UNIVERSITY OF WATERLOO

ECE 650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FINAL COURSE PROJECT

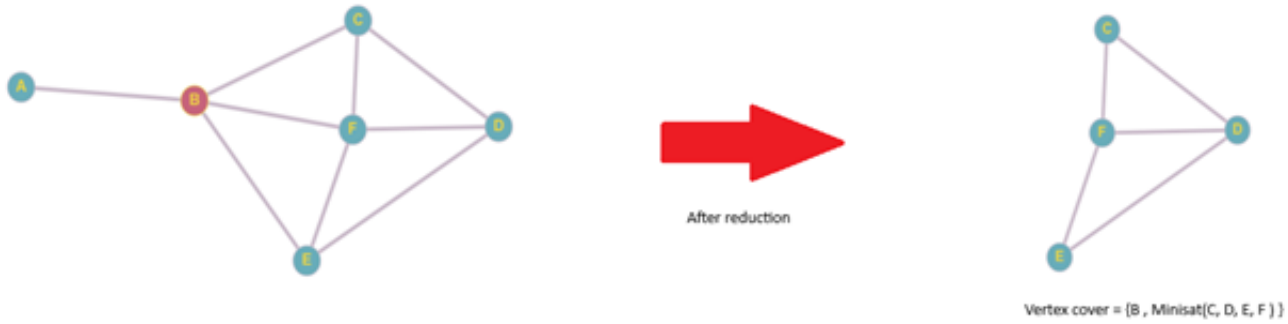
1 Objective of the Project:

The goal of the project is to create a C++ program that solves and analyzes the vertex cover optimization solution using three different approaches: CNF-SAT-VC, APPROX-VC-1, and APPROX-VC-2. This involves running each of these logics in parallel using threads.

2 Optimizations Used:

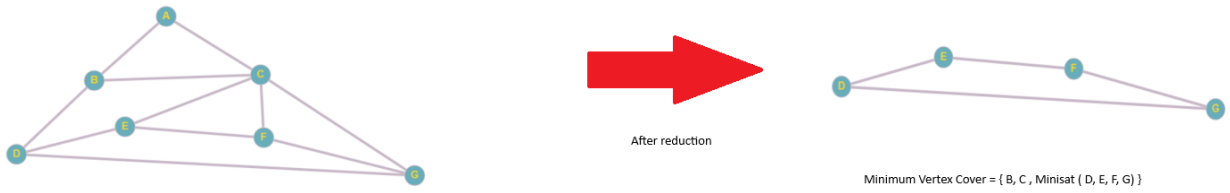
2.1 Reducing the graph size, thereby reducing the number of clauses:

A) When there is a case of an edge with vertices A and B, where vertex A is connected to only one other vertex B, and vertex B is an intersection with connections to more than one vertex, we can add vertex B to our vertex cover and remove all edges corresponding to vertices A and B. For example, by applying the above reduction, our graph decreases in size as shown below.



When the complete graph is passed to the SAT solver, in the edge 'AB', the solver must choose either vertex 'A' or 'B' to achieve satisfiability. Regardless of MiniSAT's choice, manually selecting Vertex 'B' instead of Vertex 'A' will guarantee us to arrive at the minimum vertex cover.

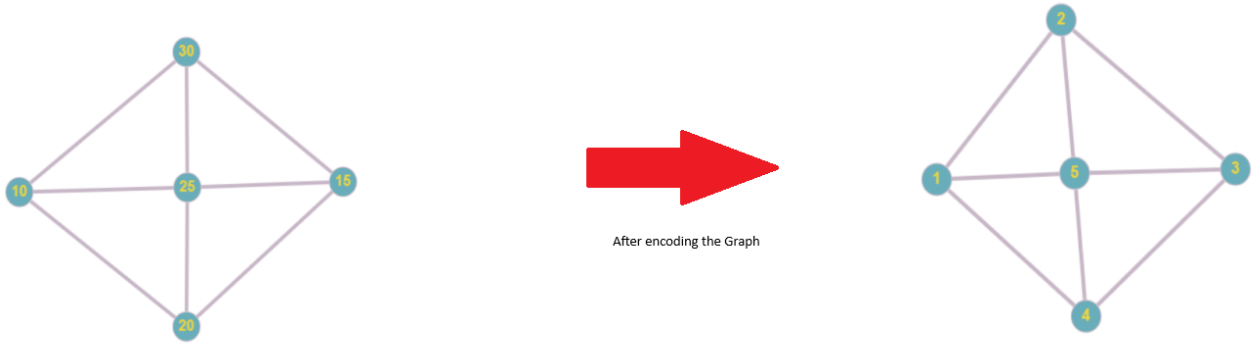
B) In a graph, if we have any three vertices in a triangular pattern, let's say vertex A, B, and C form a triangle, and at least one of these vertices is not connected to more than two other vertices, we can apply the following reduction. In such instances, we only need to select two vertices for the minimum vertex cover. While selecting vertices for the SAT Cover, we prioritize the vertices connected to more than two other vertices and then remove the edges associated with these vertices that form the triangle under consideration, resulting in a reduced graph. This optimization is illustrated below:



C) In instances where a pentagon is identified, a comparable reduction strategy can be applied reduction as above. The objective is to select three points forming a crucial part of the minimum vertex cover, with a preference for vertices having connections to more than two other vertices, as this prioritization effectively covers a greater number of edges or streets. Subsequently, the edges associated with the vertices forming the pentagon under consideration can be removed, leading to a streamlined or reduced graph. This optimization process is visually demonstrated in the illustration below:

2.2 Scaling down the Number of Vertices to reduce the number of clauses :

An effective clause reduction could be done by mapping the vertices of a given graph and assigning them new vertex numbers. The major purpose of this method is to reduce the total number of vertex counts, which results in a large reduction in the number of linked clauses. The encoded graph is then entered into Minisat. Following the production of the vertex cover with Minisat, the vertex numbering is reversed to align with the vertices of the original graph.



3 How we use these Reduction Strategies?

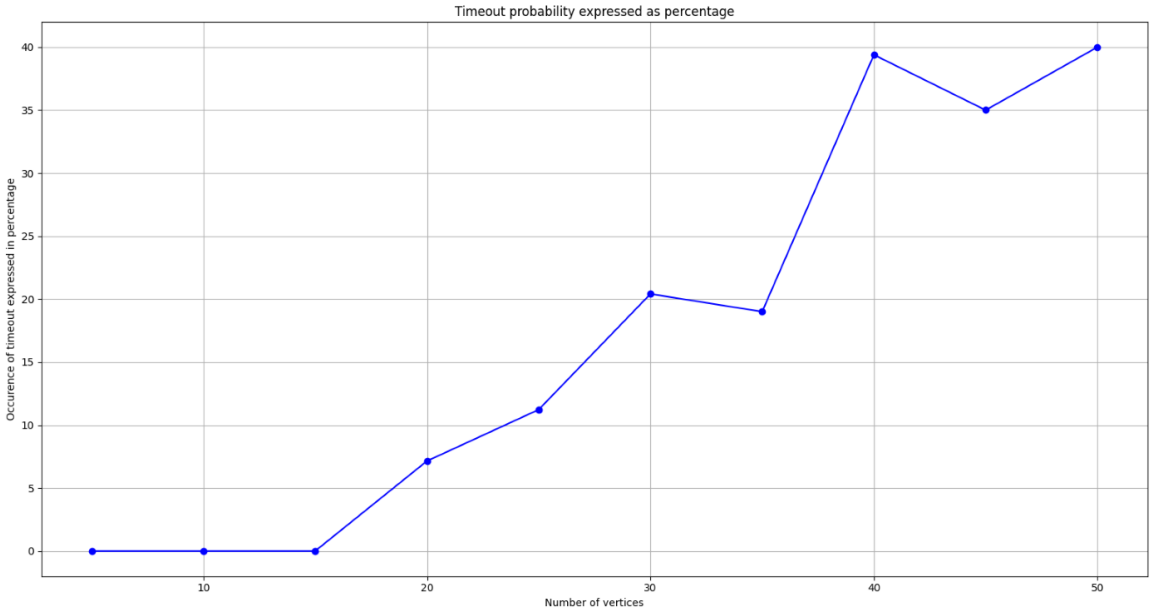
All the reductions mentioned in **2.1** are applied repeatedly until there is no scope for reduction left. By implementing these optimizations, we've successfully solved most test cases (generated using graphGen from eceubuntu) within a reasonable timeframe. However, a few instances still require an extended duration, depending on the graph's complexity. Since our optimization strategy relies primarily on graph reduction, if the reduced graph comprises fewer than 15 vertices, our code can generate the minimum vertex cover; otherwise, it times out.

4 Test Results

For our testing, the graphs generated by the program (/home/agurfink/ece650/graphGen/graphGen) were used. The choice of these graphs was purely random.

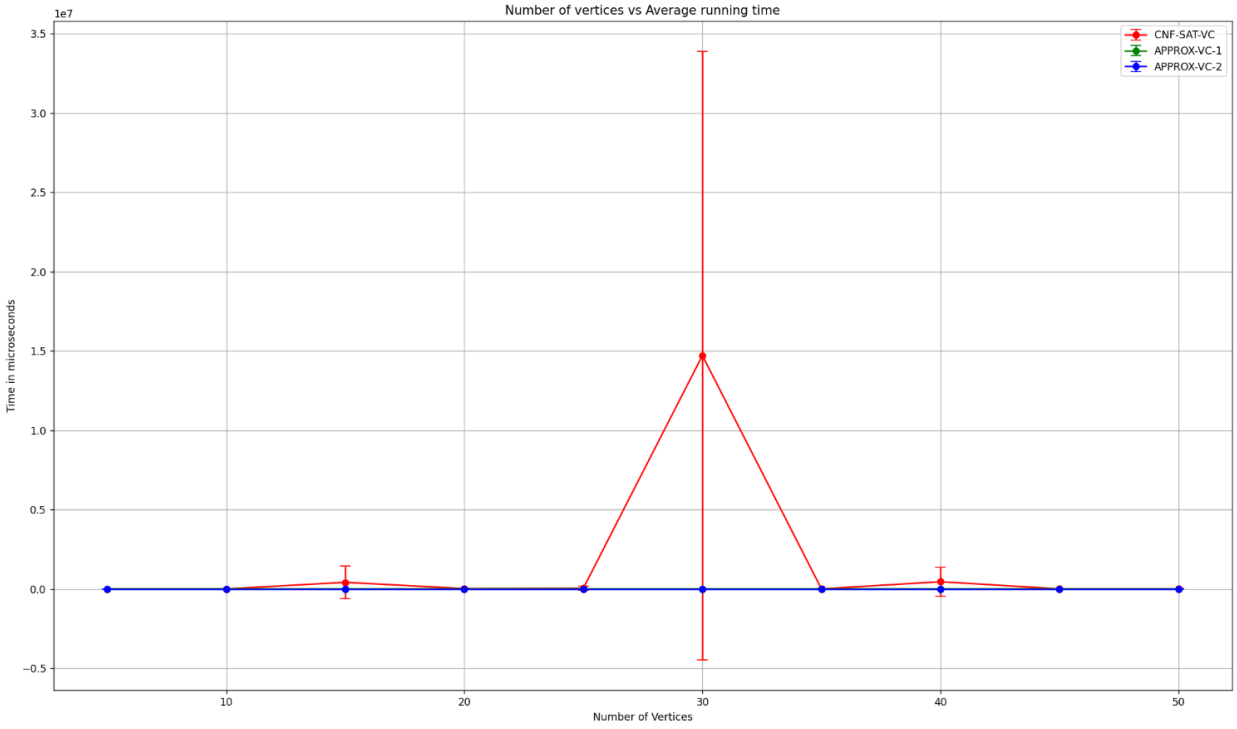
4.1 Observation 1: Timeout Trend of Graphs vs No. of Vertices

In the scenario below, a random selection of 100 graphs for each V5, V10, V20, V25, V30, V35, V40, V45, V50 was tested, and the percentage of graphs that timed out was measured.
[*Notation - VX means No. of Vertices in the graph = X]



The enhanced CNF-SAT-VC algorithm demonstrated a remarkable success rate, successfully resolving 82.72 % of the test cases. In contrast, the traditional CNF-SAT-VC algorithm achieved a significantly lower success rate, approximately 30%. Notably, the timeout occurrences in the improved algorithm are intricately tied to the number of vertices in the final reduced graph. A crucial observation is that a more substantial reduction in the graph size corresponds to an increased likelihood of successfully arriving at a solution. Analyzing the graphical depiction highlights the relationship between the timeout rate and the number of vertices even further. The final reduced graph's vertex count increases proportionally to the number of vertices in the given graph. This increase in vertex count directly correlates to a greater likelihood of encountering timeouts.

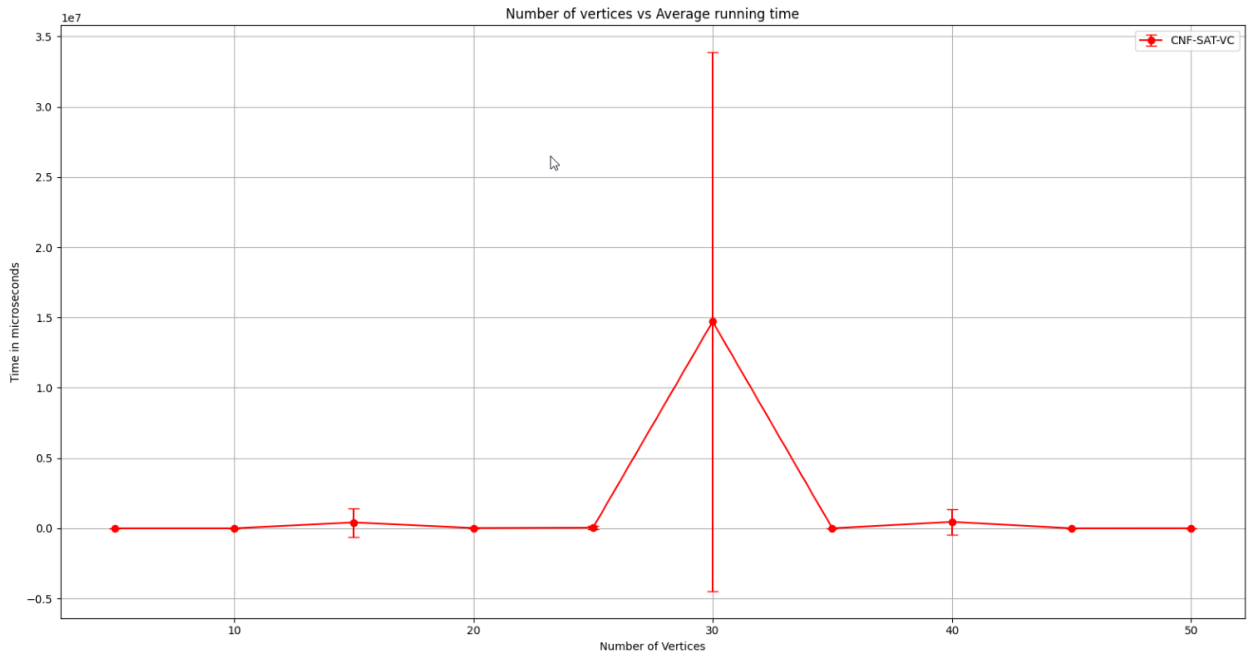
4.2 Observation 2: Average Running Time of All Three Vertex Cover Algorithms



In the above graph, two trends are evident. The CNF-SAT-VC operates on a timescale of seconds, while APPROX-VC-1 and APPROX-VC-2 operate in Microseconds. Therefore, it would be better to analyze them separately.

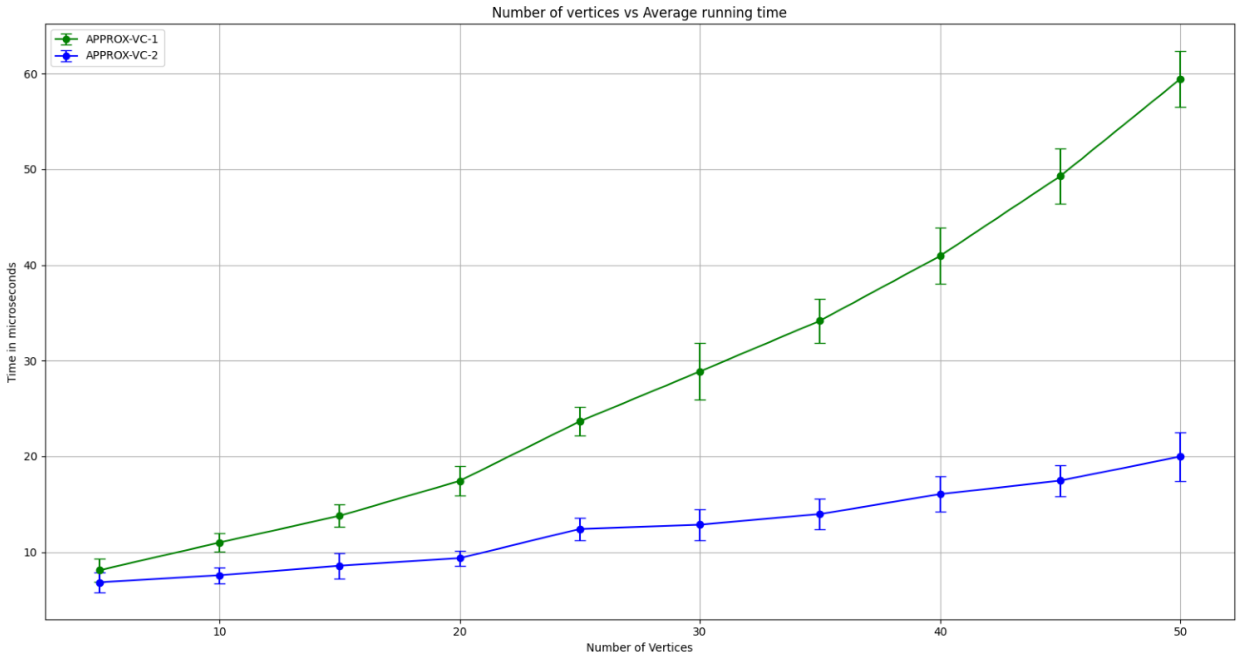
4.2.1 Case 1: CNF-SAT Average Runtime

An intriguing observation in the CNF-SAT-VC results is evident at the y-axis corresponding to vertex number 30. This anomaly is attributed to the existence of a few infrequently encountered graph types. These graphs, unlike the majority of test cases resolved in microseconds, pose a unique challenge. The SAT solver, tasked with finding a solution, requires several seconds due to the inherent complexity of these particular instances.
The distinctive nature of these graphs places them in a middle ground on the complexity spectrum. They are neither intricate enough to trigger timeouts nor straightforward enough for instantaneous resolution in microseconds. The sporadic and random occurrence of these atypical cases within the test pool elucidates the singular observation at vertex number 30. Additionally, the wide standard deviation observed in such instances further emphasizes the rare and unpredictable nature of these particular graphs, contributing to the observed behavior in the results.



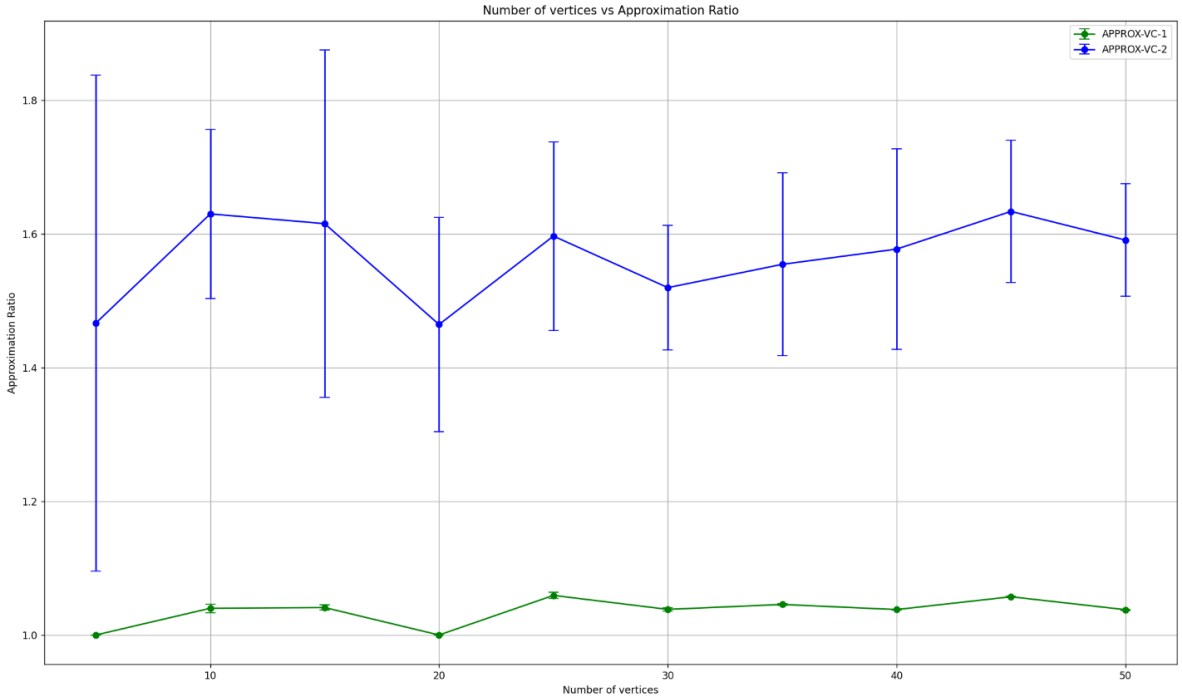
4.3 Case 2: Approx-VC-1 and Approx-VC-2 Average Runtime

The graph composition of the APPROX-VC-1 and APPROX-VC-2 algorithms demonstrates predictability. Notably, both methods execute quickly, at the microsecond scale, with computing times scaling linearly with the number of vertices. While both techniques perform similarly at low vertex counts, their efficiency difference becomes increasingly apparent as the number of vertices grows. In scenarios with an increasing number of vertices, APPROX-VC-2 outperforms APPROX-VC-1 in terms of efficiency, establishing it as the superior algorithm. This discrepancy in performance emphasizes the importance of method selection, particularly in circumstances with greater graph sizes. It is also worth noting that in some cases, the generated cover from Approx VC-2 may not be the smallest possible vertex cover for the given graph. So, while APPROX-VC-2 is efficient, quality of the vertex cover may take a hit.



4.4 Observation: Approximation Ratio Calculation

The graph below shows the approximation ratio for APPROX-VC-1 and APPROX-VC-2. An approximation ratio is the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover.
The graph clearly indicates that APPROX-VC-1 has a lower Approximation ratio compared to APPROX-VC-2. A lower Approximation ratio means the computed cover is closer to the optimal vertex cover. APPROX-VC-1's vertex cover results are very close to the optimal cover, explaining why its standard deviation is negligible. On the other hand, we observe that APPROX-VC-2 has a poor Approximation ratio, indicating that the computed vertex cover is not close to the optimal cover, and the wider standard deviation reveals the instability of the approximation ratio. The instability of the Approximation ratio in this case can be attributed to the random nature of the test cases used.



5 Conclusion:

The optimizations used in the improved CNF-SAT-VC algorithm drastically improve the solving rate compared to the traditional CNF-SAT-VC algorithm. In the case of approximation algorithms, we clearly observe an accuracy vs. runtime trade-off between APPROX-VC-1 and APPROX-VC-2