ECE 650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Project: Quantitative Analysis of Mini-Vertex Cover Algorithms

Submitted by:
Karthik Prakash Sivakumar- 21043979
Prerona Ghosh- 21048873

Date: April 12, 2023

Contents

1 Introduction

The aim of the project is to examine some methods used to solve the parameterized vertex cover problem, which entails locating a subset of vertices in a graph G(V, E) such that every edge (u, v) is a subset of E and at least one of the vertices u or v is in the vertex cover.

Throughout the project, we looked at six different algorithms. Of them, two involve encoding the vertex cover issue into CNF-SAT-VC and CNF-3-SAT-VC. Also, we used the Approx-VC-1 and Approx-VC-2 approximation techniques to calculate the vertex cover in the graph. Refined-Approx-VC-1 and Refined-Approx-VC-2 two enhanced versions of the previously described Approx-VC-1 and Approx-VC-2 algorithms, were also taken into consideration.

One significant application of the vertex cover problem is Minimal Vertex Cover (MVC). It entails determining the graph's minimal vertex cover. MVC has several practical uses, including in bioinformatics, scheduling, and social network research. It can be used, for instance, in social network research to pinpoint a small group of people with significant influence over a sizable portion of a network. It can be applied to scheduling to determine the minimum of resources needed to complete a task, and to bioinformatics to determine the minimum of genes or proteins required for a biological process.

Efficient algorithms for solving the minimum vertex cover problem have practical implications in these applications and can lead to optimized solutions.

2 Algorithms

The six algorithms used for analysis have been described below.

2.1 CNF-SAT-VC

The reduction of vertex cover to CNF-SAT is a polynomial time reduction. We have utilized the same encoding as in assignment 4 for this algorithm. The graph is implemented as a collection of literals and CNF-format clauses, where we know that CNF is a conjunction of literals in disjunction, i.e the literals are OR with each other and the clause are AND with each other. This parameter is fed to the MiniSat library in C++, which checks if a vertex cover for the current vertex cover size already exists. Vertex cover size is increased with each cycle until the minimal cover is discovered.

2.2 CNF-3-SAT-VC

We utilized the same clauses as in A4 for this method, but each clause is in 3-CNF form and can only include a maximum of three literals. The code is an implementation of a known NP-complete technique for finding a vertex cover with the smallest size in a given graph. The solution resolves this issue by using the MiniSat SAT solver and the 3-SAT problem. Here is a quick summary of the code used:

1. Initialize the vertex cover to be empty.
2. For each k from 1 to the number of vertices in the graph, do the following:

a. Create a new instance of the MiniSat solver.
b. Create a matrix of k literals for each vertex in the graph.
c. Add the following clauses to the solver:
i. At least one vertex from the list of vertices is present in the vertex cover
ii. No one vertex can appear twice in a vertex cover
iii. No more than one vertex appears at any position of the vertex cover
iv. Every edge is incident to at least one vertex in the vertex cover
d. Use MiniSat solver to solve the 3-SAT instance.
e. If a solution is found, return the vertex cover from the solution.

3. Give back the empty vertex cover if none of the k problems have a solution.

The expansion used to convert k-CNF to 3-CNF is:

$$(A \vee B \vee x_0) \wedge (\sim x_0 \vee C \vee x_1) \wedge (\sim x_1 \vee D \vee x2)...(x_{n-1} \vee Y \vee Z)$$

where $x_0...x_n$ is the new literals added to encode into the 3-CNF clause.

The algorithm performs tests on all k possible vertex coverings. The problem is encoded as a 3-SAT instance for each k, and the MiniSat solver is used to resolve it. In the event that a solution is discovered, it retrieves the associated vertex cover and returns it. It returns an empty vertex cover if there isn't a solution for any k.

Note that the code also contains a method for turning the clauses in the 3-SAT instance into 3-CNF clauses, hence reducing the number of clauses. To increase the solver's effectiveness, this is done.

2.3 APPROX-VC-1

This approach gives an approximate estimation for the Vertex Cover problem. A list of edges and a certain number of vertices are provided as input, and the output is a vertex cover of the input graph. The algorithm calculates the degree of the vertices of each edge in the edge list.

The algorithm works by looping over the list of edges within the edge list and steps up the degree of the vertices of the given edge. It then selects the vertex which has been provided the maximum number of times and adds it to the result. Following that, it removes all the edges incident to that vertex. The process is repeated until no edges remain.

$$Time\ Complexity = O(mn)$$

where m is the number of edges and n is the number of vertices.

The time complexity is as above because the algorithm iterates over all edges in the worst case and checks the degree of each vertex for each edge.

The method won't always find the minimal vertex cover, but it will almost always find one that is no larger than twice the size of the smallest vertex cover. This is because every edge must have at least one vertex covering it, and when the additional vertex is added, more edges are also covered.

2.4 APPROX-VC-2

This approach is also a vertex cover problem approximation algorithm. All the vertices connected to an edge are eagerly chosen by the method and added to the vertex cover set. After that, it eliminates all edges attached to these vertices and keeps running until all edges have been removed.

$$Time\ Complexity = O(mlog(m))$$

where m is the number of edges in the input graph. This is because the algorithm has to sort the result vector at the end, which takes $mlog(m)$ time.

Compared to the prior approach, this one offers a stronger approximation guarantee. The size of the vertex cover generated by this approach is at most twice as large as the ideal vertex cover because it has an approximation ratio of 2. The best outcome might not always be achieved by this algorithm, though.

2.5 REFINED-APPROX-VC-1

This is a refinement algorithm for the vertex cover problem. It is an improvement over the previous approximation algorithm APPROX-VC-1.

Similar to APPROX-VC-1, the algorithm starts with a greedy approach to creating a vertex cover. The vertex with the highest degree is added to the vertex cover after iterating over all the edges. Once all edges have been eliminated, it continues the process until none remain.

The algorithm outperforms APPROX-VC-1 in the refinement phase. It examines each vertex in the vertex cover and searches for any nearby vertices that are not in the vertex cover for each vertex. The current vertex gets deleted from the vertex cover if there are no similar nearby vertices. By eliminating vertices that were included in the greedy technique but were not necessary to cover all the edges, this step raises the quality of the vertex cover.

Overall, this refinement algorithm gives a better approximation for the minimum vertex cover problem compared to the APPROX-VC-1 algorithm.

2.6 REFINED-APPROX-VC-2

This is also a refinement of the vertex cover approximation algorithm. It creates an initial approximation by greedily adding vertices that cover uncovered edges similarly to the prior technique. Examining each vertex in the cover and eliminating it if it has no uncovered neighbors, then improves this approximation.

This algorithm is similar to the previous one with regards to implementation, with the exception that each iteration of the main loop only takes into account the first edge in the list. This strategy might shorten the algorithm's execution time, but it might also provide a less-than-ideal vertex cover.

Overall, this refining process can enhance the quality of the vertex cover approximation produced by the fundamental greedy algorithm, but its performance is dependent on the input graph's structure and the particular algorithmic parameters employed.

3 Multithreading

We have implemented the solution to all six algorithms using Multithreading, with the goal of achieving concurrency and maximum utilization in terms of execution time. For this, we designed the thread skeleton using 'pthreads'. There are a total of 7 threads running concurrently as mentioned in the project requirement. In the main function, the program creates a main I/O thread, which parses the user input of V and E and creates the initial list of edges, and also handles stipulated errors in the provided input. The threads for each of the six algorithms then run in parallel, making use of the previously mentioned edge list. The running time of each thread is calculated using the 'high_resolution_clock()' function for each of the algorithms.

4 Methods of Analysis

Running time and Approximation ratio are the two performance metric components that we concentrate on for the analysis. For running time, the goal is to quantify the approximation ratio and running time for graphs formed by graphGen for various values of V (number of vertices). The graph has an interval of 5, from V=5 to V=50.

We generate 10 sets of graphs for every vertex, calculating their approximate ratios as well as their running times. Next, compute the mean and standard deviation for each value of V over those 100 samples.

4.1 Running Time

The pthread utility provides us with the CPU execution time for that thread, which we use to calculate the running time of an algorithm. We can determine the running time for each algorithm and the maximum CPU time each algorithm uses based on the number of vertices provided as input.

4.2 Approximation Ratio

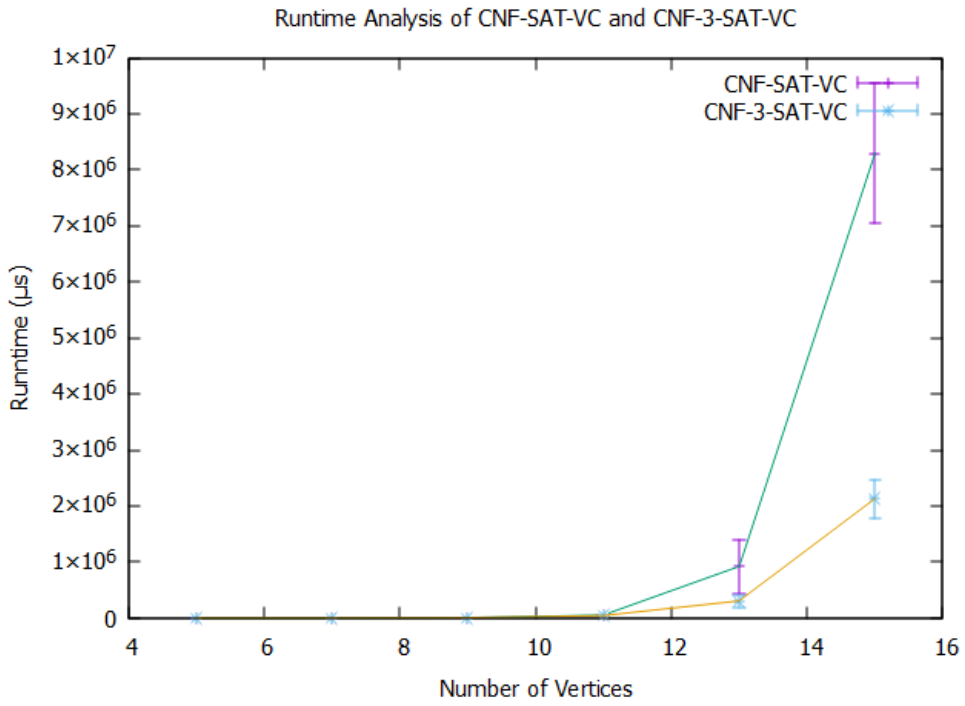
For approximation ratio, we consider the optimal vertex cover to be given by the CNF-SAT-VC algorithm. We calculate the Approximation Ratios for Approximation algorithms and the Refined Approximation algorithms using the following generic formula:

$$Approx\ Ratio = \frac{VertexCover - Size(Algorithm)}{VertexCover - Size(CNF - SAT)}$$

5 Analysis

This section provides runtime and approximation ratio analysis for each of the six vertex cover algorithms.

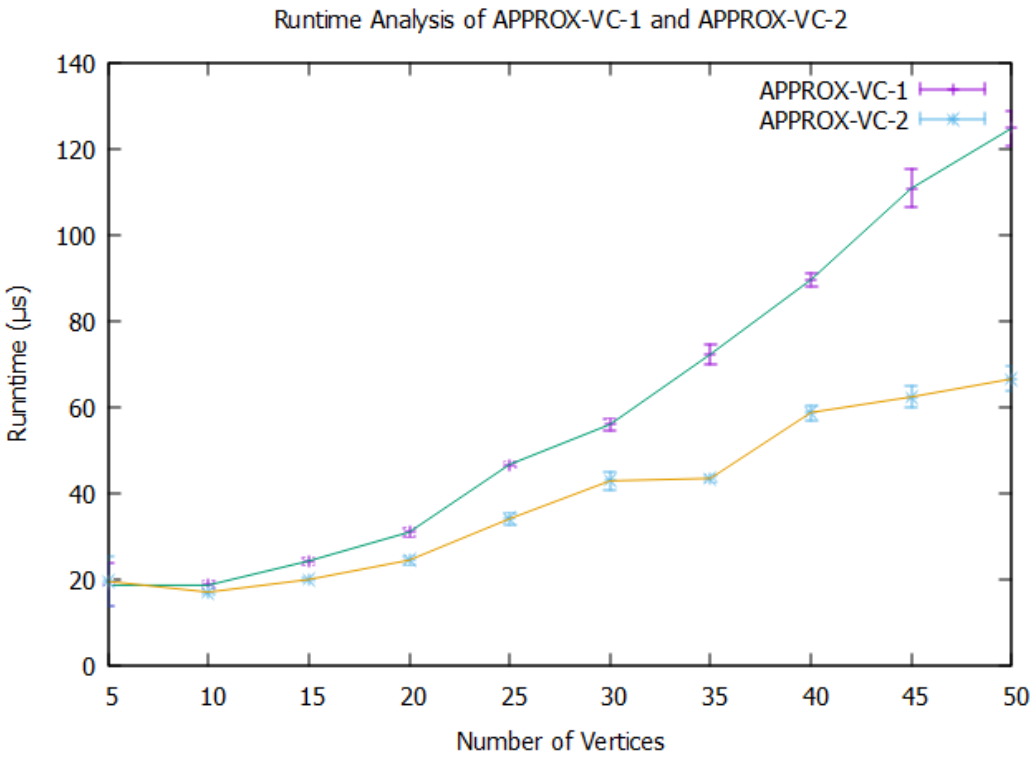
5.1 Running Time of CNF-SAT-VC and CNF-3-SAT-VC



This approach is expected to provide optimal vertex cover solutions C for a given graph G = (V, E), such that C is a subset of G and each edge E is incident to at least one vertex in C. The mean and standard deviation analyzed over 10 graphs with vertices in the range of 5 to 50 has been plotted in the above figure. We can see that the runtime for CNF-SAT is more than 3-CNF SAT and thus, converting k-CNF to 3-CNF has improved the performance. There is an exponential increase in runtime for V >15 for CNF-SAT, whereas it is lesser for the 3-CNF SAT algorithm and the graph is expected to steadily with an increase in the number of vertices. To handle the cases where the CNF-SAT-VC and CNF-3-SAT-VC threads require a large amount of time to execute, we have introduced a timeout of 120 sec. Eventually, we found out that for vertices greater than 15, it gets timed out because the number of clauses is also increasing in such cases.

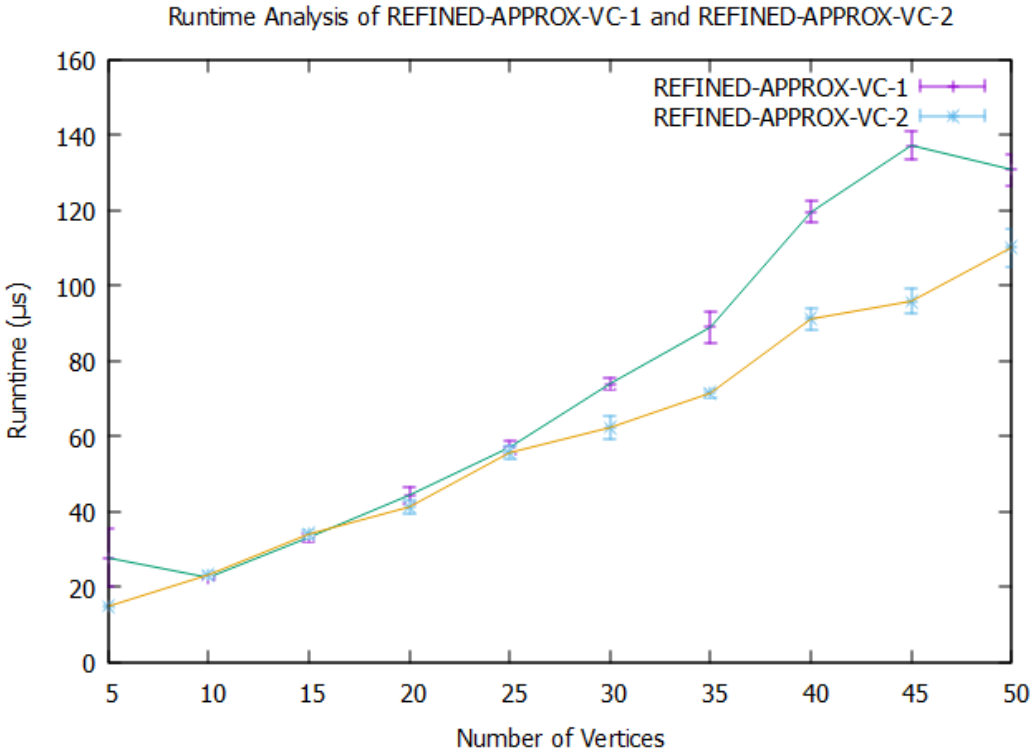
Also, the error bar that represents the standard deviation is increased greatly after vertex 15, which shows that the running time calculation starts to vary a lot even if different graphs with the same vertices have been provided.

5.2 Running Time of APPROX-VC-1 and APPROX-VC-2



In this section, we have picked two approximation algorithms and plotted the mean and standard deviation of runtime for every value of V within 5-50. As discussed in the previous sections, both of these algorithms are distinct from each other in the way that they pick the vertex cover. As depicted in the above figure, the runtime of both algorithms increases with an increase in the size of the graph (V). However, the rise is not exponential as compared to CNF-SAT algorithms. The runtime of APPROX-VC-1 is higher than that of APPROX-VC-2 because the former tries to find the highest degree vertex as opposed to the latter algorithm, which fetches each edge without any constraints. APPROX-VC-2 does not find the highest degree vertex and does not go back to the vertex that has already been visited. In turn, the number of vertices to be traversed over in the next loop is lesser. Thus, the time complexity of APPROX-VC-1 is higher than that of APPROX-VC-2.

5.3 Running Time of REFINED-APPROX-VC-1 and REFINED-APPROX-VC-2



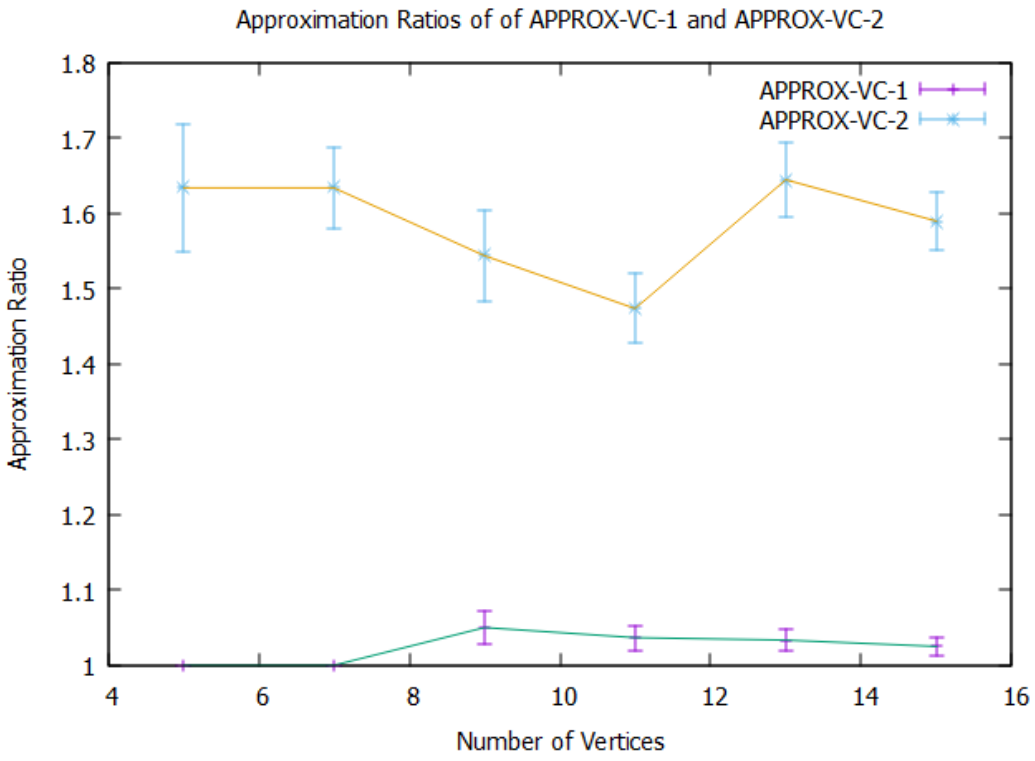
In this section, we have plotted the mean and standard deviation of running times for Refined Approximation Algorithms, which aims at removing any extra vertices from the approximation algorithm vertex covers if they still leave a vertex cover. The idea behind this refinement algorithm is to get closer to the minimum vertex cover of a particular graph.

The runtime of REFINED-APPROX-VC-1 is higher than that of REFINED-APPROX-VC-2, in alignment with the behavior of the respective approximation algorithms as both these algorithms work with the result of their respective approximation algorithms.

5.4 Approximation Ratio of CNF-SAT-VC and CNF-3-SAT-VC

Since we have considered CNF-SAT as the optimal solution, the approximation ratio will be 1 for both CNF-SAT and CNF-3-SAT algorithms.

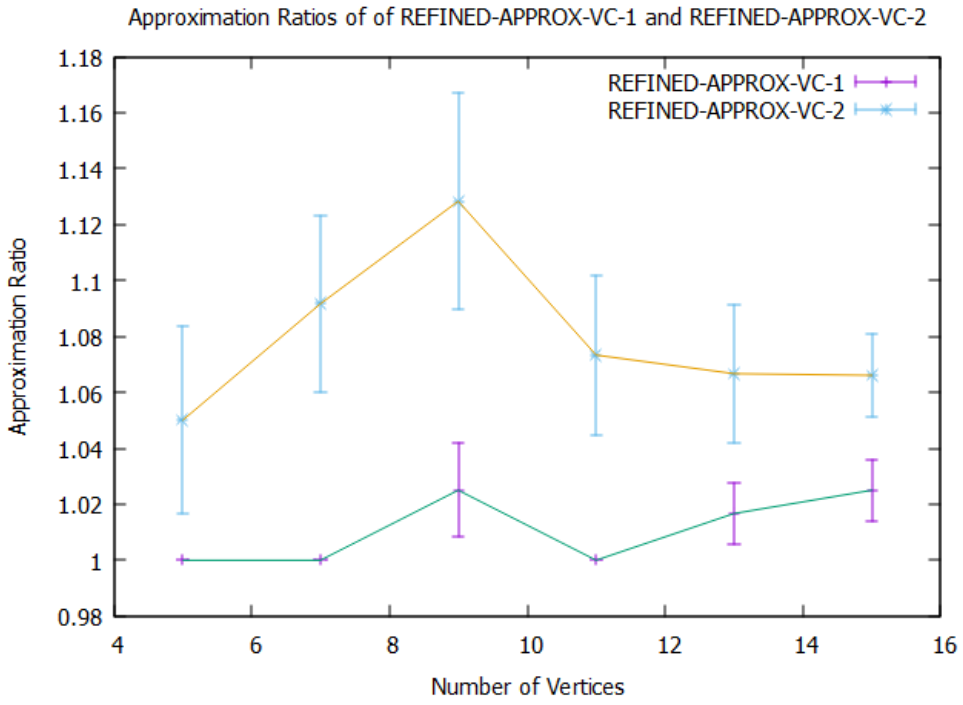
5.5 Approximation Ratio of APPROX-VC-1 and APPROX-VC-2



From the above figure, we can see that the approximation ratio of APPROX-VC-1 is close to 1, which is the best that we can get. This indicates that the APPROX-VC-1 algorithm gives a solution almost as optimal as the one given by CNF-SAT. The vertex cover provided by APPROX-VC-2 is almost twice the size of an optimal solution, even though it is faster than APPROX-VC-1. Thus, APPROX-VC-1 will provide more optimal solutions to vertex cover than APPROX-VC-2 but performs poorly. Therefore, APPROX-VC-1 provides a trade-off between solution quality and computational efficiency, by providing a near-optimal solution in a reasonable amount of time.

5.6 Approximation Ratio of REFINED-APPROX-VC-1 and REFINED-APPROX-VC-2

We have plotted the mean and standard deviation of the result of the refinement algorithms in the figure provided below. This graph indicates the accuracy of each of the REFINED-APPROX-VC1 and REFINED-APPROX-VC1 algorithms. Similar to the accuracy of the approximation algorithms, the REFINED-APPROX-VC1 and REFINED-APPROX-VC-2 algorithms also provide near-to-optimal vertex covers for a graph but not in a very efficient way. The accuracy of REFINED-APPROX-VC-1 is higher than the accuracy of REFINED-APPROX-VC-2 as the approximation ratio for the former is close to 1 on average. The latter has a higher value, indicating that the solution contains more vertices than the optimal solution of CNF-SAT.



6 Conclusion

In conclusion, we were able to implement and analyze six given algorithms to solve the vertex cover of a graph, with some of them providing optimal solutions. As part of the runtime analysis, we noticed that CNF-SAT took the longest amount of time to provide a solution, whereas reducing the CNF clause to a 3-CNF clause improved the performance significantly. Compared to the second approximation algorithm, the first approximation algorithm proved more accurate, as indicated by the approximation ratio. However, APPROX-VC-1 is slower than APPROX-VC-2. However, both approximation algorithms do not provide a perfect or minimal vertex cover for most graphs. On the other hand, refined approximation algorithms went closer to computing a minimal vertex cover than the general approximation algorithms, but even they may not be optimal solutions. The time taken by refinement algorithms is more than approximation algorithms as they work on the output of them, but the overall behavior, in terms of accuracy and running time, is similar to approximation algorithms. Accuracy is higher in comparison.