UNIVERSITY OF WATERLOO

ECE 650

Final Course Project

Report Analysis

Submitted By:

Rashandeep Singh (21015920) rsrashandeepsingh@uwaterloo.ca
Dilpreet Singh (21049904) d29singh@uwaterloo.ca

Winter 2023

Contents

1 Objective

The objective of the given assignment is to solve the vertex cover problem with 6 different Algorithms. Further, we have to concurrently implement these algorithms using multi-threading. As a result of which we will be computing and analyzing the run time of all the algorithms. Also, it is for sure that one of the six algorithms will provide the most optimal solution i.e. the minimum vertex cover. So, based on this we have to also compute and analyze the approximation ratio of all the algorithms.

1.1 What is vertex cover?

Given a graph i.e. no. of vertices v and edges (u,v) between them, the vertex cover provides the set of vertices such that every u or v of an edge is part of it.
This problem is considered an NP-hard problem therefore it cannot be solved by a polynomial time [2]. We convert the problem to propositional logic formulas and we can use those formulas in a boolean SAT solver which gives us the result of the formula if it is satisfiable. Therefore, using the resultant values we get the solution to the vertex cover problem and it is sure that the answer is a minimum vertex cover.
The propositional logic used by the SAT solver is Conjunctive Normal Form [5].

1.2 What is CNF?

The conjunctive Normal form is the conjunction of clauses where each clause is a disjunction of literals [4]. A literal is defined as an atomic proposition or negation of an atomic proposition.
For example,

$$(AVB)\Lambda(CVD) \tag{1}$$

is in CNF whereas

$$(A\Lambda B)V(C) \tag{2}$$

is not in CNF
Further, a CNF can also be defined as 2-CNF and 3-CNF. Every formula can be converted into 3-CNF but it is not sure whether it can be converted into 2-CNF or not [1].
For a formula to be in 3-CNF, the formula should be in CNF and each clause will have at most 3 literals [3].
For example,

$$(AVBVC)\Lambda(CVD) \tag{3}$$

is in 3-CNF whereas

$$(AVBVCVD)\Lambda(EVF) \tag{4}$$

is not in 3-CNF. It is because the first clause contains more than 3 literals.

2 Algorithms

We have implemented the vertex cover problem using 6 algorithms such as

1. CNF-SAT-VC

2. CNF-3-SAT-VC
3. APPROX-VC-1
4. APPROX-VC-2
5. REFINED-APPROX-VC-1
6. REFINED-APPROX-VC-2

We used an adjacency matrix data structure of size [vertex x vertex] to store all the edges. For example, if there is an edge (1,2). We store matrix[1][2] = matrix [2][1] = 1 and everything else is set to 0 in the matrix. This process is repeated for all the given edges. Further for storing results we have used a vector data structure. Finally, the result will be sorted in ascending order.

2.1 CNF-SAT-VC

In this algorithm, the vertex cover problem was reduced to CNF-SAT. The formula F was constructed using four conditions :

1. At least one vertex should be there in the vertex cover.
2. Secondly, the vertex will appear only once in the vertex cover.
3. Thirdly, only one vertex can be present for the same position in the vertex cover.
4. Lastly, for every edge(u,v) either u or v should be present in the vertex cover.

These four conditions were coded and were solved using minisat which in return will provide the values for which the formula was satisfiable.

2.2 CNF-3-SAT-VC

The first and fourth conditions of the above algorithm contained more than 3 literals in the clause. Therefore, the clauses were broken down into smaller clauses using dummy literals which were then computed using minisat.

2.3 APPROX-VC-1

In this algorithm,

1. Pick the vertex with the highest no. edges around. (The highest degree was checked by counting the no. of ones in each matrix row.)
2. Add the vertex to the vertex cover.
3. The edges incident to that vertex were discarded (the values in the matrix were set to 0)
4. Pick the second highest degree.
5. Repeat till no edge is remaining.

2.4 APPROX-VC-2

In this algorithm,

1. Pick the first edge
2. Add its u, v in the vertex cover
3. Discard all the edges incident to both u and v
4. Repeat the above steps till no edges remain.

2.5 REFINED-APPROX-VC-1 and REFINED-APPROX-VC-2

We solved these algorithms using 2 approaches -

1. Recursion - In the recursive algorithm, we use the output of approximate algorithms and then we compute if we can drop any vertex from that solution.
   Algorithm:
   (a) Start iterating from index 0 till the length of a vertex cover of approx algorithm
   (b) Initialise an empty vertex cover that will be our solution for a recursive algorithm
   (c) While iterating each vertex we have two options
      i. Either we can add the current vertex to our optimized vertex cover and move to the next vertex.
      ii. Or, we can discard this vertex and move on to the next vertex.
   (d) when we reach the end, we need to verify if the computed optimized vertex cover is still our vertex cover i.e. it covers all the edges.
   (e) Take the minimum size of all the vertex covers that satisfy step 4.
2. Greedy
   Algorithm:
   (a) Pick the highest degree vertex let's say "U" from the vertex cover returned by APPROX-VC-1 (say this as Q).
   (b) Add that vertex in the vertex cover and remove all its adjacent vertices from the adjacency matrix.
   (c) Now consider an arbitrary vertex "V" that has an edge with vertex "U", now if "V" is also part of Q then we need to make a call if we can remove "V" from Q
   (d) Check if all the vertices adjacent to "V" are present in Q then we can remove "V" from Q else even if any one of them is absent then simply continue.
   (e) Again calculate the highest degree vertex.
   (f) Repeat it till there is no vertex left in Q.

For the above six algorithms, it is for sure that CNF-SAT-VC and CNF-3-SAT-VC will give the most optimal solution for vertex cover.

So, the six algorithms we implemented concurrently with the use of threads. One thread was assigned to input and one to each algorithm. The main thread was asked to wait for the threads to finish before giving the final result. Further, the time taken by each thread was computed using pthreadgetcpuclockid(). We were asked to generate at least 10 graphs for each value of V (V being

vertex from [5, 50]). Further, we ran 10 runs of each graph resulting in 100 runs for each value of V. This process was automated, and mean and standard deviation for time and approximation ratio (ratio of the size of computed vertex over the size of most optimal vertex cover) were calculated. The graphs are shown in the following section.

3 Graphs

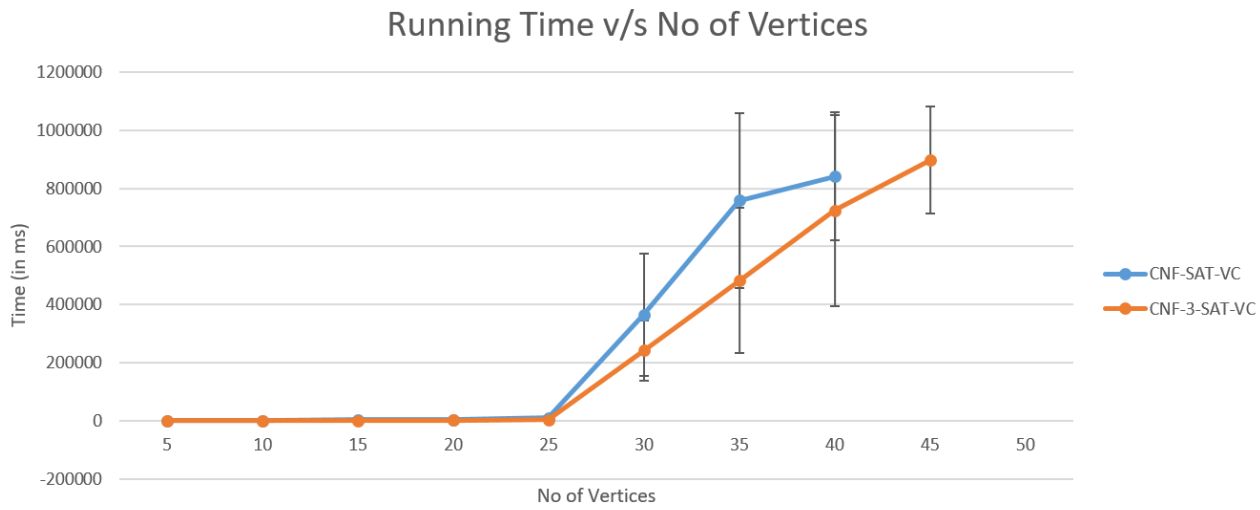

Figure 1: CNF v/s 3 CNF Time

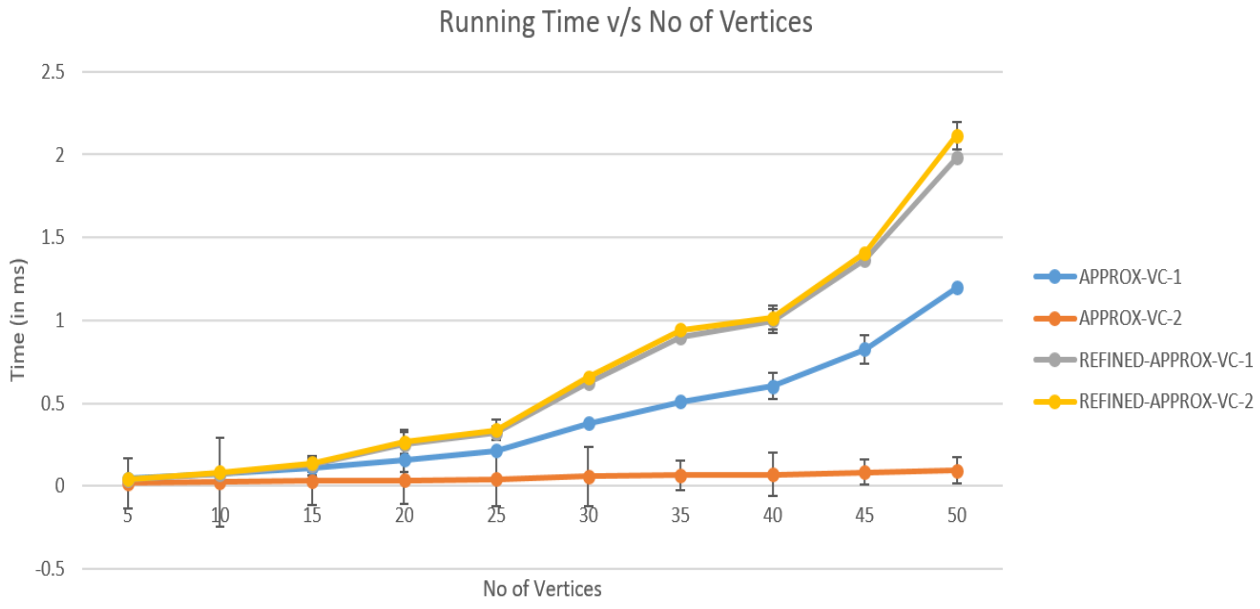

Figure 2: Other 4 Algorithms Time

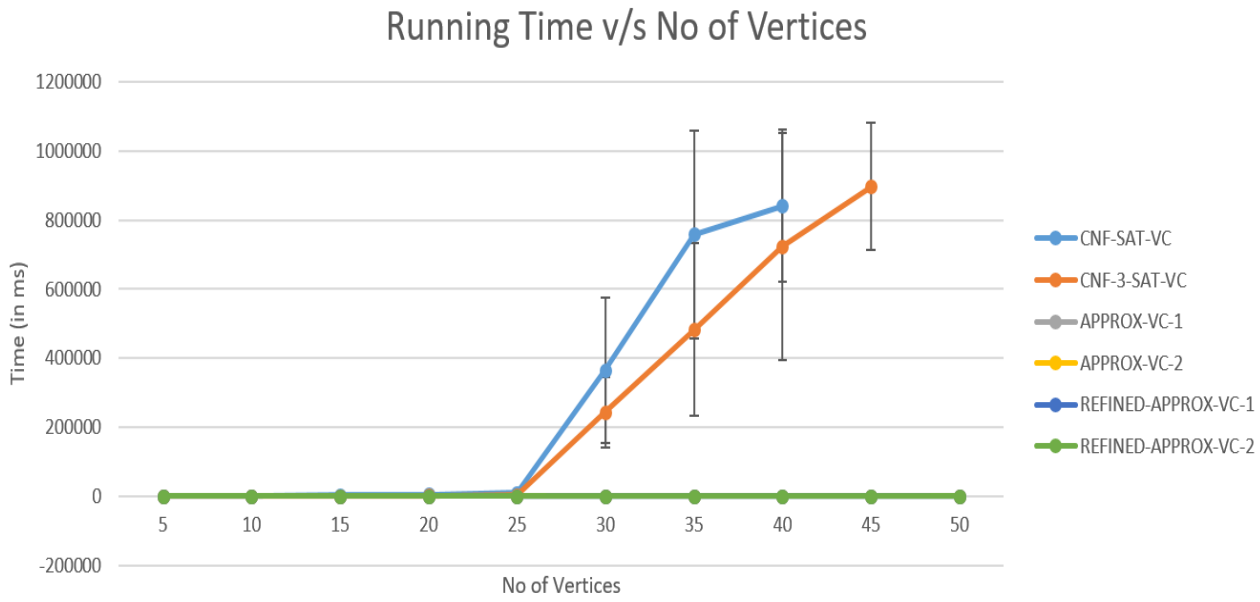

Figure 3: All 6 Algorithms Time

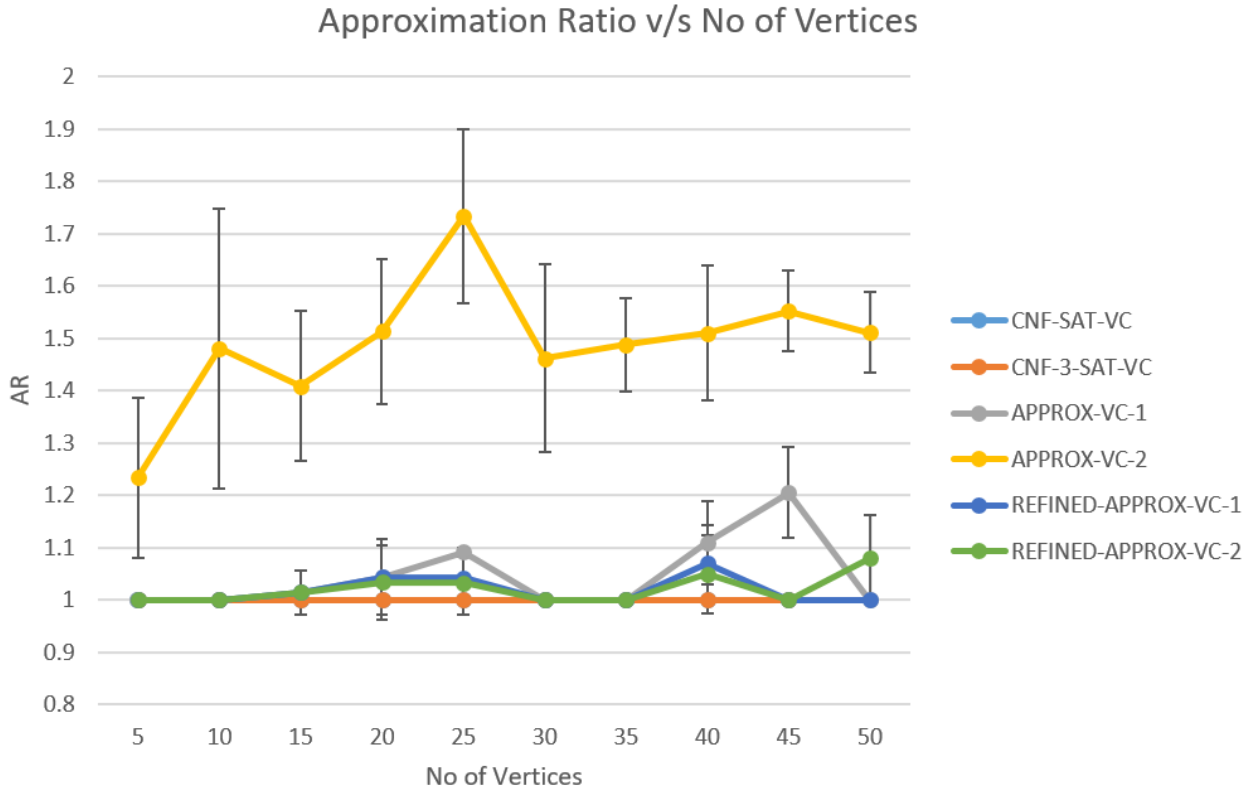

Figure 4: Approximation Ratio of all 6 Algorithms

4 Data values of Mean

Table 1: Mean values of time in ms for V = [5,50]

No of Vertices	CNF-SAT-VC	CNF-3-SAT-VC	APPROX-VC-1	APPROX-VC-2	REFINED-APPROX-VC-1	REFINED-APPROX-VC-2
5	0.28944	0.53414	0.04675	0.01795	0.03782	0.04233
10	15.1764	16.0633	0.07084	0.0248	0.07113	0.08405
15	2418.78	448.926	0.109	0.0302	0.1267	0.1387
20	4508.45	1509.23	0.15852	0.03443	0.25303	0.26593
25	9309.12	3207.32	0.21253	0.0427	0.31883	0.33843
30	365234	242432	0.37727	0.05755	0.62221	0.65568
35	758142	482621	0.50796	0.06373	0.89871	0.94151
40	841323	723451	0.60161	0.07041	0.99762	1.01629
45		897322	0.82548	0.0841	1.36417	1.40639
50			1.199	0.095	1.982	2.116

Table 2: Mean values of Approximation ratio for V = [5,50]

No of Vertices	CNF-SAT-VC	CNF-3-SAT-VC	APPROX-VC-1	APPROX-VC-2	REFINED-APPROX-VC-1	REFINED-APPROX-VC-2
5	1	1	1	1.2333	1	1
10	1	1	1	1.48	1	1
15	1	1	1.01429	1.40833	1.01429	1.01429
20	1	1	1.04333	1.51333	1.04333	1.03333
25	1	1	1.09167	1.7333	1.04167	1.03167 3
30	1	1	1	1.46144	1	1
35	1	1	1	1.48748	1	1
40	1	1	1.108992	1.50975	1.06929	1.04892
45		1	1.20482	1.552	1	1
50			1	1.5108	1	1.07923

5 Analysis

5.1 Analysis of CNF and 3 CNF

As seen in Figure 1 and Table 1 it is observed that for 5 and 10 vertices, the CNF was returning the vertex cover in a shorter time than 3-CNF. As we know, due to smaller vertices number of clauses in CNF is less. In 3 CNF more and more literals are added to make the clauses even smaller(at most 3 literals in a clause). As a result, in 3 CNF we added more literals and more clauses were added to make the size of each clause smaller whereas no more clauses and literals were added in the CNF algorithm. Hence, In order to satisfy the vertex cover problem the SAT solver has to compute the truth values for added-on literals as well. In a nutshell, 3 CNF is computing more literals and solving more clauses as compared to 3CNF for smaller vertices which costs more computation time for 3CNF. For a large number of vertices, each clause of CNF will have a large number of literals, and SAT solver will have to compute every combination of literals to check if it is satisfiable or not. For Example, for Vertices = 30 and No of Edges = 20 and now consider a clause that covers all edges so the size of this clause is 60 and such clauses are 20. Now the SAT solver has to compute this clause using a truth table of 60 literals which consumes a lot of time. On the other hand, 3 CNF will have at most 3 literals in each clause and if any of the clauses which are computed first comes out to be false the whole formula will be unsatisfied as we have a conjunction between each clause and the algorithm will compute for the incremented size of vertex cover without computing other clauses. Hence, a difference in time computation of both algorithms comes from the factor that how quickly a clause is computed to false which avoids the computation of other clauses and moves on to the computation of the incremented size of vertex cover. Therefore, CNF will compute for the larger size of literals in each clause whereas 3 CNF will check for very smaller sizes and give the vertex cover more optimally.

Standard Deviation till V=25 is not visible because the mean computation time is really less. However, we can see the data deviated largely for vertices greater than or equal to 30. The reason for the same is that we computed the data for 10 different graphs and the size of the minimum vertex cover for each graph was different. As a result computation time for each iteration deviated a lot from each other resulting in a larger standard deviation.
Additionally, for vertex 40 around 71 iterations timed out (time out = 15 minutes, and this data was ignored) for CNF. For the remaining 29 iterations, the computation time came out closer to 14 minutes resulting in a smaller standard deviation. For vertex 45 and 50 our CNF was not able to compute the vertex cover.
For 3 CNF, we were able to compute the vertex cover till 45 vertices (out of 100 iterations only 23 were able to compute the answer). So, for these 23 iterations, the computation time was close to 15 minutes resulting in a smaller standard deviation. Also, for 3-CNF we were not able to find the minimum vertex cover within 15 minutes.

5.2 Analysis of other 4 Algorithms

For the other 4 Algorithms in Figure 2, it is observed that the time for each algorithm increases linearly with respect to the number of vertices as they are solved in polynomial time.
For every vertex, APPROX-VC-2 gives the vertex in the smallest time of all, it is because no additional computations are done in this algorithm. It simply picks an edge adds its vertices to the vertex cover and continues doing so till no edges remain.
Moreover, APPROX-VC-2-REFINED has the highest computation among all for all vertices. It works on the output of the APPROX-VC-2 algorithm and then removes the vertex which is not part of the minimum vertex cover. It chooses the vertex to be removed greedily depending upon the number of other vertices to which a vertex is connected.
If we compare APPROX-VC-2-REFINED and APPROX-VC-1-REFINED, as we can see in the graph the latter is more optimal. The reason for this is both of them work on the output of APPROX-VC-2 and APPROX-VC-1 respectively and the size of the vertex cover returned by APPROX-VC-2 is always larger than APPROX-VC-1 and the algorithm that internally works in both of them is same. Hence, the only difference comes from the size of the input due to which APPROX-VC-2-REFINED takes more time to compute.
Now if we compare APPROX-VC-1 and APPROX-VC-2, the latter takes less time because it arbitrarily adds a vertex to the vertex cover whereas APPROX-VC-1 computes the highest incident edges and then adds the vertex to the solution. Hence, no computation is involved in APPROX-VC-2 due to which it takes less time.
The standard deviation is largely visible for the APPROX-VC-2 algorithm because the computation time of the algorithm depends upon the input data as we are just choosing the vertices blindly, so if a vertex which is more connected to others is chosen first we get smaller computation time and if an isolated vertex is chosen first then more and more iterations are required to get the solution which increases the running time. Due to this variation in running time, the standard deviation is large.
However, the standard deviation for other algorithms is not much visible because all these algorithms don't much depends upon the input data, they simply follow the same flow to find the solution.

5.3 Analysis of Approximation Ratio

For the approximation ratio in Figure 4 and Table 2, it is clearly observed that the mean for CNF and 3 CNF is coming as one. It is because these algorithms for sure give the optimal solution for any number of vertices.
Additionally, APPROX-VC-1 picks the vertex which has the largest number of edges whereas APPROX-

VC-2 blindly picks edges and add its vertex to the vertex cover resulting in additional unnecessary vertices to the answer. Therefore, we can observe that the mean for the approximation ratio of APPROX-VC-1 is closer to 1 (more than 1) and sometimes 1 and is always less than that of APPROX-VC-2. Moreover, the approximation ratio of APPROX-VC-2 majorly depends upon the input data of 10 graphs due to which the size of APPROX-VC-2 vertex cover varies a lot. Therefore, the standard deviation for APPROX-VC-2 is high for every vertex.
For vertex size 50 we did not get any answer for vertex cover using CNF and 3 CNF as it got times out (15 mins). So we took REFINED-APPROX-VC-1 as the most optimal solution. So, the answer to it is 1 which clearly defines the downfall from 45 vertices to 50 vertices.
For every vertex, if the mean of APPROX-VC-1 and APPROX-VC-2 will come less, the mean of its refinements will also be less as it works on the output of the former and further optimizes it to bring the size of the vertex cover to a minimum.
The standard deviation for APPROX-VC-1 is small as the mean for it is always close or equal to 1, so its error bar is only slightly visible.
The standard deviation for the mean 1 comes out to be 0. So we can't see any error bars for the CNF and 3 CNF.

6 Optimization and Encoding

6.1 Encoding for CNF and 3 CNF algorithms

Earlier both CNF and 3CNF were only executed till vertices = 25 but then we optimized the encoding involved in CNF and 3 CNF algorithms. Optimizations involved:

1. There was a clause in which we say no vertex can be present at two different indices of vertex cover. Optimization to this is, if a vertex is not present in any of the edges of the given graph then we can discard this clause for this vertex as if a vertex is absent in the graph then it can't be part of vertex cover.

2. There was another clause that two different vertices can't be present at the same index of vertex cover. We can use logical reasoning to avoid clauses in this condition such as if we pick two vertices say u and v and we know either "u" or "v" is not present in given edges then we can avoid the clause of "u" and "v" to be present at the same index of vertex cover as we are sure that in minimum vertex cover, either of one which is missing in edges won't be the part or minimum vertex cover of the graph.

3. There was a clause in which we say that at each index of vertex cover, at least one vertex should be present, so here we can use the reasoning that if any vertex which is absent in the graph need not to be present at any index of vertex cover. Therefore, we can avoid many literals in each clause of this condition.

4. Another major optimization is that we used to iterate for the size or vertex cover starting from 1 till a number of vertices and whenever we got the answer for any size of vertex cover that is considered as the final answer. Optimization to these iterations is firstly we used binary search to iterate for the size of vertex cover using low =1 and high= number of vertices and further choose the interval based upon the fact that CNF was satisfied for vertex cover size = mid = (low+high)/2. After doing the data analysis it was observed that binary search improved the computation time for larger instances but further iterations were reduced using the following logic:
   We used the upper limit of vertex cover size as the size of the vertex cover returned by the

REFINED-APPROX-VC-1 algorithm as we know that the size returned by REFINED-APPROX-VC-1 will return near to optimal solution which can also be depicted from the approximation ratio as approximation ratio of REFINED-APPROX-VC-1 is close to 1 always. So we did the back iteration for the size of the vertex cover in CNF and 3CNF algorithm starting from the size of the vertex cover returned by REFINED-APPROX-VC-1. If the size returned by REFINED-APPROX-VC-1 is 20, we started the iteration of CNF and 3CNF from 20 and went back to 0. Both CNF and 3CNF will satisfy for the size of REFINED-APPROX-VC-1 as these are the most optimal algorithms. For the first size where the problem is unsatisfied in back iteration, the size + 1 is the most optimal vertex cover then.

6.2 Optimiztion for other algorithms

Initially, we used the recursive algorithms for APPROX-VC-1-REFINED and APPROX-VC-2-REFINED in which we used the output of APPROX-VC-1 and APPROX-VC-2 and made two recursive calls:

1. Call 1 in which we include the current vertex in the final output of Optimised vertex cover
2. Call 2 in which we discard the current vertex and move on to the next.
3. At the end, we test again if the optimized vertex cover is still a vertex cover if it is so then we take a minimum of all optimized vertex covers as the answer.

This algorithm returned the optimal answer as it was checking for each and every combination but this was timed out for vertices = 35 and onwards. As the time complexity of the recursive algorithm is exponential.
As part of optimization, we used greedy algorithms in both refined algorithms which we explained earlier. The optimized greedy algorithms were executed for all vertices to 50 with an execution time of less than 2 mili-seconds.

7 Conclusion

To conclude, CNF and 3 CNF provide the optimal solution but the execution time for these algorithms becomes very large for a large number of vertices. That's why we used encoding to optimize these algorithms to scale for larger vertices. Whereas, the other four algorithms are solved within polynomial time so it computes the vertex cover very quickly but it does give the optimal vertex cover.
For approximation ratio, CNF and 3 CNF will always have a mean of 1 whereas for other algorithms it may be 1 or greater depending on the input data.

References

[1] Andrei Z Broder, Alan M Frieze, and Eli Upfal. On the satisfiability and maximum satisfiability of random 3-cnf formulas. In *SODA*, volume 93, pages 322–330, 1993.

[2] Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

[3] Abraham Flaxman. A spectral technique for random satisfiable 3 cnf formulas. In *SODA*, volume 3, pages 357–363, 2003.

[4] Steven D Prestwich. Cnf encodings. *Handbook of satisfiability*, 185:75–97, 2009.

[5] Joost P Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.