# Analysis of Vertex Cover Using Various Algorithms

**ECE650**

**Methods and Tools for Software Engineering**


**University of Waterloo**

**Department of Electrical and Computer Engineering**

**Submitted By:**

Lovepreet Singh Gill (ID: 21167690)

Arshita (ID: 21115687 )

**Date: November 25, 2024**

# 1 Introduction

The primary objective of this project is to analyze and implement algorithms for solving the parameterized Vertex Cover problem, a classical optimization problem in graph theory. Formally, a vertex cover of a graph G(V,E) is a subset of vertices V such that for every edge (u,v) in E, at least one of the vertices u or v is in the vertex cover. Finding a minimum vertex cover is an NP-hard problem, making it computationally infeasible to solve optimally for large graphs. This has led to the development of approximation algorithms to provide near-optimal solutions efficiently. In this project, three approaches have been implemented and analyzed:

- Encoding the Vertex Cover problem into CNF-SAT clauses and solving it using the MiniSat solver.
- Using two approximation algorithms, Approx-VC-1 and Approx-VC-2, to compute the vertex cover for comparison.

# 2 Algorithms

## 2.1 CNF-SAT

The CNF-SAT approach involves reducing the Vertex Cover problem into a Boolean satisfiability problem. This is achieved through a polynomial-time reduction. The graph is encoded as a set of literals and clauses in Conjunctive Normal Form (CNF), using A4 encoding. CNF is a conjunction of disjunctions of literals, where literals are joined using OR operations, and clauses are connected through AND operations. These ANDed clauses are passed to the MiniSat solver, which determines whether the clauses are satisfiable. If satisfiable, the solution provides the vertex cover for the given graph.

## 2.2 Approx-VC-1

The Approx-VC-1 algorithm selects the vertex with the highest degree (i.e., the vertex most frequently occurring in the edge set). This vertex is added to the vertex cover, and all edges connected to it are removed from the graph. This process is repeated iteratively until no edges remain in the graph.

## 2.3 Approx-VC-2

The Approx-VC-2 algorithm selects an arbitrary edge (u,v) from the graph's edge set. Both endpoints u and v of the edge are added to the vertex cover. All edges connected to u and v

are then removed. This process continues until no edges remain in the graph.

To optimize the computation, a multithreaded approach is used with four threads: one thread is dedicated to I/O, responsible for reading input and creating adjacency lists, and three separate threads handle the computations for CNF-SAT, Approx-VC-1, and Approx-VC-2. This parallel execution enhances efficiency and reduces computational time.

# 3 Methods Used for Analysis

To evaluate the efficiency of the three algorithms, two key factors are considered: (1) *Running Time* and (2) *Approximation Ratio*. For each value of the number of vertices V, 10 graphs are generated using the `GraphGen` tool. This allows us to compute both the running time and the approximation ratio for each graph. To ensure a fair evaluation, the graphs are generated with varying edge configurations but the same number of vertices.

## 3.1 Running Time

The running time of each algorithm is measured by tracking the CPU time consumed by their respective threads. This is accomplished using the utility `pthread_getcpuclockid()`, which provides the CPU execution time for a thread. For each input graph, the running time of all algorithms is calculated, and the maximum CPU time taken by any algorithm is recorded. These values are then used to compute the average and standard deviation.

## 3.2 Approximation Ratio

The *Approximation Ratio* is defined as the ratio of the size of the vertex cover computed by an algorithm to the size of the optimal (minimum-sized) vertex cover. This metric provides insight into the accuracy of the solutions generated by the approximation algorithms.

For the generated graphs, we calculate the approximation ratio and running time across different values of V (number of vertices). The values of V range from 5 to 50, increasing in steps of 5. For each value of V, 10 graphs are created, and each algorithm is run 10 times on each graph, resulting in 100 total runs for each value of V. The average and standard deviation are computed across these 100 samples to ensure robustness of the results.

# 4 Analysis

## 4.1 Running Time

The running times of the CNF-SAT-VC, Approx-VC-1, and Approx-VC-2 algorithms are analyzed and plotted in Figures 1 and 2. The x-axis in both figures represents the number of vertices V, while the y-axis represents the running time in microseconds ($\mu s$).

As shown in Figure 1, the CNF-SAT algorithm's running time increases exponentially with the number of vertices. For smaller graphs, the execution time is lower due to fewer literals and clauses. However, as the number of vertices increases, the number of literals and clauses grows, significantly increasing the computational complexity. This behavior aligns with the fact that CNF-SAT is computationally intensive for larger graphs. Additionally, the variance in execution time (indicated by the standard deviation) becomes more pronounced for larger graphs, particularly for 15 vertices.

Figure 2 depicts the running times of Approx-VC-1 and Approx-VC-2 for graphs with vertices ranging from 5 to 50. Both algorithms show an increasing trend in execution time as the number of vertices grows. Approx-VC-1 consistently takes longer than Approx-VC-2 due to its computational overhead in identifying the vertex with the highest degree and removing its associated edges iteratively. In contrast, Approx-VC-2 employs a simpler approach by selecting an edge at random, which results in fewer computations per iteration.

The variability in execution times for Approx-VC-1 and Approx-VC-2 also increases as the graph size grows, as seen in the standard deviation values. This indicates that larger graphs introduce greater complexity, leading to more diverse execution times.

The CNF-SAT algorithm is highly accurate but becomes infeasible for larger graphs due to its exponential time complexity. On the other hand, Approx-VC-1 and Approx-VC-2 provide faster solutions, with Approx-VC-2 being the more time-efficient choice. However, the trade-off lies in Approx-VC-2's slightly lower precision compared to Approx-VC-1.

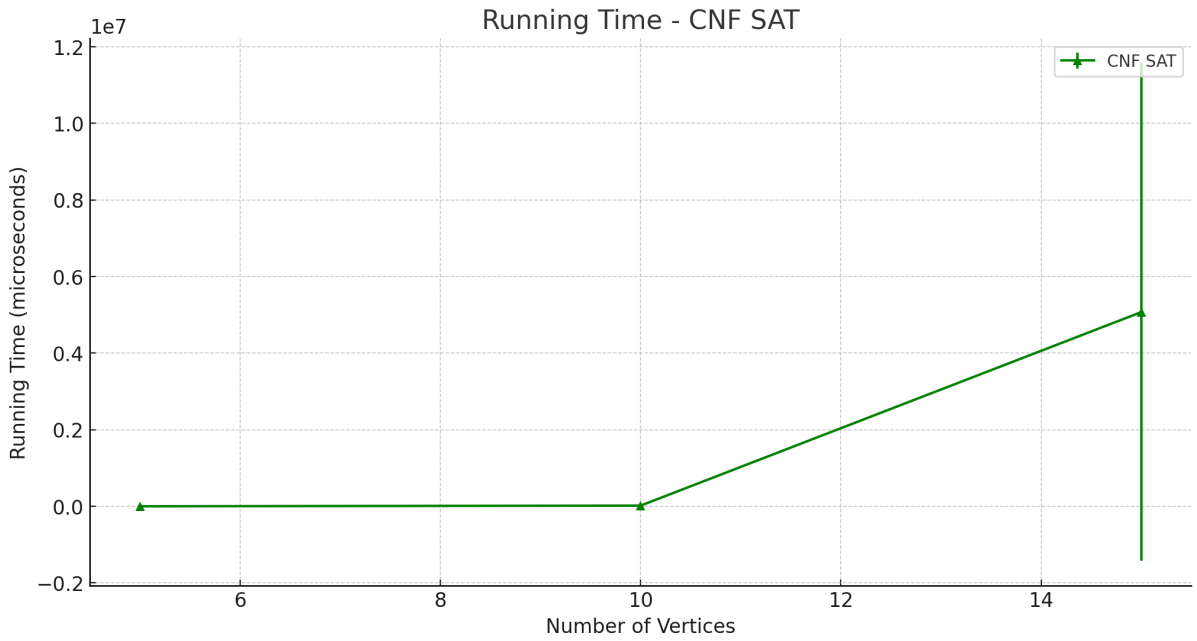


**Figure 1:** Running time of CNF-SAT-VC.

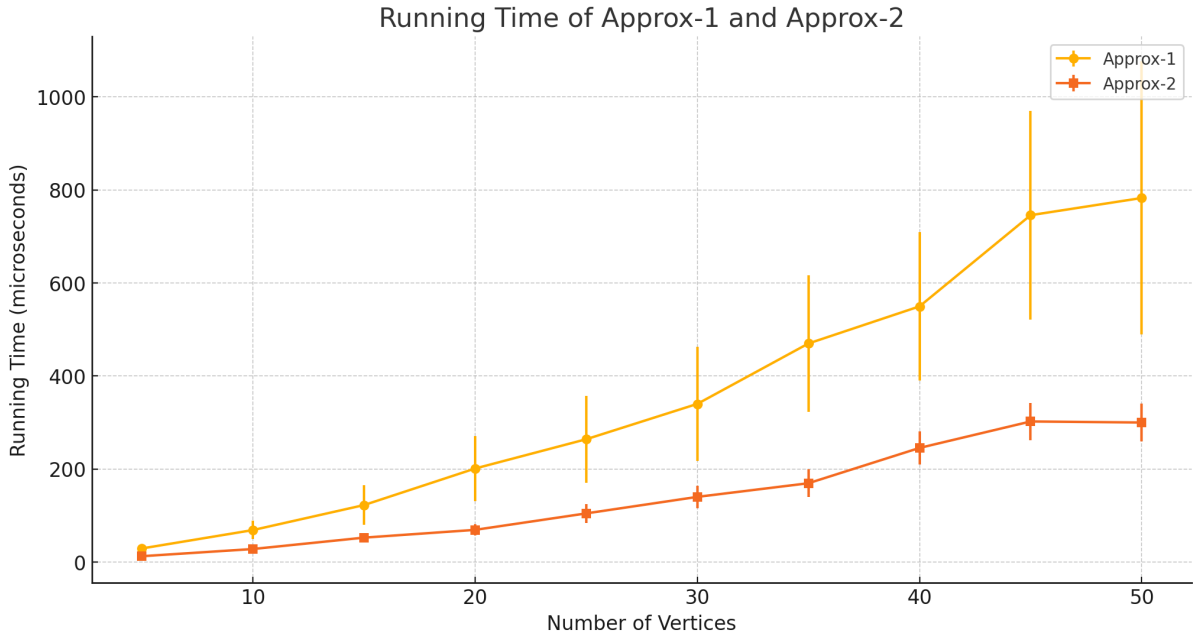


**Figure 2:** Running time of APPROX-VC-1 and APPROX-VC-2.

## 4.2 Approximation Ratio

In Figure 3, the approximation ratio is plotted against the number of vertices V, where the y-axis represents the approximation ratio and the x-axis represents the vertex count. In this

analysis, **CNF-SAT-VC** is considered the optimal solution, and the approximation ratio is defined by comparing the vertex cover sizes obtained from **Approx-VC-1** and **Approx-VC-2** with the optimal results provided by **CNF-SAT-VC**.

**Approx-VC-1** consistently delivers an approximation ratio close to 1, indicating that its heuristic approach effectively approximates the minimum vertex cover. In contrast, **Approx-VC-2** exhibits a significantly higher approximation ratio, reflecting its less precise heuristic. By selecting arbitrary vertices, Approx-VC-2 often results in a vertex cover larger than the optimal. This disparity becomes more pronounced as the graph size increases, further highlighting its limitations in approximating the minimum vertex cover.

Overall, Approx-VC-1 demonstrates superior performance in terms of approximation ratio, making it the preferred method for scenarios where solution quality is crucial while balancing computational efficiency.

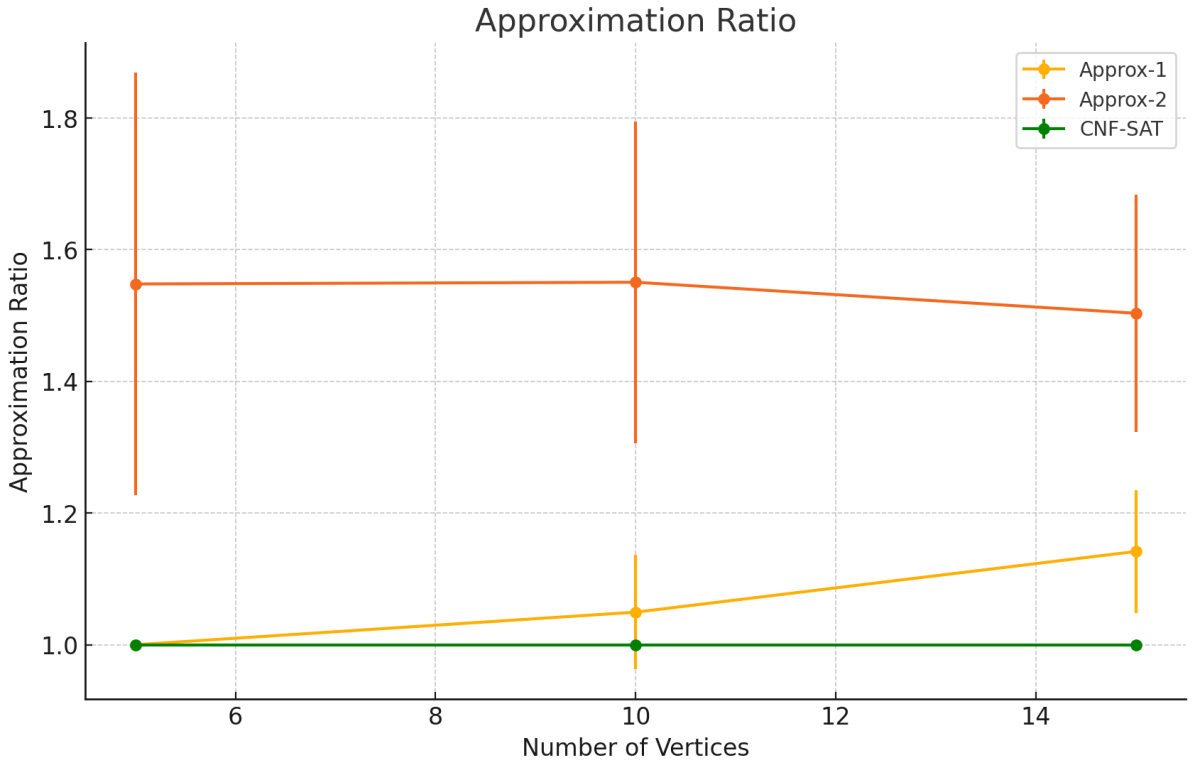


**Figure 3:** Approximation Ratio.

# 5 Conclusion

This project analyzed three algorithms—**CNF-SAT**, **Approx-VC-1**, and **Approx-VC-2**—for solving the vertex cover problem, with a focus on their running times and approximation ratios.

- **CNF-SAT** guarantees the optimal solution but exhibits exponential running time as the graph size increases, making it impractical for larger datasets.
- **Approx-VC-1** provides a near-optimal solution with significantly reduced computational complexity, making it the most balanced algorithm for practical scenarios.
- **Approx-VC-2** is the fastest of the three, but its approximation ratio is suboptimal, limiting its utility in cases where solution accuracy is critical.

For graphs with fewer vertices V, **CNF-SAT** is the ideal choice, offering exact solutions. For larger graphs, **Approx-VC-1** strikes the best balance between efficiency and accuracy. On the other hand, **Approx-VC-2** is suitable for time-sensitive applications where precision is less important.

In conclusion, the choice of algorithm depends on the specific requirements of the application, with Approx-VC-1 being the preferred heuristic in most practical scenarios.