1 Introduction:

In this project we are going to work on classical vertex cover problem. A vertex cover (or node cover) of a graph is a vertex set that contains at least one vertex of each edge of the graph. The challenge of finding a minimal vertex cover is a classic computer science optimization problem and a good example of an NP-hard optimization problem with an approximation algorithm. There are several use cases of solving minimum vertex cover such as installing minimum traffic cameras at the road junction to cover all the roads in a city. Other than that, the application also extends in the field of computational biology.

In this project, the different approaches are implemented and compared. The project is organized as follows: In section II, we will discuss problem statement. In section III, we will discuss about background of all employed approaches. In section IV, we will talk about implementation. In section V, we will compare all the results and finally end with conclusion in section VI.

2 Problem statement:

For minimum vertex cover problem, input is provided in the form of graph and output in the form of vertex set is needed to be determined. Here, the input graph is provided in the form of maximum vertex V followed by edges in the range [0, V-1].

For Instance:
V 5
E {<2,1>, <2,0>, <2,3>, <1,4>, <4,3>}

It is required to find minimum vertex cover using three approaches. The three approaches used in this project are CNF-SAT-VC, APPROX-VC-1 and APPROX-VC-2. We will discuss algorithms in the next section. The output set of vertices is printed along with the name of all approaches involved as shown below.

CNF-SAT-VC: 2,4
APPROX-VC-1: 2,4
APPROX-VC-2: 0,2,3,4

3 Approach to solve minimum vertex cover:

CNF-SAT-VC – This approach uses a polynomial time reduction to CNF-SAT, and then use a SAT solver to get the final results. For SAT solver, MiniSAT was used to find if it is possible to find assignments to Boolean variables that would make a given expression

true. If the expression is satisfiable, most MiniSAT shows a set of assignments that make the expression "true". Many problems can be broken down into a large SAT problem, so SAT solvers have a variety of uses. This approach results in minimum solution; however, it comes at the expense of run time of the algorithm which increases drastically as the size of graph is increased.

APPROX-VC-1 – In this approach, firstly a vertex with highest degree (i.e., common to maximum number of edges) is picked and the edges connected to vertex with highest degree is removed. This process is repeated until whole graph is covered. This is an approximate method to determine minimum vertex, but this results in very close to minimum vertices and saves significant run time.

APPROX-VC-2 – In this approach, firstly a random edge <u,v>is picked and the incident edges on <u,v>is removed. This process is repeated until complete graph is covered. This method is also an approximate and it was observed that the final minimum vertex may vary with different initial edge <u,v>and may not result in minimum solution. The advantage of this approach is run time is significantly less than other methods and this difference increases as the size of graph is increased.

Please note that, in our c++ code instead of randomly choosing edge ¡u, v¿, always first edge of the remaining edge list is chosen. Therefore, there is no random factor involved and code will result in same output in every run.

4 Implementation:

We have implemented three different approaches to solve the minimum vertex cover problem by using C++ program which has 4 threads, one for I/O, and one each for the three different algorithms. In computer architecture, multithreading is the ability of a central processing unit (CPU) (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system.

In the implemented c++ code, the main the program creates the I/O thread to accept the inputs from the user. In the I/O thread, after getting the input, the program creates three threads for each algorithm. Therefore, the four threads run concurrently. When these three algorithm threads finish, the I/O thread get the results and print them out. The final output is compared and the effect of graph size on each algorithm is discussed in next section.

5 Result and analysis:

We varied number of vertices from 5 to 50 at an interval of 5. However, CNF-SAT-VC approach started exceeding time limit after V=15. Therefore, we could compare all three approaches from V=5 to V=15 only. It is expected that CNF-SAT-VC provides accurate minimum vertex cover solution, and the approximation ratio is defined as "number of vertices in approximate method number of vertices in CNF-SAT-VC". Therefore, this ratio could only be computed till V=15

The figure 1 shows run time comparison all three methods when V is varied from V=5 to V=15. It was observed that run time of CNF-SAT-VC is significantly higher as compared to APPROX-VC-1 and APPROX-VC-2. Since it was hard to observe run time variation in APPROX-VC-1 and APPROX-VC-1, we plotted another graph shown in figure 2 to compare their run time when V is varies from V=5 to V=15. From both these figures, it can be concluded that in general run time of CNF-SAT-VC is maximum, followed by APPROX-VC-1 and least being APPROX-VC-2. The run time, in general increases as number of vertices in graph increased for all three approaches. However, there were few abnormalities in the trend. This can be because of the random graph generated by the function "graphgen" used in the project. For example, there is a spike in figure 2 when V=15, this is because the random graph generated for V=15 required a greater number of iterations in APPROX-VC-1 and APPROX-VC-2 than random graph generated when V=10 and V=20. This abnormal spike can be removed if the number of cases run for each vertex is increased to find more accurate average run time.

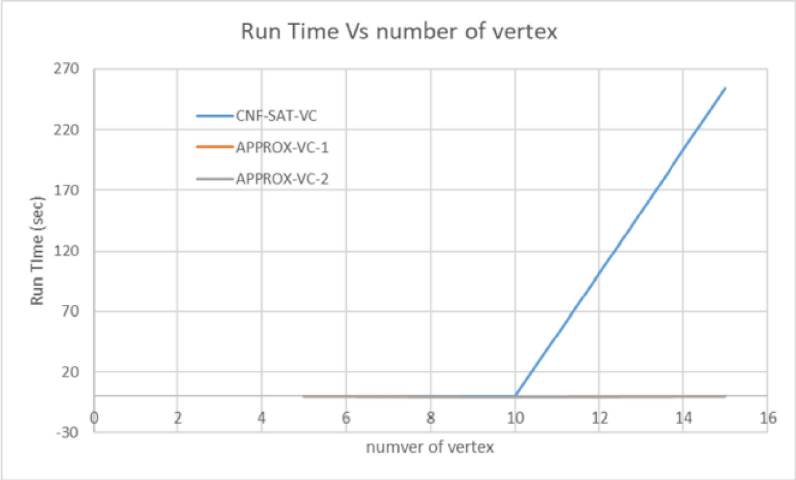

Figure 1: Run Time Vs number of vertex (varying from 5 to 15) and comparing all three approaches.

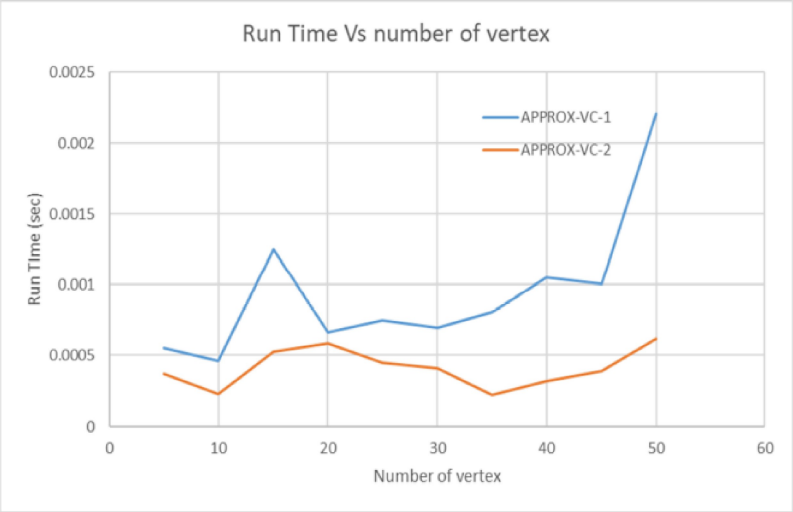

Figure 2: Run Time Vs number of vertex (varying from 5 to 50) and comparing APPROX-VC-1 and APPROX-VC-2.

The figure 3 shows the standard deviation in run time for all three approaches. As the run time was increased, the randomness in run time increased as well, implying that the randomness in run time is directly correlated with the run time. This trend is followed for all three approaches.

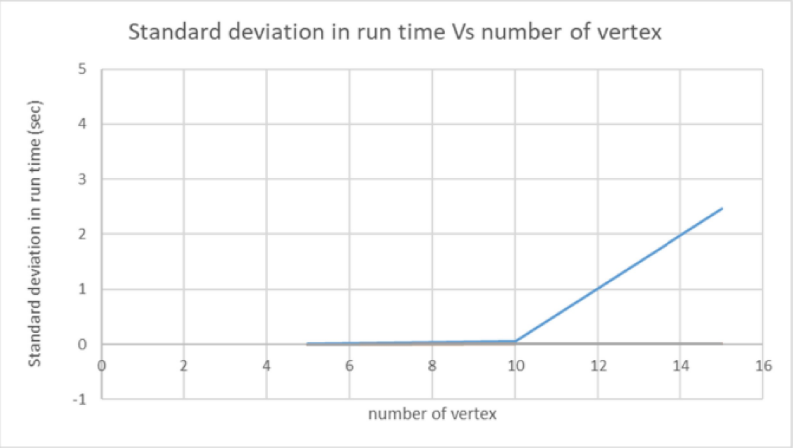

Figure 3: Standar deviation in Run Time Vs number of vertex (varying from 5 to 15) and comparing all three approaches.

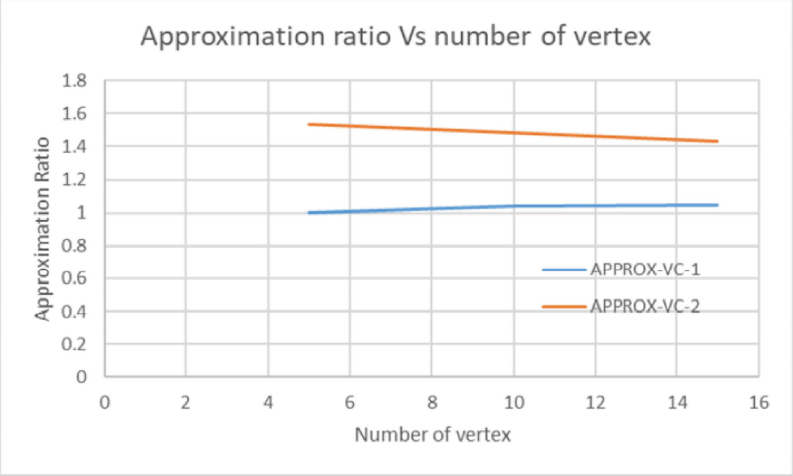

Figure 4: Approximation ratio Vs number of vertex(V varying from 5 to 15)

The figure 4 shows the approximation ratio (number of vertices in approximate method/ number of vertices in CNF-SAT-VC) with respect to number of vertices. This curve determines the efficiency of the approximation methods. The closer the ratio is to value '1', the more efficient the approximation method is.

It can be seen from the figure that the method APPROX-VC-1 has approximation ratio 1 for smaller graph (or lower input vertices number) and increases only slightly above 1 as graph becomes bigger (or higher input vertices number). This implies that APPROX-VC-1 results in very close behavior as CNF-SAT-VC. Or in other words, APPROX-VC-1 is very efficient. On the other hand, APPROX-VC-2 has significantly higher approximation ratio implying that the results obtain is not ideal. As the number of input vertices increases, the approximation ratio of APPROX-VC-2 slightly drops (still very high compared to ideal scenario). This was because for smaller graph (V=5), in general vertex cover solution involved 2 vertices and APPROX-VC-2 resulting in solution involving 3 or 4 vertices. Thus, the ratio is between 1.5 and 2. As the graph size increased the absolute difference between ideal solution and solution by APPROX-VC-2 increased, however, the ratio decreased a little bit. This can change with different set of inputs, but it is still safe to conclude that APPROX-VC-2 is less efficient. The result for approximation ratio was limited to maximum number of vertices till '15', as when it is increased beyond that the CNF-SAT-VC run time increases drastically.

In our code, since there was no random factor involved running same set of input graph resulted in same output everytime. Or in other words, standard deviation is "0". This is because as explained in section III, instead of choosing random edge in APPROX-VC-2,

always the first edge is chosen. While in APPROX-VC-1 and CNF-SAT-VC, there is no random variable involved. The ratio may vary with different set of edges having same number of vertices, but it will always give same solution for a particular set of edges and number of vertices.

6 Conclusions:

Three different approaches for solving minimum vertex cover problem in a graph was implemented and studied. It was observed that the run time of all approaches increased as the size of graph was increased. In particular, the run time of CNF-SAT-VC increased drastically with graph size. The run time of APPROX-VC-1 and APPROX-VC-2 was significantly lower and within the range even if number of input vertex was increased to '50'. It was also observed that APPROX-VC-1 provided solution very close to ideal or solution by CNF-SAT-VC. The run time for APPROX-VC-2 was very less, but solution is far from optimal. It is safe to say that APPROX-VC-1 approach is the most efficient approach among all three considering the run time and the optimality of the vertex cover solution.