UNIVERSITY OF WATERLOO

Department of Electrical and Computer Engineering

ECE650 Methods and Tools for Software Engineering

Instructor: Rezza Babaaee

Run Time Analysis for Minimum Vertex Cover Algorithms

Team Name: a27kumar-s3yadav

Sushilkumar Yadav(21050889)

Ankush Kumar(21014893)

Abstract

The Minimum Vertex Cover (MVC) problem is a well-known combinatorial optimization problem with numerous applications in various domains. In this project, we analyed the runtime performance and approximation ratios of CNF SAT, 3CNF SAT, ApproxVC1, ApproxVC2, refined ApproxVC1, and refined ApproxVC2 algorithms. These algorithms are used to find the minimum vertex cover of a graph, which is finding the vertices that would encorporate all the edges of graph in it.

Our first algorithm is CNF SAT algorithm which is reducing the minimum vertex cover problem to the satisfiability problem in Conjunctive Normal Form(CNF) so that the MVC could be solved using a SAT solver. We then implemented the 3CNF SAT algorithm which is the optimized version of the CNF that restrics the clauses to have atmost three literals.

We also worked on the approximation algorithms named Approx VC1 and Approx VC2 which provides the approximate solutions to the minimum vertex cover problem.

We did further optimization on approximation algorithms. The optimized algorithms are known as Refined Approx VC1 and Refined Approx VC2.

In this project, we captured the Runtime, Standard Deviation and Approximation Ratios of all the six algorithms and compared the results in detail.

Contents

1 Introduction

The vertex cover problem is an NP-complete problem in computer science. It involves finding the smallest set of vertices that covers all edges in a given graph. Here are the algorithms that we have used to solve the vertex cover problem.

1.1 CNF SAT

To solve the minimum vertex cover problem using a CNF-SAT solver, we can convert the problem into a CNF formula and feed that to a SAT solver which can then return us whether given formula is SAT or not.

Following are the reduction steps that can be used to convert the problem to a CNF formula

1. At least one vertex must be included in the vertex cover: This means that for each vertex in the graph, either it is included in the vertex cover or not.

2. No vertex can be included in the vertex cover more than once: This means that if a vertex is included in the vertex cover, it cannot be included again.

3. Every edge must be connected to at least one vertex in the vertex cover: This signifies each edge in the graph has at least one of its endpoints connected to a vertex in the minimal vertex cover.

By representing these conditions as formulas in CNF, a CNF-SAT solver can be used to find a satisfiable assignment for the formula. If a solution is found, we can query the model values of the literals in our formula and compute the minimum vertex cover based on that.

1.2 3CNF SAT

In 3-CNF-SAT we use the same encoding as CNF-SAT and we make sure every clause has at maximum 3 literals. We have used the following:

$$(\mathcal{L}_0 \vee b_0) \wedge (\neg b_0 \vee \mathcal{L}_1 \vee b_1) \wedge \ldots \wedge (\neg b_{n-1} \vee \mathcal{L}_n)$$

We have applied the above reduction for reduction step 1 and 3 in the above reduction.

1.3 Approx VC1

```
Algorithm:
while edge set is not empty do
    Step 1: Pick a vertex with higher in-degree;
    Step 2: Remove all edges directly connected to vertex in Step 1;
end
```

1.4 Refined Approx VC1

```
Algorithm:
Data: Graph $G$
Result: Valid vertex cover of $G$
while vertex cover is valid do
    Step 1: Greedily pick a vertex;
    Step 2: Check if removing the vertex in Step 1 makes vertex cover invalid:
    if removing the vertex makes vertex cover invalid then
        Do not remove the vertex;
    end
    else
        Remove the vertex;
    end
end
```

1.5 Approx VC2

```
Algorithm:
Data: Graph $G$
Result: Vertex cover of $G$
while edge set is not empty do
    Step 1: Pick an edge $\langle u, v \rangle$;
    Step 2: Add $u$ and $v$ to vertex cover;
    Step 3: Remove all edges directly connected to $u$ and $v$;
end
```

1.6 Refined Approx VC2

```
Algorithm:
Data: Graph $G$
Result: Vertex cover of $G$
while vertex cover is valid do
    Step 1: Greedily pick a vertex;
    Step 2: Check if removing the vertex in Step 1 makes vertex cover invalid:
    if removing vertex in Step 1 makes vertex cover invalid then
        Do not remove the vertex in Step 1;
    end
    else
        Remove the vertex in Step 1;
    end
end
```

2 Project implementation

2.1 Multithreading

- Threads:
  - In total, we have created nine threads.
  - Seven threads to handle input-output and all the algorithms to calculate minimum vertex cover.
  - In the implementation, we created a specific thread to handle input-output.
  - That thread further creates 6 threads to defer all calculations for 6 methods used for calculating MVC. Using this model it is made sure that the input-output never gets blocked and the program always accepts input even if it is currently busy executing the current input.
- Two threads to handle the timeout so that the program is not blocked by CNF or 3 CNF threads as these methods can take exponentially larger time as input size grows.
  - We have created two threads to make sure we timeout CNF and 3-CNF so that our program is not blocked for an infeasible time. We used 300000 milliseconds as the threshold timeout value.

2.2 Synchronization

- Since both refined(Refined VC1 and Refined VC2) algorithms have a dependency on the output from approximate(Approx VC1 and Approx VC2) algorithms.
- We have made sure to synchronize both threads using mutex and conditional variables. So that the refined algorithm's thread handler is always executed after the approximate algorith thread handler is done.
- We have used conditional variables to wait for signals from other dependent threads and maintained non-blocking behaviour.
- The conditional variables keep on checking whether the flag is (true or not) or if the timeout has occurred. Without busy waiting hence zero performance degradation.
- Following are the scenarios that can happen:
  1. SAT solver calculates whether the CNF formula given as input is SAT. Then the CNF algorithm thread sets the flag as true and notifies threads listening on it's conditional variable. The thread waiting on the same conditional variable wakes up and does it's course of work.
  2. SAT solver takes a long time to calculate whether the CNF formula is SAT and timeout reaches the threshold value then the waiting thread interrupts the SAT solver and sets the flag as true. The CNF worker thread checks that flag and since it is true it knows that timeout has reached and the thread exits gracefully.

3 Analysis

3.1 CNFSAT Vs 3CNFSAT runtime analysis

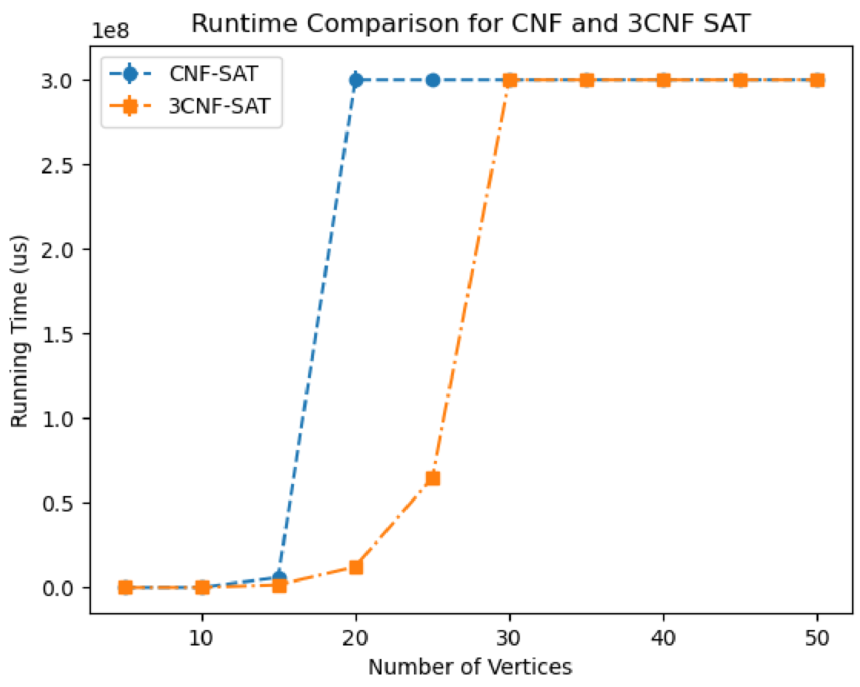
For CNF SAT: For smaller inputs (vertex values from 5 to 15), the runtime is reasonable, ranging from 1325.74 µs to 5967619.34 µs.

From vertex 20 onwards, we observed timeout issues with the CNF_SAT algorithm. This was because the SAT solver used in the algorithm took longer to execute, and our code had a defined timeout of 5 minutes to calculate the minimum vertex cover. The SAT solver did not provide a satisfiable assignment within the timeout period, causing our threads to gracefully exit and resulting in a timeout for the algorithm. This indicates that the performance of the CNF SAT algorithm may degrade significantly for larger inputs, and the timeout limit may need to be adjusted more efficiently.

For 3 CNF SAT: The runtime is generally higher than CNF SAT for smaller inputs (vertex values from 5 to 10), ranging from 1856.48 µs to 31165.25 µs whereas in CNF SAT the time for Vertex values from 5 to 10 was ranging from 1325.74 µs to 29433.45 µs.

The advantage of Three CNF SAT is that as the input size grows, it performs reasonably well as compared to CNF SAT. Also CNF SAT was getting timed out after vertex 15 whereas 3CNF SAT was getting timed out after vertex 27. This shows that 3CNF SAT performs reasonably well for the larger inputs.

Based on the above analysis, it appears that both CNF SAT and Three CNF SAT algorithms have reasonable runtimes for smaller inputs (vertex values up to 15). However, they are exhibiting performance issues for larger inputs (vertex values from 15 to 50 in CNFSAT and from 28 to 50 in 3CNFSAT), with runtimes reaching up to 300003332.5 µs(Timeout threshold defined in the code), which could indicate scalability limitations.



(a) CNFSAT Vs 3CNFSAT runtime analysis

3.2 VC1 Vs Refined VC1 runtime analysis

Approx VC1:

Runtime:

The runtime of Approx VC1 varies from 45.3 to 207.52 µs, with an average runtime of around 100 µs.

Standard Deviation:

The standard deviation of Approx VC1 varies from 7.28 to 797.80 µs, with an average standard deviation of around 240 µs. Overall depending on the input, standard deviation is fluctuating and we couldn't find any well defined pattern as input size increases.

Refined Approx VC1:

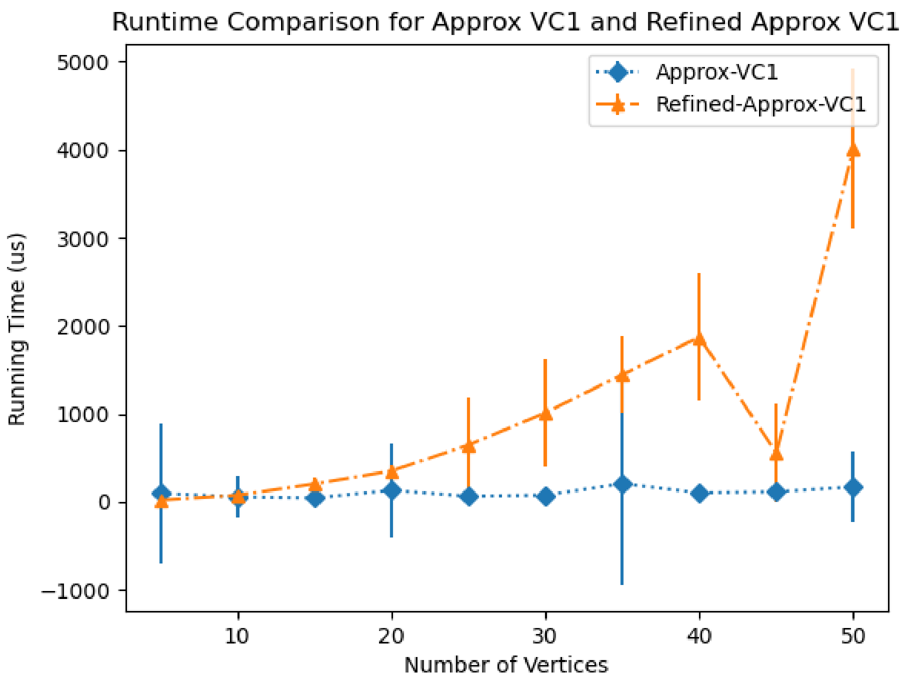
Runtime:

The runtime of Refined Approx VC1 varies from 20.86 to 4007.07 µs, with an average runtime of around 1000 µs. The runtime is generally higher compared to Approx VC1, specifically for large vertex size.

Standard Deviation:

The standard deviation of Refined Approx VC1 varies from 6.51 to 1204.79 µs, with an average of around 230 µs. We observed that there is a steady increase in the standard deviation as the vertex size grows indicating the higher variability in the algorithm's runtime.

Overall, Refined Approx VC1 generally having higher runtimes and standard deviations compared to Approx VC1. The increase in runtime for the Refined Approx VC1 algorithm is due to the repeated verification step it performs after removing each vertex to ensure that the vertex cover still holds. This additional step adds to the overall runtime of the algorithm as it needs to be performed after every vertex removal, resulting in increased run time.



(a) VC1 Vs Refined VC1 runtime analysis

3.3 VC2 Vs Refined VC2 runtime analysis

Approx VC2:

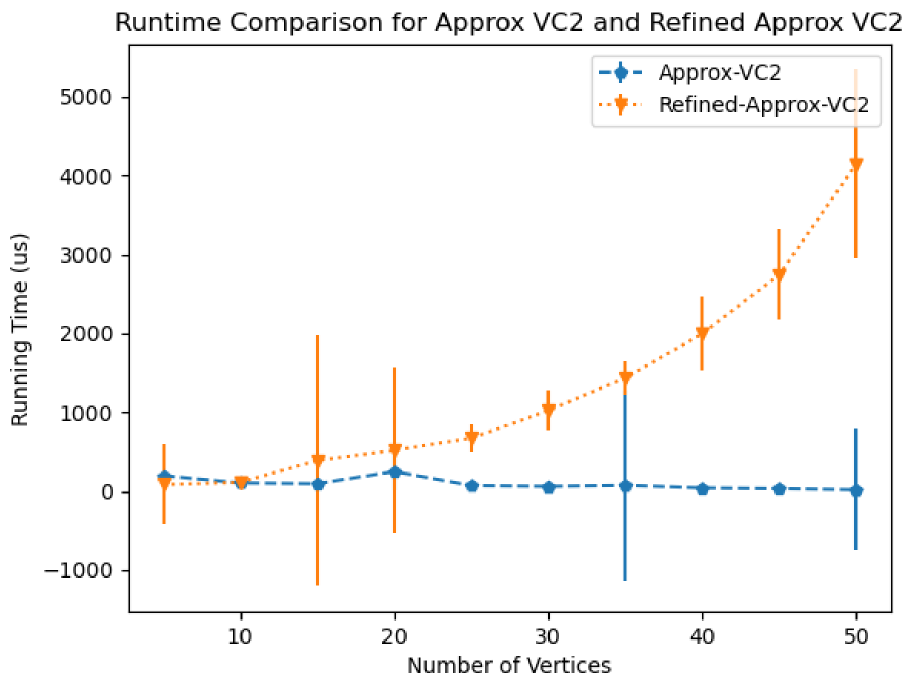
Based on the runtime values, we can see that the runtime increases as the number of vertices increases, which indicates that the algorithm's runtime is dependent on the size of the input (number of vertices). Looking at the standard deviation values, we can say that the algorithm's runtime is relatively stable for different input sizes. From the graph, we can also see that The run time value is decreasing from 195.06 to 16.13 (i.e., the values decrease as the number of vertices increase).

Refined Approx VC2:

Similar to Refined Approx VC1, the runtime for Refined Approx VC2 also shows that the algorithm's runtime is highly correlated with the input size. unlike the standard deviation in Approx VC2, Refined Approx VC2 has higher standard deviation which shows that the run time changes significantly for same vertex sizes but for different edge configurations.

Comparing the runtime for both the algorithms we can see that the runtime for refined VC2 is more than that of a approx VC2 algorithm. this is because Refined VC2 algorithm repeates the verification step after removing each vertex to ensure that the vertex cover still holds. This additional step adds to the overall runtime of the algorithm as it needs to be performed after every vertex removal, resulting in increased run time.

The standard deviation for Approx VC2 is pretty stable as the input size grows. However, in the case of refined Approx VC2, the standard deviation keeps on increasing as the input size grows.



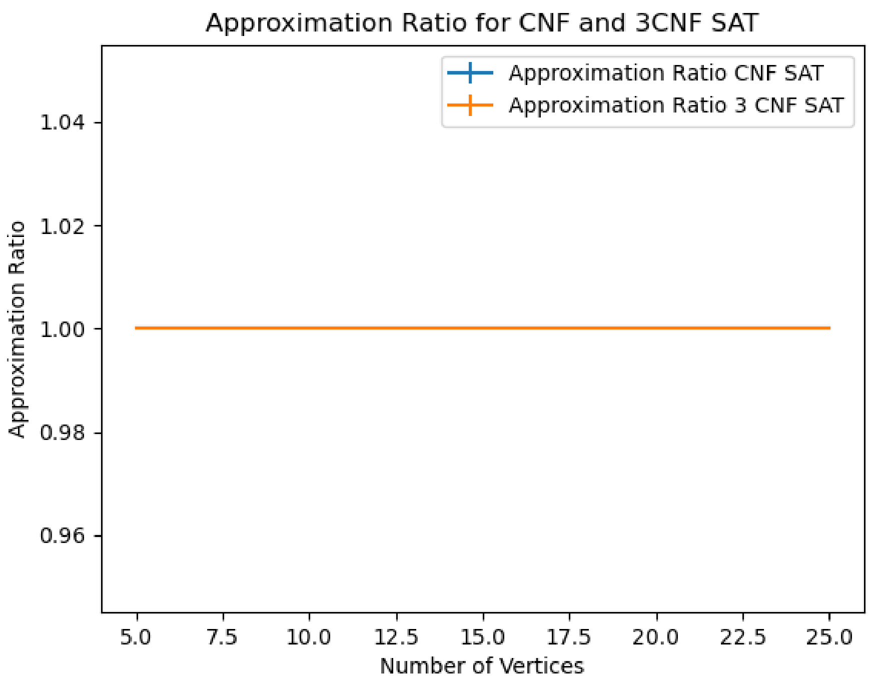
(a) VC2 Vs Refined VC2 runtime analysis

3.4 Approximation Ratio CNF and 3 CNF

The standard deviation for Approx VC2 is pretty stable as the input size grows. However, in the case of refined Approx VC2, the standard deviation keeps on increasing as the input size grows.

The below graph suggests that the approximation algorithms used for CNF SAT and 3CNF SAT are providing consistent and optimal solutions across the range of vertices considered. The value of 1.0 indicates that the approximate solution for the minimum vertex problem using CNF SAT and 3CNF SAT is equal to the optimal solution.

CNF and 3CNF SAT gives us optimal solution because SAT solvers work on finding an satisfiable assignment for the given CNF formula. As long as the reduction of the problem is done correctly into CNF, model values returned by SAT solvers always gives an optimal solution. However because vertex cover problem is NP complete, finding a satisfiable assignment is computationally expensive and not feasbible for large input sizes.



(a) Approximation Ratio for CNF and 3CNF

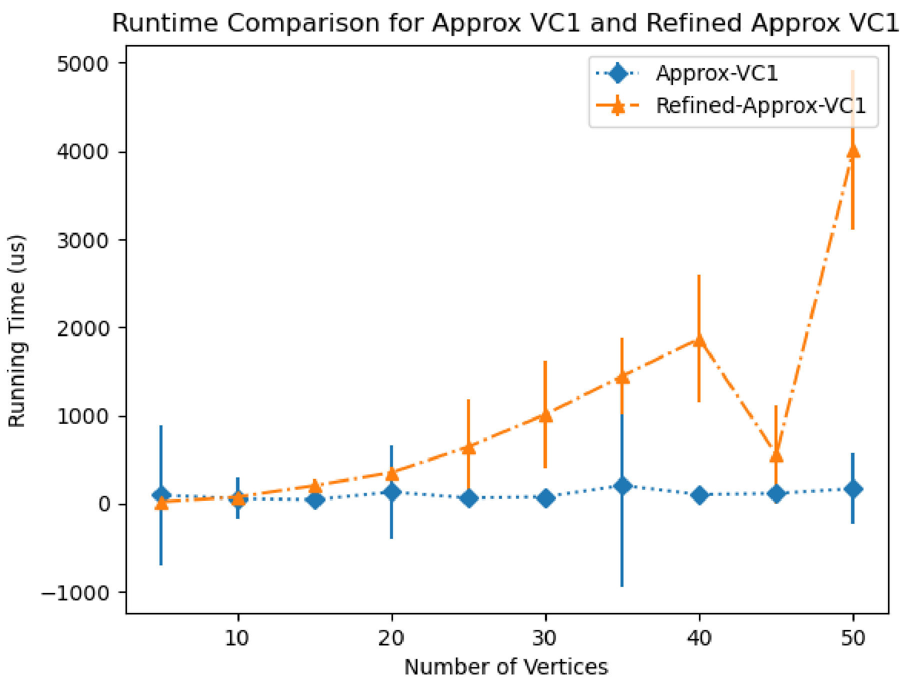
3.5 Approximation Ratio VC1 and Refined VC1

Approx VC1:

After analysing the approximation ratio graph for Approx VC1 and Refined Approx VC1, we can see that approx VC1 approximation ratio is greater that 1 for all the vertices which means that Approx VC1 algorithm provides the the solutions that are larger than the optimal solutions. also we can see that as the input size increases(number of vertex increases), the approximate values also keeps on increasing.

Refined Approx VC1:

incase of Refined Approx VC1, the approximation ratios are very close to 1 for low vertex values which means that the algorithm is able to provide near optimal solution for lower vertices but increases slightly as the vertex increases. However, the increase is minimal and is very low as compared to the Approx VC1.



(a) VC1 Vs Refined VC1 runtime analysis

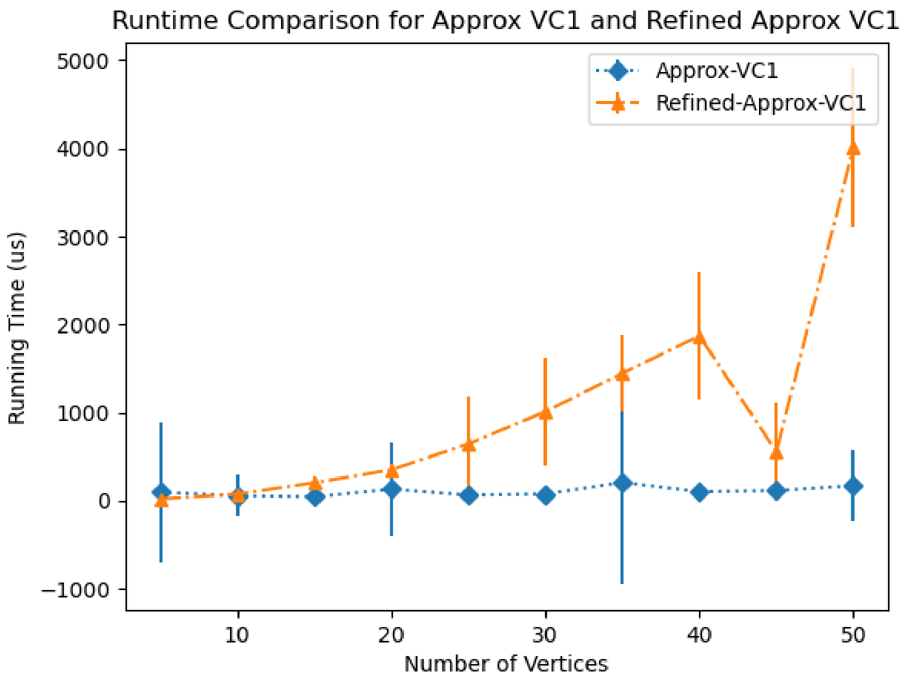
3.6 Approximation Ratio VC2 and Refined VC2

Approx VC2:

From the graph we can see that the approximation ratio value is decreasing as the number of vertex increases. from this, we can also say that the the algorithm is getting to near optimal solution as the vertex increases. however, if we compare the approximation ratio of Approx VC2 with the Refined Approx VC2, we can see that the refined VC2 algorithms is already very close to the optimal solution.

Refined Approx VC2:

From the graph we can see that approximation ration remains quite stable with the slight increase as the number of vertex increases. by comparing the values of both the graphs, we can say that the Refined Approx VC2 is providing better results as compared to the Approx VC2.



(a) VC1 Vs Refined VC1 runtime analysis

4 Conclusion

Runtime analysis

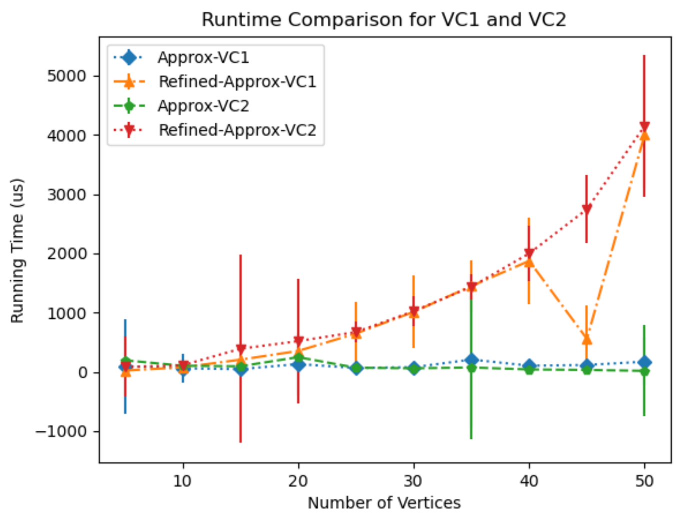
CNF SAT and 3CNF SAT show an exponential increase with increase in the vertex size of graph from 5 to 50. The runtime for CNF SAT ranges from 1325.74 to 300003332.5, and for Three CNF SAT it ranges from 1856.48 to 300003332.5. The standard deviations for these runtimes are relatively small and remain constant at 0 after a certain point, indicating consistent behaviour and performance.

Approx VC1 and Refined Approx VC1 runtimes show a fluctuating pattern. For Approx VC1, the runtime initially decreases from 97.21 to 45.30, and then increases to 171.48. However, the standard deviation is relatively small for lower vertex ranges (5 to 15) but increases significantly for higher vertex ranges (20 to 50), indicating less consistent behaviour because of the design of the algorithm. For Refined Approx VC1, the runtime increases significantly from 20.86 to 4007.07, with a large standard deviation for the vertex range 50.

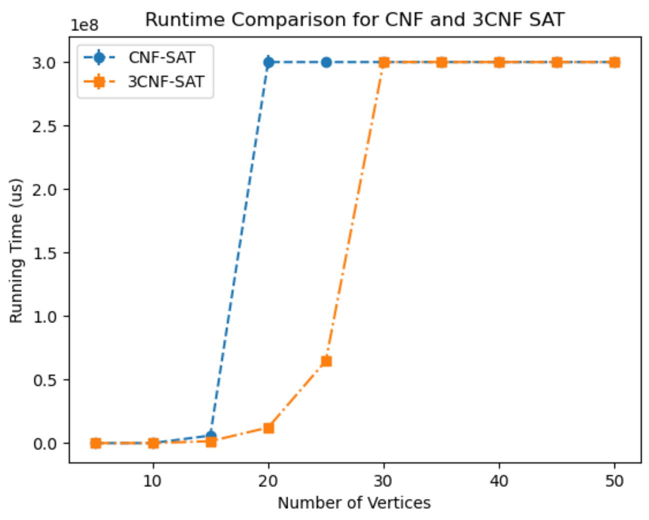
Approx VC2 and Refined Approx VC2 runtimes show a decreasing pattern with increasing vertex size from 5 to 50. For Approx VC2, the runtime decreases from 195.06 to 16.13, and for Refined Approx VC2, it decreases from 81.91 to 4150.86. The standard deviations for these runtimes are relatively small for lower vertex ranges (5 to 25) but increase significantly for higher vertex ranges (30 to 50).

The standard deviations for CNF SAT and 3CNF SAT are mostly constant at 0 after a certain point, indicating uniforrmity in performance. Whereas, the standard deviations for Approx VC1, Approx VC2, and Refined Approx VC2 increase significantly for higher vertex sizes, showing and more dependency over the input configurations due to the algorithm design. The standard deviation for Refined Approx VC1 remains relatively constant, but it increases for vertex range 50, showing slightly fluctuating performance.

In summary, CNF SAT and 3CNF SAT algorithms show consistent performance with increasing runtimes, while Approx VC1, Refined Approx VC1, Approx VC2, and Refined Approx VC2 algorithms show varying performance with fluctuating runtimes and standard deviations.



(a) VC1 and VC2 runtime analysis



(b) CNF SAT and 3CNF SAT runtime analysis

Figure 7: Runtime analysis of VC1, VC2, CNF SAT, and 3CNF SAT

Approximation Ratio Analysis

Approximate Ratio for CNF SAT: The approximate ratio remains constant at 1.0 for all vertex values (5, 10, 15), indicating that the CNF SAT algorithm always finds an optimal solution for the minimum vertex cover problem. However, for vertex greater than 15 we are getting timed out as we have kept the maximum wait time of 5 minutes in our code.

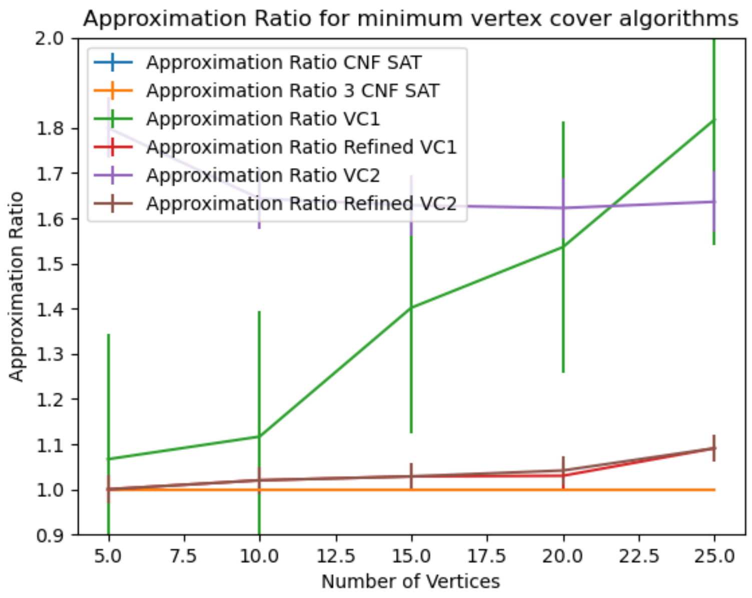
Approximate Ratio for 3CNF SAT: Equivalent to Approximate Ratio for CNF SAT, the approximate ratio remains equal at 1.0 for Vertex 5 to 25, indicating that the CNF 3-SAT algorithm also always finds a minimum vertex cover for the given graph. However, for vertex greater than 25 we are getting timed out as we have kept the maximum wait time of 5 minutes in our code.

Approximate Ratio for Approx VC1: The approximate ratio increases incrementally with increasing vertex sizes of graph, ranging from 1.06667 for 5 vertices to 1.818182 for 25 vertices. This shows that approx VC1 algorithm calculates graph that are less optimal and larger in size compared to optimal minimal vertex cover as graph size increases.

Approximate Ratio for Refined Approx VC1: The approximate ratio remains closer to 1.0 for all vertex values, with a minute increase ranging from 1.0 for 5 vertices to 1.090909 for 25 vertices. This suggests that the Refined Approx VC 1 algorithm helps refining the calculated vertex cover from Approx VC1 by eliminating any vertices that is not needed.

Approximate Ratio for Approx VC2: The approximate ratio shows some variation, ranging from 1.622593 for 20 vertices to 1.8 for 5 vertices. This suggests that the Approx VC 2 algorithm may not always find a minimum vertex cover, and the quality of the vertex cover may vary depending on the graph size and the way the input is configured. This is due to the random behaviour entailed with our algorithm design.

Approximate Ratio for Refined Approx VC2: Similar to Approximate Ratio for Refined Approx VC1, the approximate ratio remains close to 1.0 for all vertex values, with a slight increase ranging from 1.0 for 5 vertices to 1.090909 for 25 vertices. This suggests that the Refined Approx VC 2 algorithm refines minimum vertex cover close to the optimal solution.



(a) Approximation Ration of six algorithms