ECE650 - Foundations of Software Engineering

Department of Electrical and Computer Engineering

Project

Authors:
Vijiithaa Sasidharan (ID: 20969795)
Bala Murli Krishnan Muthu (ID: 20936500)

1 INTRODUCTION

The main objective of this project is to implement additional ways to solve the minimum vertex cover problem. The problem of finding a minimum vertex cover is a classic optimization problem and is a typical example of a NP-hard optimization problem. In this report, along with the approach used for **CNF-SAT-VC**, two more additional ways to solve minimum vertex cover called **APPROX-VC-1** and **APPROX-VC-2** have been implemented. The effectiveness of the above-mentioned techniques has been calculated in relation to the number of vertices.

2 ALGORITHMS

Techniques used to solve minimum vertex cover for this project have been explained below:

2.1 CNF-SAT-VC

The reduction of vertex cover to CNF is a polynomial-time reduction. The graph is implemented as a formatted set of clauses in CNF. CNF is a conjunction of one or more clauses, where a clause is a disjunction of literals and a literal is a variable or its negation. The MINISAT solver is provided with the clauses as given in section 4 to determine if the provided conditions are satisfied to check if a vertex cover exists for the current graph or not. SAT solver checks all the possible combinations of vertices to find minimum vertex cover.

2.2 APPROX-VC-1

In this approach, the vertex is chosen with the highest degree. This vertex is added to the vertex cover list and we remove the edges that are attached to this vertex. This process is repeated until no edges remain.

2.3 APPROX-VC-2

In this approach, an arbitrary edge (u, v) is chosen from the set of edges E and add this u and v to the Vertex cover list. Remove all the edges that have been connected to each of these vertices u and v. This process is repeated till no edges remain.

To create a steady testing environment, a thread has been created for each of the different approaches to run the three algorithms concurrently. In first thread, we run CNF-SAT-VC, in second thread APPROX-VC-1 is executed and in the other thread APPROX-VC-2 is executed. The “pthread_timedjoin_np()” method has been used to conduct a join-with-timeout for the SAT solver, as big vertices increase the execution time significantly. With the provided timeout, execution of CNF-SAT-VC terminates gracefully.

3 METHODS USED FOR ANALYSIS

We utilise two parameters to determine the efficiency of these three algorithms with respect to the number of vertices: (1) Running time and (2) Approximation Ratio.

3.1 RUNNING TIME:

The "Running time" is analyzed by measuring the CPU clock time for the whole period of an individual thread. We utilise the "pthread_getcpuclockid()" cvxfddfxc library to get the CPU execution time for the thread. We can determine the running time for each method and the maximum CPU time taken by an algorithm based on the number of vertices provided as input.

3.2 APPROXIMATION RATIO:

The "approximation ratio" is defined as the ratio of the size of the computed vertex cover to the minimum vertex cover obtained by CNF-SAT-VC(as we find this optimal).

For each vertices, ten graphs have been created. For each of these graphs, the "Running time" and "Approximation Ratio" have been computed. "Running Time" has been computed for each of these graphs for at least 10 runs. Then the "average" value and "standard deviation" for 100 runs for each value of vertices have been computed.

4 CNF - SAT CLAUSES:

The reduction for the minimum vertex cover to the CNF consists of the following clauses:

Clause 1: At least one vertex is the ith vertex in the vertex cover.
$\forall i \in [1,k]$, a clause $(x_{1,i} \vee x_{2,i} \vee .... \vee x_{n,i})$

Clause 2: No one vertex can appear twice in a vertex cover.
$\forall m \in [1,n]$, $\forall p,q \in [1,k]$ with $p < q$, a clause $(\neg x_{m,p} \vee \neg x_{m,q})$

Clause 3: No more than one vertex appears in the mth position of the vertex cover.
$\forall m \in [1,k]$, $\forall p,q \in [1,n]$ with $p < q$, a clause $(\neg x_{p,m} \vee \neg x_{q,m})$

Clause 4: Every edge is incident to at least one vertex in the vertex cover.
$\forall < i,j > \in E$, a clause $(x_{i,1} \vee x_{i,2} \vee ....x_{i,k} \vee x_{j,1} \vee x_{j,2} \vee ....x_{j,k})$

The running time for a graph with a constant number of vertices is proportional to the predicted size of the vertex cover. Rather than linear probing, we use binary search to determine the minimum vertex cover in our implementation.

The main factor is the number of vertices in the graph. It is obvious that (n choose 2) will lead $O(a^n)$, that is, with an increased number of vertices contributes to the exponential growth of the time consumption.

We drew the graphs separately because compared to the approaches "APPROX-VC-1" and "APPROX-VC-2", CNF-SAT-VC takes a much longer running time.

5 Running Time Analysis of CNF-SAT-VC

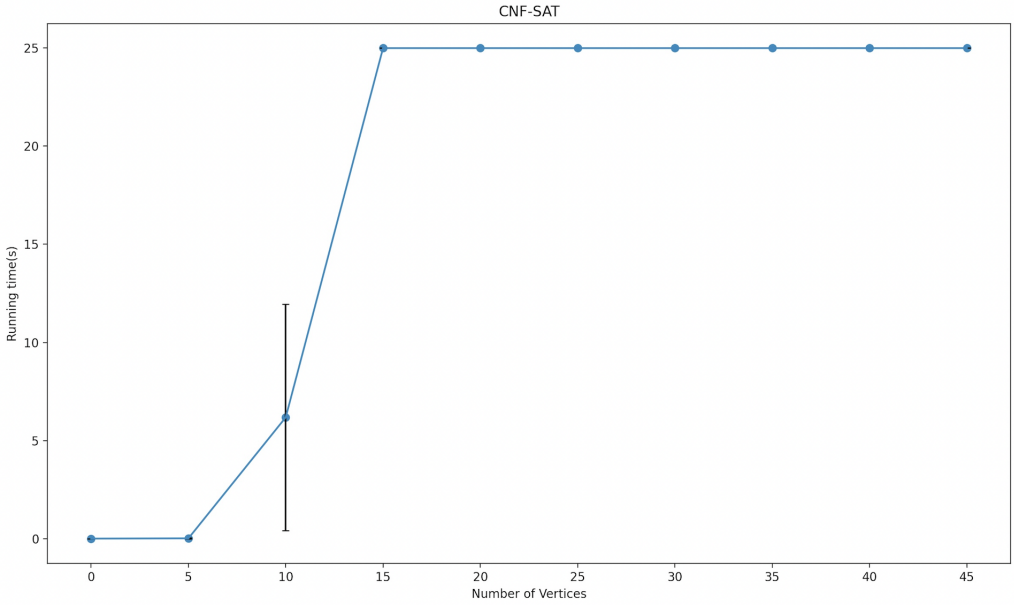

Figure 1: Running time of CNF-SAT-VC vs Number of Vertices.

Here, the “x-axis” represents the “Number of Vertices” and “y-axis” represent the ”Running time” in seconds.

We use “GraphGen”, to produce different graphs with the different numbers of vertices and it has been generated in increments of 5.

As shown in the Figure 1, it is also clear that the performance of SAT solver is very high for the vertices (5..10). When the number of vertices increases from 15, the time taken to find the minimum vertex cover is very high. Sometimes the time taken for execution of SAT-Solver may be in minutes or hours. From various iterations of the run, we have assumed the “timeout” value to be “5 minutes”. We can observe an exponential change between vertex 10 and 15 and it remains the same when the number of vertices is greater than 15. This is because as the number of vertices v increases, it increases the number of clauses $k * v * (v-1)/2$ where k is the size of the vertex cover. In the case of the small number of vertices, it takes less number of clauses. So the time taken is lesser, as it has to satisfy less number of combinations of variables, because of which exponential growth of SAT solver is justified.

For each vertex, 10 graphs have been generated and each of these graphs has been run 10 times. During each execution, the time taken would differ based on the workload, CPU etc. and hence there would be a minor deviation. On calculating the mean of these 100 runs for each vertex, “running time” has been plotted in “Blue” dots and “errors bars” which are clearly visible for the

vertex 10, have been plotted in the "Black" line. Error bars for other vertices are very minuscule, hence it isn't clearly visible.

6 Running Time Analysis of "APPROX-VC-1"

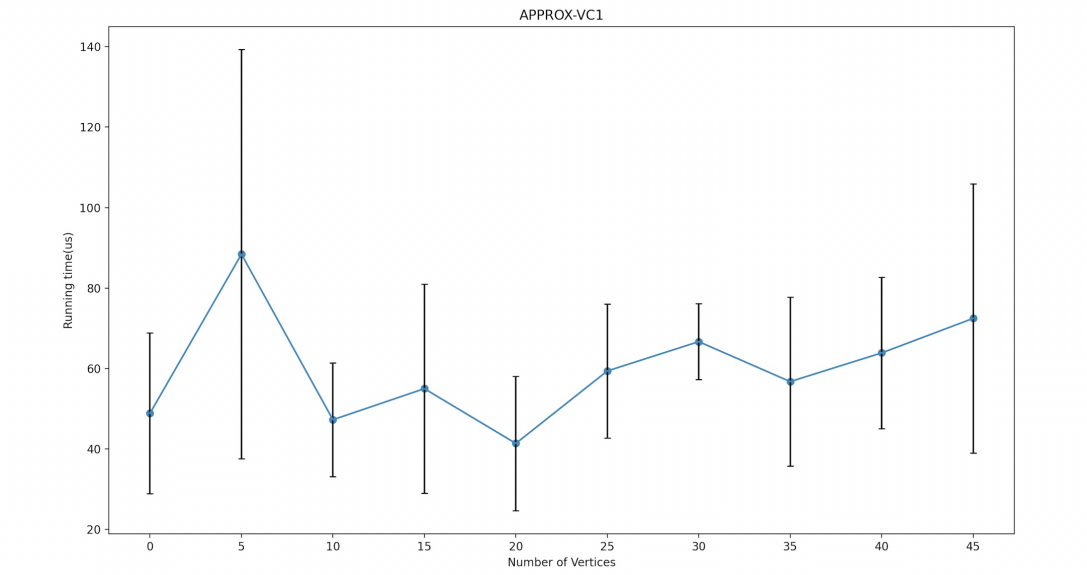

Figure 2: Running time of APPROX-VC-1 vs Number of Vertices

Here, the "x-axis" represents the "Number of Vertices" and the "y-axis" represent the "Running time" in microseconds.

APPROX-VC-1 calculates the maximum degree of the vertex and then it finds the vertex cover list by visiting each node in every iteration and removes the vertices associated with it. Then this process is repeated until no edges are left. Locating the most incident vertex and removing all related vertices takes $O(n)$ of time in "APPROX-VC-1". This process will be iterated at most n times, resulting in an $O(n^2)$ running time.

We analysed the running time of "APPROX-VC-1" by plotting the mean and standard deviation for each value of V in 5..50 as shown in the Figure 2. From the graph, we can observe that the "standard deviation" for vertex 5 is high because there would be a minor deviation in the running time for each run. For different graphs with the same number of vertices, the running time is varied the most in this case. It's also due to the fact that different graphs have different edge connections. Based on that number of vertex cover also varies. This variation is much greater in the case of vertex 5. As a result, the time it took to calculate them differed. In this scenario, the maximum standard deviation is obtained. For APPROX-VC-1, the running time increases with the increase in the vertex in most cases.

7 Running Time Analysis of "APPROX-VC-2"

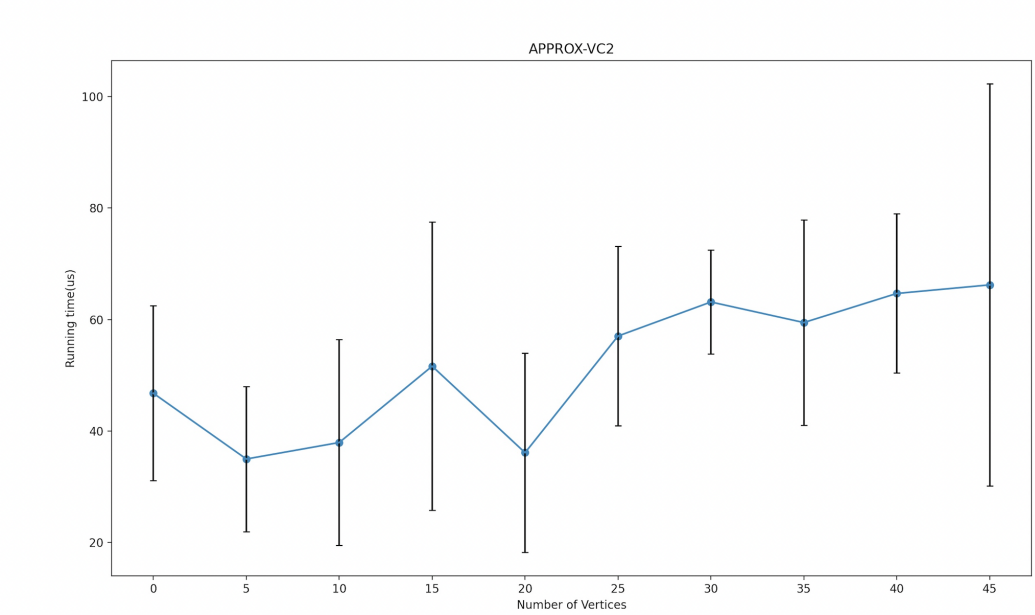

Figure 3: Running time of APPROX-VC-2 vs Number of Vertices

The time taken by "APPROX-VC-2" is less in comparison with "APPROX-VC-1" as we pick random edges and we remove the vertices associated with them. To remove all connected vertices in "APPROX-VC-2", requires $O(n)$ time. This process will be iterated at most n times, resulting in an $O(n^2)$ running time.

We analysed the running time of "APPROX-VC-2" by plotting the mean and standard deviation for each value of V in 5..50 as shown in the Figure 3. From the graph, we can observe that the "standard deviation" for vertex 45 is high, as the running time was steadily increasing with the number of vertices.

The difference in standard deviation between APPROX-VC-1 and APPROX-VC-2 is due to a difference in their algorithms. Each selection is defined in "APPROX-VC-1" (always the most incident edge), but there is no control over each selection in "APPROX-VC-2" (arbitrary selection). As a result, the output of "APPROX-VC-2" for the same graph may differ, whereas "APPROX-VC-1" always produces the same result for the same graph. When compared to "APPROX-VC-2", "APPROX-VC-1" delivers more stable findings throughout a wide range of graphs with varying vertices sizes, resulting in less variation.

8 Approximation Ratio Analysis of “APPROX-VC-1” & “APPROX-VC-2”

We have taken the “Optimal” solution to be “CNF-SAT-VC” and hence approximation ratio has been defined by comparing the vertex cover of “APPROX-VC-1” and “APPROX-VC-2” with “CNF-SAT-VC”.

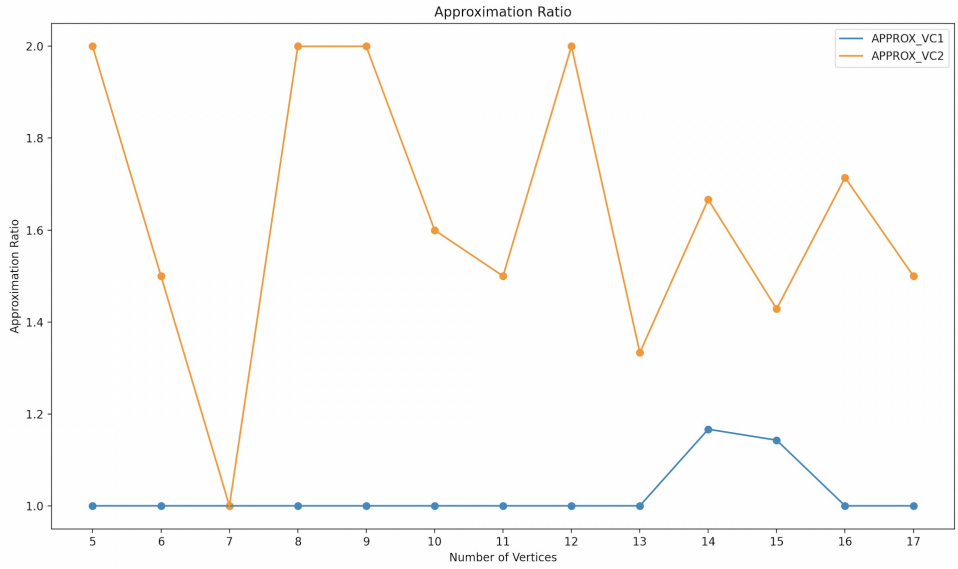

Figure 4: Approximation ratio vs Number of Vertices.

We may deduce from the Figure 4, that the minimum vertex cover for each vertex generated by “APPROX-VC-1” is nearly identical to that generated by the optimal algorithm “CNF-SAT-VC”. Because “CNF-SAT-VC” ensures that the minimum number of vertex cover is found. The approximation ratio in most of the vertex is equal to 1 for “APPROX-VC-1”, but this is not the case for “APPROX-VC-2”. We can visualise the spike for “APPROX-VC-2” from the Figure 4 and it has the worst approximation ratio compared to the other two algorithms since it takes the arbitrary vertex rather than the highest degree vertex, and it has more chances to calculate a bigger vertex cover than “APPROX-VC-1”. It’s likely that the graphs were tightly connected, but because this technique doesn’t check for maximum connectivity, it deletes edges one by one, resulting in a solution that’s nearly twice as large as the best option.

9 CONCLUSION:

Based on the analysis presented above, we can infer that “CNF-SAT-VC” gives the most accurate results, whereas “APPROX-VC-2” is time-efficient when compared between these two algorithms “APPROX-VC-2” and “APPROX-VC-1”. CNF-SAT-VC’s time efficiency grows exponentially with the

number of vertices, while “APPROX-VC-1” and “APPROX-VC-2” time efficiency grows polynomially with the number of vertices with respect to the same vertices input. “CNF- SAT”s performance is strongly dependent on the input graph provided. The behaviour of “APPROX-VC-2”, on the other hand, is mostly independent of the arbitrary edge. Meanwhile, the performance of “APPROX-VC-1” is moderately affected by input.

“APPROX-VC-1” and “APPROX-VC-2” wouldn’t be recommended for large graphs but they are efficient in terms of time and space complexity. “CNF-SAT-VC” would be the best optimal solution to produce a minimal vertex cover for a real-world application.