Project Report

In this project, I have to find out vertex cover using 3 approaches:

1. Using CNF-SAT
2. Using Approximate vertex cover by choosing highest degree vertex first.
3. Using Approximate vertex cover by picking edge.

In this report, I will discuss this 3 approaches and analysis regarding performance and find out efficiency of each algorithm.

This three algorithms and input-output of graph, these tasks are performed by 4 different threads in program. There is no critical code in this program because vertex and edges are shared between the algorithms but none of the algorithm change the value of it. There is no need of mutex lock in this program. All the 3 approaches can be executed parallelly.

CNF-SAT

It finds optimal vertex cover from the given graph. The essence of this approach is its encoding. Here the graph is converted into clauses of CNF-SAT. And this reduction is done in polynomial time. There are few rules of encoding from which clauses are generated:

1. Each vertex in vertex cover must be present in graph.
2. Vertices in vertex cover should be unique (means not repeated).
3. Vertices in graph should be unique (means not repeated).
4. For every edge of graph, atleast one vertex of edge should be present in vertex cover.

All the clauses generated using these rules, will be CNF formula. The vertex cover of specific size is possible if the CNF formula produced is satisfiable. And if it is satisfiable then for which vertices of graph it returns true, this are vertices that are included in vertex cover.

This algorithm works optimally only for smaller graphs and takes too much time for computation for big graphs. Therefore, I have restricted the time of execution of algorithm which is 60 seconds. If the thread is not returning vertex cover for given graph before 60 seconds, then thread will be terminated, and I will not check for the further bigger graphs. The reason is if the computation time of given graph exceeds the time limit, then the bigger graphs will take more computation time and it will take more computation time than 60 seconds. The algorithm with input as graph of 15 vertices or less returns vertex cover but the graph with 20 vertices or more exceeds time limit which is 60 seconds, so thread is terminated, and vertex cover is not generated.

So, this was the summary about the CNF-SAT Vertex Cover Algorithm implementation.

Approximation Vertex Cover Algorithm -1 (Approx-1)

The essence of these algorithm is choosing the highest degree vertex and delete all the edges connected to that vertex and add that highest degree vertex in vertex cover. To find minimum-sized vertex cover, the vertex is chosen first which have highest number of edges connected to it. This process will be continued until all edges are deleted, to find the vertex cover.

The efficiency of Approx-1 is not optimal compared to CNF-SAT Algorithm but only for smaller graphs. Apparently, Approx Algorithm works with big graphs also. So, it can be said that CNF-SAT is optimal approach for smaller graph, but this Approx Algorithm is better approach for bigger graph. CNF-SAT takes too much time to compute for bigger graph like if the size of graph is 20 or greater then CNF-SAT would not be able to solve it in efficient time because computation of too many clauses.

Approximate Vertex Cover Algorithm – 2 (Approx-2)

The aim is to find minimum sized vertex cover of graph. The essence of this algorithm is it pick a edge in the graph, add these 2 vertices of edge in vertex cover and delete all the edges connected to those vertices. This process is repeated until all edges are deleted.

Here the main thing is which edge to pick. If we select randomly, I may end up choosing the edge whose vertices are having low degree 1 or 2. So, good option will be picking the edges, which has highest degree vertex in it. But by observation, I found out that number of vertices in vertex cover will be greater than half the number of vertices in graph. Because it adds both the vertices of chosen edge to vertex cover. So, choosing the edge which includes highest degree vertex or choosing the edge randomly or choosing the first edge of graph will not change the size of vertex cover drastically. The computation time will be increased if I tried to pick edge which has highest degree vertex. But there will be literally no additional computation required if we choose the first edge of graph. So, I decided to go for this option of choosing the first edge from graph for better runtime of algorithm.

In terms of efficiency, the computation time required in this approach is less than Approx-1. But the number of vertices in vertex cover of Approx-1 is less than this Approx-2. The reason behind it is in Approx -1, every time only one vertex with highest degree is added in vertex cover but in Approx -2, both the vertices of edges picked is added to vertex cover every time. So, this leads to a greater number of vertices in vertex cover in Approx-2.

Approx-2 also works good with bigger graphs as Approx-1 but CNF-SAT doesn't.

ANALYSIS

In analysis, efficiency of algorithms is calculated and plotted in graph form. The efficiency will be calculated using running time and approximation ratio.

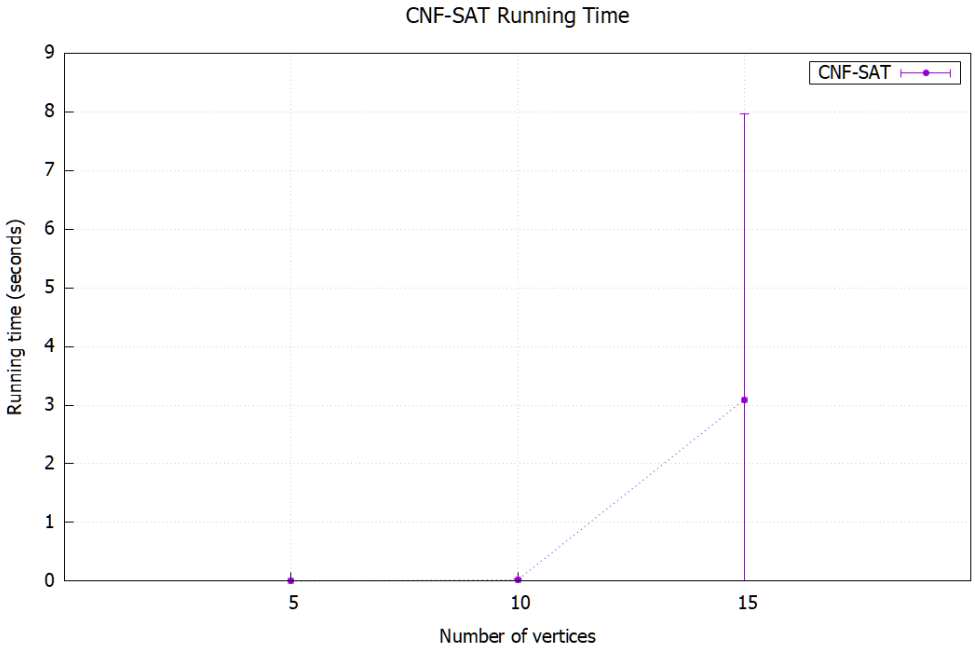
Running Time

CNF_SAT running time is drastically different with other 2 approximation algorithm because of computational time required by clauses of CNF formula.

The running time of CNF-SAT is calculated in seconds and running time of Approximation Algorithms is calculated in milliseconds (ms) because the running time is too small compared to CNF-SAT. It is the reason why I generated two graphs: one for CNF-SAT running time and other for 2 approximation algorithms running time.

In the below plot, running time increases as the number of vertices increases. And standard deviation also increases as the number of vertices increases. The time period set for execution is 60 seconds means if the algorithm cannot return the vertex cover in 60 seconds, then it will be terminated and gives output as "CNF-SAT: timeout". So, while running this thread, I got the result that up to graph of 15 vertices, it returns Vertex Cover but for graph of 20 vertices or greater returns the "timeout". And time consumed

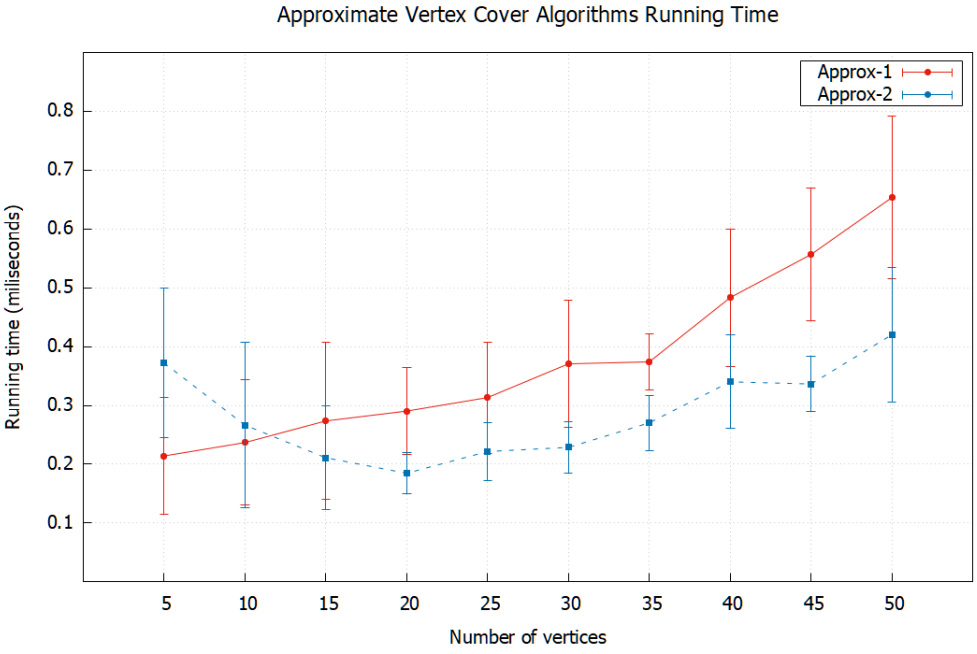
by this CNF-SAT is also dramatically high than other 2 approximation algorithms because of clause computation of CNF formula.



The below plot shows the running time of approximation algorithm 1 as “Approx-1” in miliseconds and running time of approximation algorithm 2 as “Approx-2” in miliseconds in graph.

Approx-1 plot follows the increasing trend, as number of vertices increases, the computation also increases (finding the highest degree vertex until all edges are deleted). So running time also increases.

Overall, running time of Approx-2 is lower than Approx-1 algorithm because of no additional computauion of highest degree vertex. And another reason is Approx-2, appends 2 vertices of edge in vertex cover in each iteration of algorithm, whereas, Approx-1, appends one vertex with highest degree to vertex cover in each iteration.



By the below table, I can specify that why the running time of Approx-2 is high initially then low in middle values and the getting high again slowly and steadily.

Number of vertices	Approx-1 size of Vertex Cover	Approx-2 size of Vertex Cover	Difference of number of vertices in v.c	Number of iteration for approx-1	Number of iteration for approx-2
5	2	4	2	2	2
10	4	8	4	4	4
15	7	10	3	7	5
20	10	15	5	10	8
25	11	18	7	11	9
30	14	22	8	14	11
35	17	25	8	17	13
40	19	30	11	19	15
45	22	32	10	22	16
50	25	37	12	25	19

The running time depends on both number of vertices which leads to number of iterations.

So running time is calculated by number of iterations needed with resprespect to number of vertices in graph and number of vertices in vertex cover. The running time of Approx-1 is higher than Approx-2 because I can simply state that number of iterations plays a big role in calculating the running time of algorithms.

The for first two points means where number of vertices is 5 and 10 there the running time of Approx-2 algorithm is higher than Approv-1 algorithm. The reason for it we have observed the number of iterations needed by both the algorithms is same but size of vertex cover is double. So for those 2 points the running time of Aprrox-1 algorithm is less than Approx-2 algorithm.

For the rest of the points, number of iterations needed in Approx-2 is less than Approx-1 algorithm. The gap between number of iterations of Approx-1 algorithm and Approx-2 algorithm increases as the number of vertices increases.

And for number of vertices between 15 to 50, the size of vertex cover of Approx-2 is greater but not that much, like not double of Approx-1. Additionally, number of iterations needed by Approx-2 is also less. So running time of Approx-2 is less than Approx-1 in most of the cases and running time of Approx-2 increases slowly and steadily as the number of vertices increases.

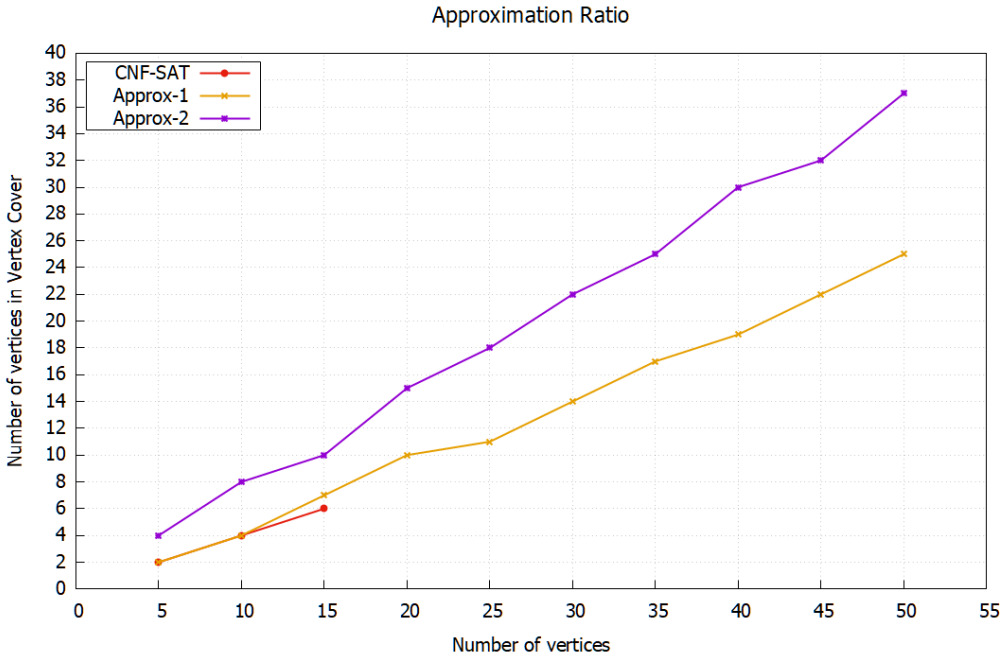
Approximation Ratio

Approximation ratio is ratio of size of vertex cover in each approach. In below graph I have compared the size of vertex cover of CNF-SAT, Approx-1 and Approx-2 algorithms.

As the number of vertices increases the size of vertex cover will also increase in each approach. Because one of the vertex of each edge in graph should be a vertex of vertex cover.

There are only 3 datapoints for CNF-SAT graph because after the number of vertices and edges are greater than 15 in given graph then the running time exceed the 60 seconds time limit means not giving any output. So, vertex cover of vertices 5,10 and 15 are plotted for CNF approach will have vertex cover of 2,4 and 6 vertices, respectively.

In CNF approach, I can observe an interesting pattern which is after every increment of 5 vertices in graph, there is increment if 2 vertices in vertex cover.



Here I can say that CNF-SAT is optimal approach as the size of vertex cover is less in CNF-SAT algorithm than other 2 approaches. But biggest cons of it is, does not work with bigger graphs.

The next better approach is Approx-1. In Approx-1 algorithm, size of vertex cover is always less than or equal to half of the size of given graph. The reason behind it is it finds highest degree vertex of graph and add it in vertex cover in each iteration.

Apparently, in Approx-2 algorithm I can observe that size of vertex cover is always greater or equal to half of size of graph. The reason behind it is in each iteration, add 2 vertices in vertex cover. The edge with lower degree would also be added in vertex cover, because if edge chosen has 2 vertices, one vertex has highest degree and other vertex has lowest degree (like 1), then in this approach both the vertices are added to vertex cover, even though it is only connected to one vertex, both the vertices are added, and these increases the size of vertex cover for Approx-2 algorithm.

As the number of vertices increases of given graph the gap between the size of vertex cover of each approach also increases slowly and steadily.

Conclusion

I have learned how to measure running time and how multithreading is implemented practically. I have also learned to plot graphs in gnu plot and do analysis on output data. My coding methods and skills are also improved. It was a meaningful project as I learned lot of new things.