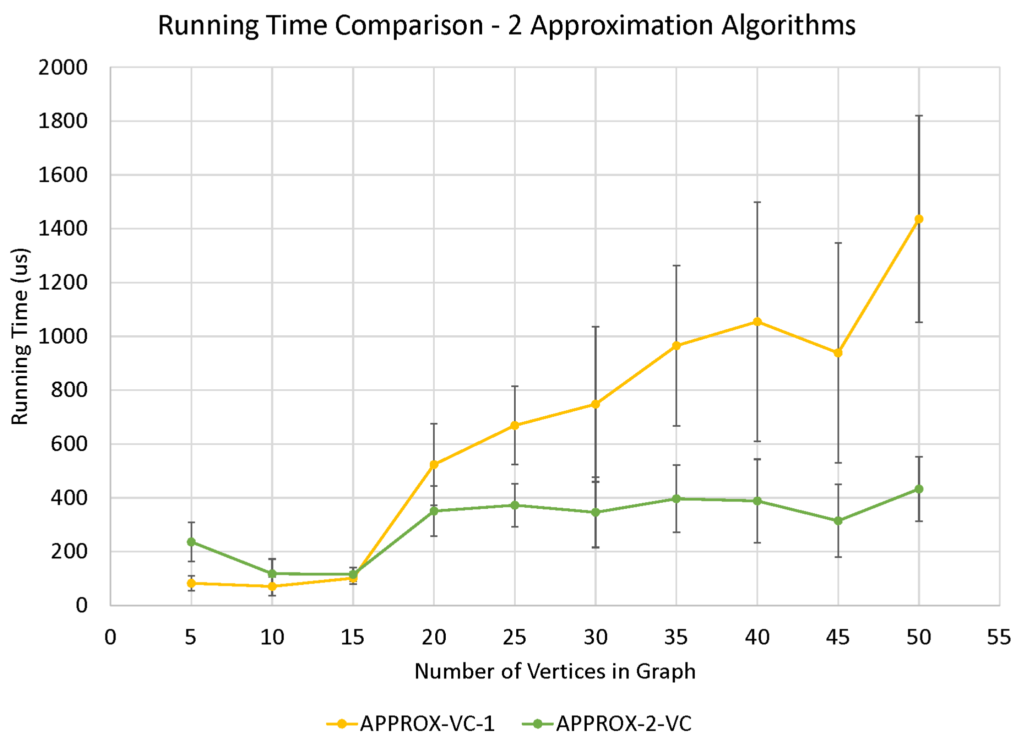

Figure 2: Running time comparison between APPROX-VC-1 and APPROX-VC-2

generated. Some of the graphs may have a small amount of vertices with edges to the rest of the vertices, while other graphs may have graphs with all vertices having no more than 1 to 2 incident edges, driving up the total running time of this algorithm.

4.2 Approximation Ratio

Another key performance benchmark is the approximation ratio defined as follows:

$$ApproximationRatio = \frac{sizeof(ApproximationAlgorithm)}{sizeof(CNF - SAT - VC)} \tag{1}$$

The results of this analysis can be seen in Figure 3 below. The results suggest that on average, APPROX-VC-1 more accurately approximates the minimum vertex cover of an undirected graph and with very small amounts of deviations. APPROX-VC-2 on the other hand, generally approximates the graph to be between 1.4 and 1.6 times the ideal minimum vertex cover. The approximation ratio of APPROX-VC-2 could be potentially trending upwards with increased graph sizes and it can be seen from 10 vertices to 15 vertices. This needs to be further explored to to confirm. The large standard deviations seen on APPROX-VC-2 are likely due to the random aspect of the algorithm. By randomly choosing an edge to add its vertices to the vertex cover and removing their incident edges, it is just as likely to choose the vertices with the most incident edges or the vertices with the fewest. As a consequence, multiple runs of the same graph will result in larger standard deviations.

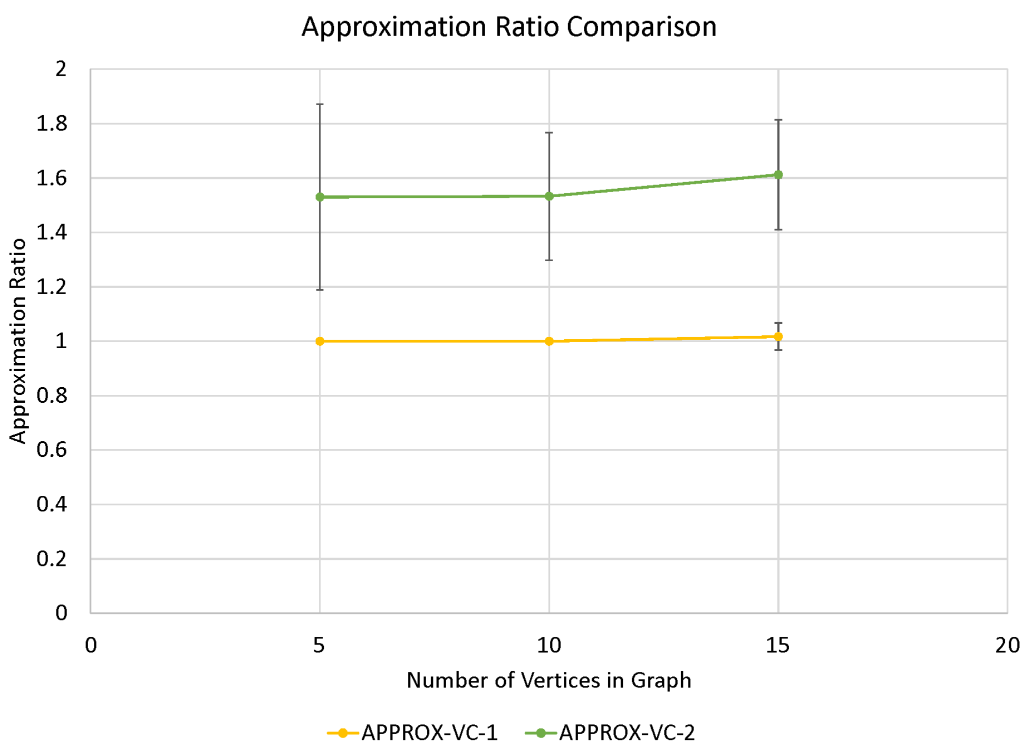

Figure 3: Approximation ratio comparison between APPROX-VC-1 and APPROX-VC-2

5 Conclusions

The performance of each algorithm varies from one another. Each run was performed using the same hardware and operating system to maintain consistency in results. From a total run-time perspective, the algorithm associated with APPROX-VC-2 was the fastest performer for graphs with a size greater than 15 while APPROX-VC-1 performed the fastest for graph sizes less than or equal to 15 vertices. While CNF-SAT-VC was always guaranteed to have the minimum vertex cover, the total run-time of the algorithm grew exponentially with increasing graph size.

From an approximation ratio perspective, APPROX-VC-1 was consistently giving a solution closer to the true answer than APPROX-VC-2, with the solutions being exactly correct for graphs of size 10 or less with very small standard deviations. APPROX-VC-2 on the other hand had consistently derived solutions that were greater than 40% higher than the ideal solution with standard deviations as high as 0.3 from its average value.

In conclusion, APPROX-VC-1 shows to be a good alternative algorithm for a minimum vertex cover solver of an undirected graph, with a running time significantly smaller than CNF-SAT-VC. For further understanding, it is recommended to further compare the running time between APPROX-VC-1 and APPROX-VC-2 with graph sizes significantly larger than what was explored in this report. Also, it is recommended to further improve the encoding algorithm to CNF in order to further explore the approximation ratio of the two approximation algorithms for larger graphs.