Analysis of Various Vertex Cover Algorithms

University of Waterloo

ECE 650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

Winter 2023

Taiwo Oluwaseyi - 20990315

Akindipe Anjolaoluwa - 21032665

Contents

1 Abstract

The minimum number of nodes connecting or covering all edges of a graph is known as minimum vertex cover. Because finding the smallest vertex is NP-Complete, it is a widely common optimization problem. NP-Complete issues can be demonstrated in polynomial time, but currently, no algorithms can solve them in polynomial time. In this program, we studied the efficiency of three vertex cover algorithms: Approximate Vertex Cover CNF-SAT-VC, APPROX1-VC, and APPROX-2-VC. We also explored their refinements in order to compare them.

We implemented these algorithms in C++ and benchmarked 100 graphs for each vertex size ranging from [5,50] by measuring their runtime in nanoseconds and the approximate ratio, which is the ratio of the number of vertices obtained from each approach to the ideal number of vertex covers obtained from our CNF-SAT-VC or 3CNF-SAT-VC. The inputs for this experiment were obtained through a software executable called ”graphGen,” which assisted in generating graphs with the same number of edges for a given number of vertex but not necessarily the same edges. The ”graphGen” program takes a number as input, representing the number of vertices in the graph it would produce; hence, we generated 10 graphs for each vertex ranging from 5 to 50 while incrementing the vertexes by 5.

We noticed that CNF-SAT-VC and its refinement CNF-3-SAT-VC provided the most accurate answer of all other methods but were the slowest when increasing the vertices by a certain amount. We also discovered that APPROX-2-VC was generally the quickest of all methods but provided the least accurate vertex coverage. Also, its inaccuracy impacts its refinement, thus adding more time to get a more accurate result. In the end, APPROX-1-VC was near in accuracy but much faster than the CNF algorithms. Therefore, while requiring more time than its APPROX-2-VC counterpart, it was more practicable due to its capacity to provide a result closest to the ideal outcome.

2 Vertex Cover Algorithms

2.1 CNF-SAT-VC

The CNF-SAT problem is known to be NP-complete, meaning it is one of the most complex problems in the complexity class NP. This algorithm implements a polynomial time reduction to a given vertex-cover problem by reducing the problem to a CNFSAT problem and using a boolean satisfiability Solver, ”Minisat,” to find the optimal vertex cover. The vertex cover problem is reduced to set clauses in order for the minisat solver to determine the satisfiability of the vertex cover problem. If the Minisat can determine a satisfiable combination of literals, then the vertex cover exists for the current graph.

2.2 CNF-3-SAT-VC

This algorithm implements a polynomial time reduction to a given vertex-cover problem by reducing the problem to a 3 CNF SAT problem. The vertex cover problem is reduced to a set of clauses that have at most 3 literals in order for the minisat solver to determine the satisfiability of the vertex cover problem. MiniSat will then try to find a satisfying assignment of truth values to the variables that make the formula evaluate to true or report that no such assignment exists (not satisfiable).

The clauses are reduced to at most 3 literals by replacing every clause of the form $(\mathcal{L}_0 \vee \ldots \vee \mathcal{L}_n)$ with 3-literal clauses $(\mathcal{L}_0 \vee b_0) \wedge (\neg b_0 \vee \mathcal{L}1 \vee b_1) \wedge \ldots \wedge (\neg bn-1 \vee \mathcal{L}_n)$ where b_i are fresh atomic propositions not appearing in F.

2.3 APPROX-1-VC

Using a greedy approach, this algorithm presents an approximation solution to the vertex cover problem. The algorithm repeatedly chooses a vertex with the highest number of edges that are incident to it. This vertex is then added to the vertex cover list, and we remove all the edges that are incident to this vertex. This process is repeated until no edges remain.

2.4 APPROX-2-VC

This algorithm uses a greedy approach to offer an approximate solution to the vertex cover issue. It starts by selecting an arbitrary edge <u, v> from the collection of edges and adds the selected edges u and v to the vertex cover. It then removes all edges that are either incident on the selected vertices or have already been covered. This process continues until all edges are covered.

2.5 REFINED-APPROX-1-VC & REFINED-APPROX-2-VC

This algorithm is a refinement of the APPROX-VC-1 and APPROX-VC-2 algorithms. It first implements the APPROX-VC-1 or APPROX-VC-2 algorithm and finds an initial vertex cover. Upon getting this, it then checks each vertex in the initial cover to see if it can be removed while maintaining a vertex cover, i.e. the vertexes left still cover all possible edges in the graph. The algorithm checks each edge incident on the vertex to determine if a vertex can be removed. If at least one of the incident edges is not in the initial vertex cover, then the vertex cannot be removed and is kept in the refined cover. If all the incident edges on the vertex are already in the initial cover, the vertex can be removed and is not included in the refined cover.

3 Design

3.1 Efficiency

In order to measure the performance of each algorithm, we use two metrics to quantify their efficiency the run-time of each algorithm and the approximation ratio. The approximation ratio is the ratio of the vertex cover produced to the optimal minimum vertex cover. This optimum vertex cover can be obtained from our CNF-SAT-VC or CNF-3-SAT-VC algorithms as they always result in an accurate minimum vertex cover; thus, this is our benchmark for our ratio. Its equation is shown below:

$$(AR) = \frac{VC_{output}}{VC_{optimal}} \tag{1}$$

where:

AR = Approximate Ratio

$\text{VC}_{output} = Output \quad Vertex \quad Cover \quad from \quad Algorithm$

$VC_{optimal} = Optimal \quad Minimum \quad Vertex \quad Cover$

The runtime is simply the time it takes for each algorithm to process each graph and produce a vertex cover as the result.

3.2 Methodology

We followed the methods below to get a fair outcome for each algorithm. We used multi-threading to allow each algorithm to execute in parallel. This significantly decreased the time required to run the graphs and ensured that each method was performed independently. This also enabled us to define a standard timeout to cancel the algorithms once a particular period had passed (60 seconds). Without multi-threading, each algorithm would have to wait for one another before starting, and because the majority of the bottleneck would be in executing the CNF algorithms, each graph for each algorithm would take longer to execute.

Each algorithm is tested with the same graph for a specific run. There are ten graphs for a specific number of vertices ([5,50], with increments of 5) with the same number of edges but not the necessarily the same edges. The ”graphGen” program provides these graphs, which are first gathered into a text file to have 100 graphs. After that, we modified the software to output each algorithm’s run-time and approximation ratio and then piped the output into a CSV file, yielding 600 results.

We made sure to timeout these algorithms after 60 seconds of run-time and took this into account in our data. The mean run-time and approximate ratios for data were calculated using a mix of Python libraries (e.g., numpy, pandas). Then, to analyze the various metrics, we constructed multiple line graphs using the ”matplotlib” library in Python and compared the graphs of one algorithm to another.

4 Result and Analysis

4.1 Analysis of CNF-SAT-VC

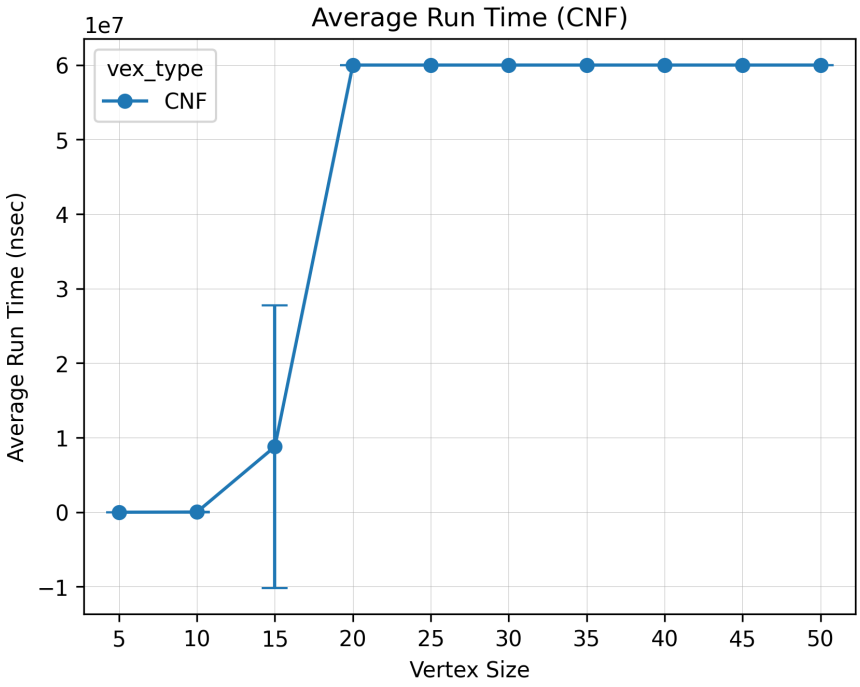

Figure 1: Average running time and the standard deviation of the CNFSAT-VC

Figure 1 shows the average running time and the standard deviation of the CNFSAT-VC algorithm for different numbers of vertices ranging from (5 to 50) in increments of 5. From the graph, we can observe that the CNF-SAT-VC running time increases exponentially with the increase in the number of vertices. With smaller values of vertices, it takes lesser time to find satisfiability because there are fewer literals and clauses to check. For larger values of vertices, a larger number of literals and clauses contributes to increased time consumption. We can also observe from the graph that after vertex (25), the running time flatlines at 60 seconds. The flatlines result from a timeout implemented to interrupt the sat solver if the solver cannot determine a satisfactory combination of literals within the time frame of 60 seconds.

4.2 Analysis of CNF-3-SAT-VC

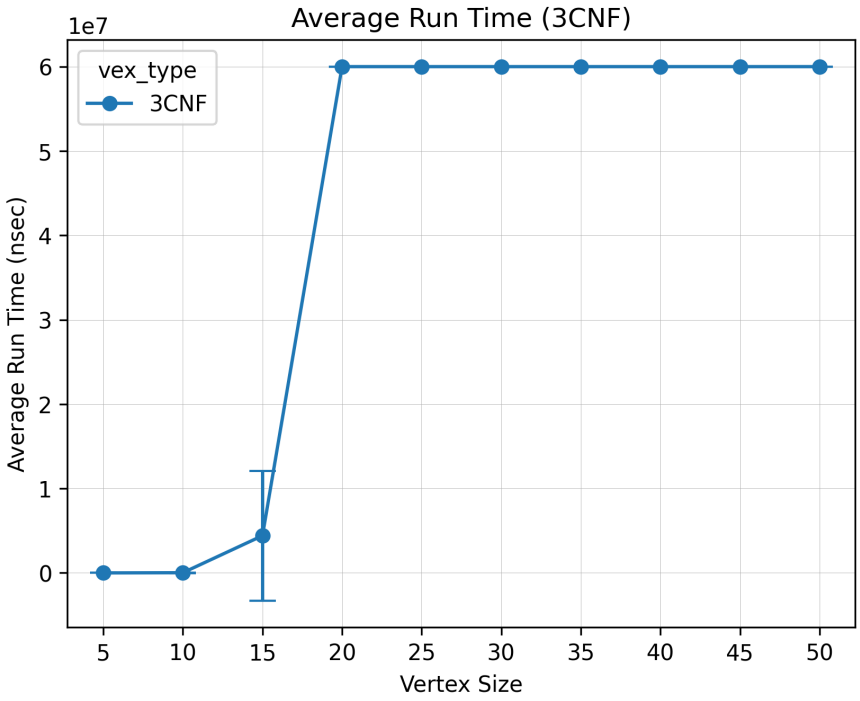

Figure 2: Average running time and the standard deviation of the 3CNF-SAT-VC

Figure 2 shows the average running time and the standard deviation of the CNF-3-SAT-VC algorithm for different numbers of vertices ranging from (5 to 50) in increments of 5. From the graph, we can observe that the 3CNF-SAT-VC running time increases exponentially with the increase in the number of vertices. With smaller values of vertices, it takes lesser time to find satisfiability because there are fewer literals and clauses to check. For larger values of vertices, a larger number of literals and clauses contributes to increased time consumption.

We can also observe from the graph that the 3CNF can provide a vertex cover for some vertex values faster than the CNF. This is because the 3CNF algorithm is an optimized version of the CNF algorithm, as it only allows for 3 literals in a clause at most. This fact reduces the time takes to determine satisfiability for vertex cover problems. Once we reach a vertex size of (25), we observe that the running time flatlines at 60 seconds same as the CNF. The flatlines result from a timeout implemented to interrupt the sat solver if the solver cannot determine a satisfactory combination of literals within the time frame of 60 seconds.

4.3 Analysis of CNF-SAT-VC & CNF-3-SAT-VC

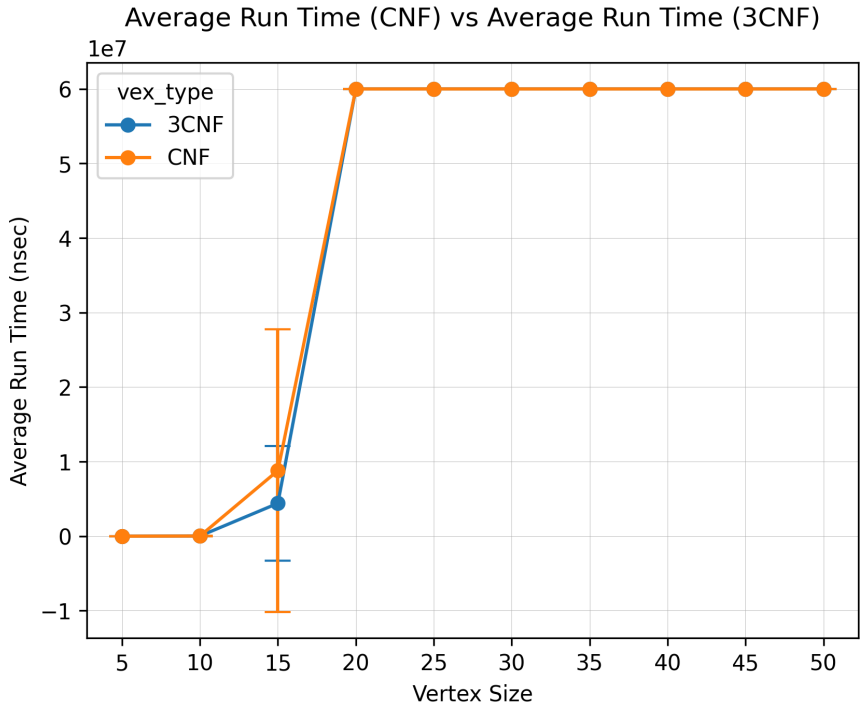

Figure 3: Average Run Time (CNF) vs Average Run Time (3CNF)

Figure 3 shows the average running time and the standard deviation of the CNF-3-SAT-VC algorithm for different numbers of vertices ranging from (5 to 50) in increments of 5. From the graph that the 3CNF can provide a vertex cover for some vertex values faster than the CNF. This is because the 3CNF algorithm is an optimized version of the CNF algorithm, as it only allows for 3 literals in a clause at most. This fact reduces the time it takes to determine satisfiability for vertex cover problems.

4.4 Analysis of APPROX-1-VC & APPROX-2-VC

Figure 4 shows the average run time and standard deviation for the APPROX-1-VC and APPROX-2-VC. For the most part, the APPROX-2-VC algorithm takes less time to run than the APPROX-1-VC algorithm, which applies as the number of vertices grows. This may be due to the fact that the APPROX-1-VC algorithm has to loop through the vertices with the most incident edges at each iteration and remove all the edges that are incident to it, whereas the APPROX-2-VC algorithm can choose a vertex at random and remove all the edges that are incident to it, cutting down on time. This means that as the vertices grow, it may take more time for the APPROX-1-VC algorithm to find the vertex with the most incident edges at each iteration. But we can also observe from figure 5 that the approximation ratio for APPROX-1-VC is closer to ”1” than that of APPROX-2-VC, even though the latter is faster. This shows that what the APPROX-1-VC algorithm lacks in run-time, it makes up for in the accuracy of getting the optimal solution.

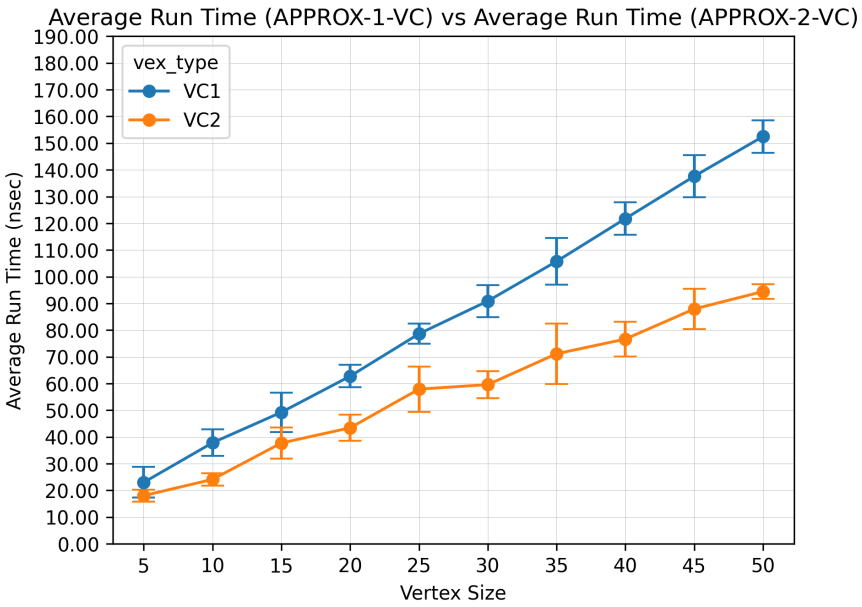

Figure 4: Average Run Time (APPROX-1-VC) vs Average Run Time (APPROX-2-VC)

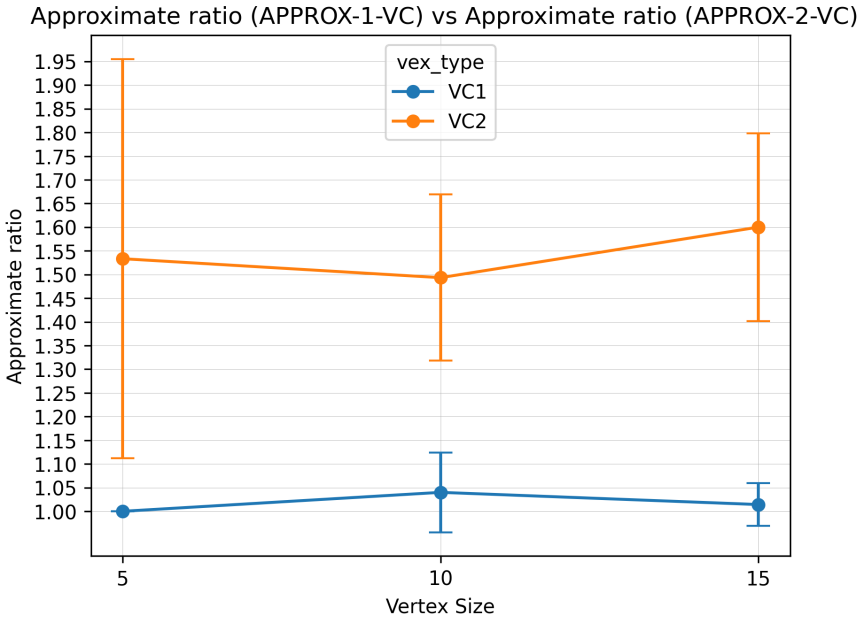

Figure 5: Approximate Ratio for APPROX-1-VC and APPROX-2-VC

4.5 Analysis of REFINED-APPROX-1-VC & REFINED-APPROX-2-VC

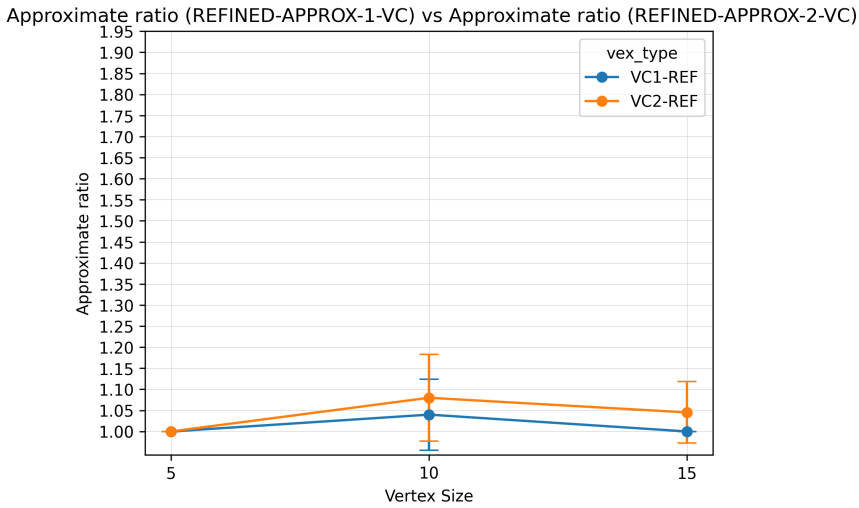

Figure 6: Approximate Ratio for REFINED-APPROX-1-VC and REFINED-APPROX-2-VC

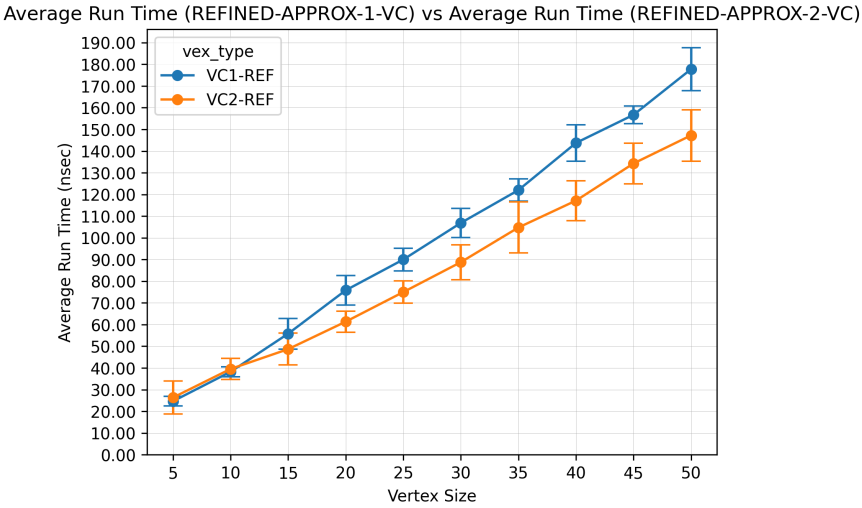

Figure 7: Runtime for REFINED-APPROX-1-VC and REFINED-APPROX-2-VC

From figure 7, After refining both APPROX-1-VC and APPROX-2-VC, we noticed that it still takes more time for APPROX-1-VC than APPROX-2-VC. Thus trying to optimize the accuracy of APPROX-1-VC has remained the same in terms of runtime to that of APPROX-2-VC refinement. We can also see from figure 6 that the refinement of APPROX-VC-2 helps improve its approximation ratio but it is still not as accurate as APPROX-VC-1. This could result from the inaccuracy of the APPROX-VC-2 algorithm, making it harder to refine to the most optimal solution.

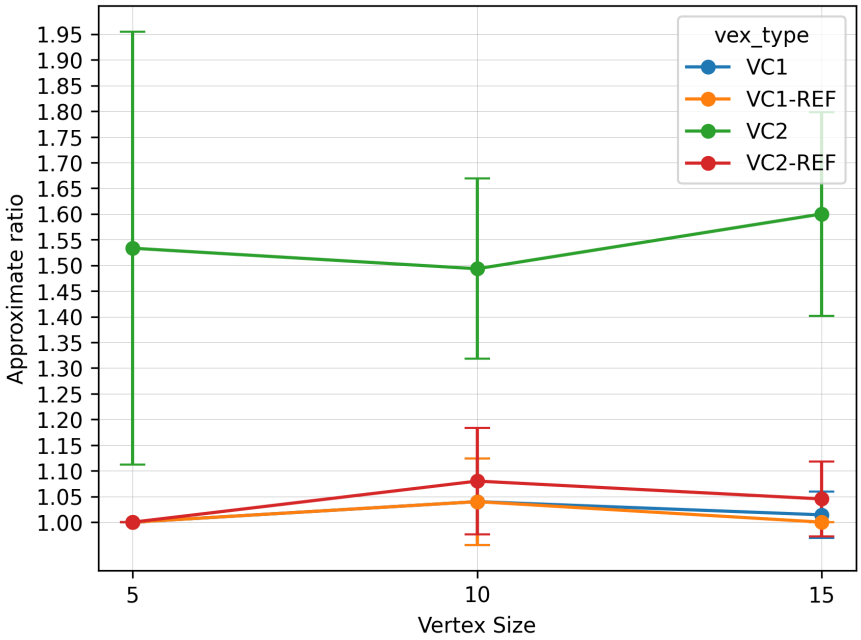

Figure 8: Approximate ratio for all Approximate algorithms

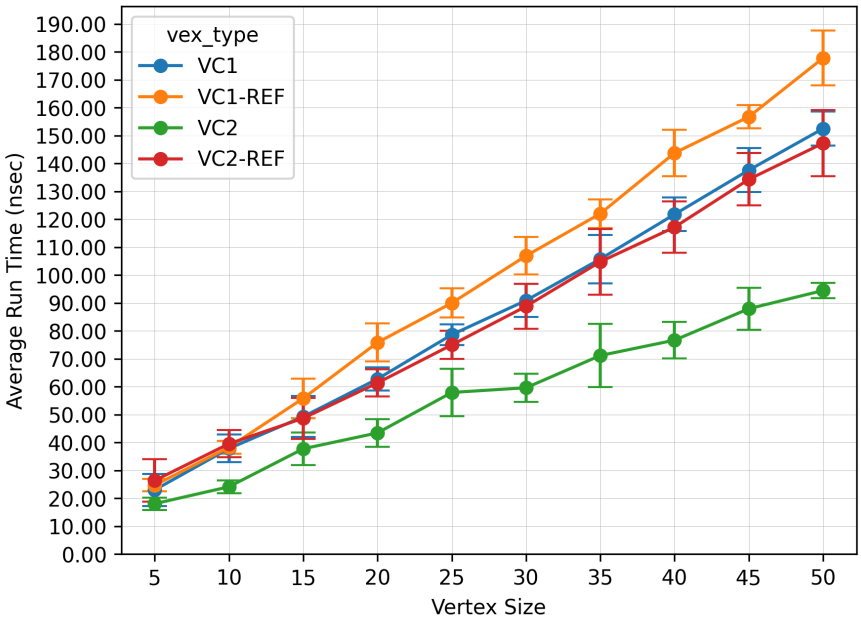

Figure 9: Average Run Time for all Approximate algorithms

We can also notice from figure 8 and 9, that the refinements add more to their respective approximate ratio's run-time but also decrease their approximation ratio in order to find a more optimal result.

4.6 Analysis of All Algorithms

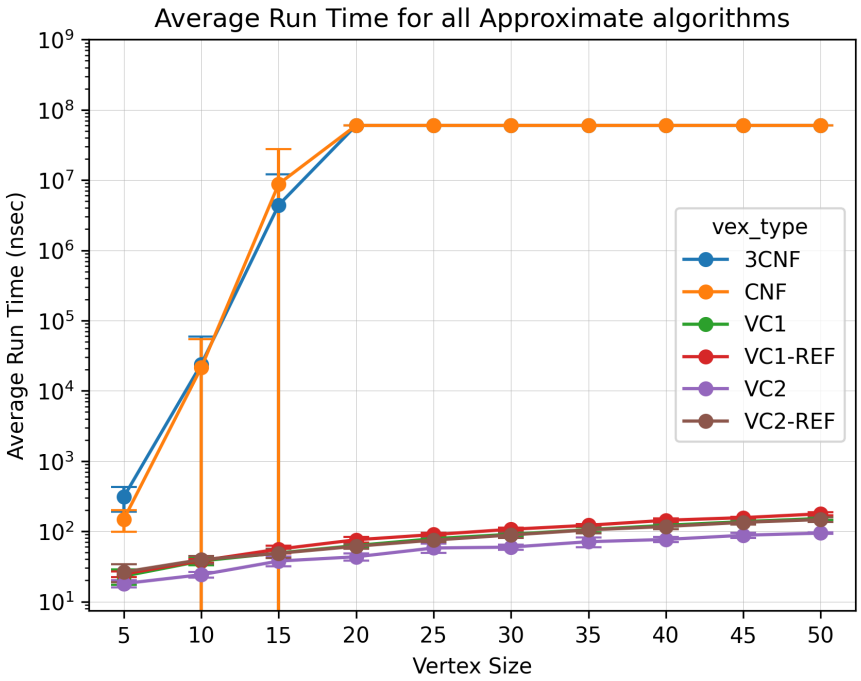

Figure 10: Average Run Time for all Algorithms

When we compare the run-times of all algorithms in figure 10, we see that the CNF methods take the longest time to get the best minimal vertex cover. As the number of vertices increases, so does the time required to discover a satisfactory solution in an exponential way. Optimizing the CNF-SAT-VC to CNF-3-SAT-VC reduces run time somewhat, but the CNF algorithms remain significantly slower than their approximation counterparts. After the CNF algorithms, we observe the APPROX-1-VC algorithm and its refinement. Its execution time is significantly less than that of the CNF algorithms, although it is still slower than that of APPROX-2-VC and its refinement. This might be related to the time required to always identify the vertex with the biggest cover in a graph, and as the number of vertices rises, so does the run time. This may also be observed in its refinement, which accomplishes the same thing but eliminates unnecessary vertices. Lastly, APPROX-2-VC was proven to be the quickest, even with graphs with a large number of vertices, however as shown in figure 8, the algorithm's speed does impair its accuracy.

5 Conclusion

We determined that CNF-SAT-VC and CNF-3-SAT-VC were the most accurate algorithms. However, as the number of vertices in a graph increases, the computing time for these algorithms increases exponentially, eventually hitting our timeout of 60 seconds. As a result, these techniques may be utilized anytime the optimal vertex cover is required in a big graph, and speed is not a constraint.

If speed is prioritized over accuracy/precision, APPROX-VC-2 and REF-APPROX-VC-2 might be employed

since they are the quickest algorithms with the shortest run-time. Because their approximation ratio is the highest, they are most likely to provide the least optimum minimal vertex cover. If both time and an optimal solution are required, APPROX-VC-1 is the most likely choice because it is much quicker than the CNF algorithms and more accurate than the second approximate algorithm (APPROX-VC-2). REF-APPROX-VC-1 could also be used in order to increase the accuracy of APPROX-VC-1