# Efficiency Analysis of Various Vertex Cover Algorithms

**ECE 650: Final Project**


**Submitted By:**
**Abhi Zanzarukiya - 20892635**
**Heli Mistry - 20985952**

**Electrical and Computer Engineering**
**University of Waterloo**

**Abstract**

It is a typical NP-hard optimization problem in computer science to determine a minimum vertex cover. As a result, we have no polynomial time algorithm for solving this problem, and we use approximation algorithms instead. This project report shows an analysis of three different approaches to solve the Vertex Cover problem viz.(1) CNF-SAT-VC (2) APPROX-VC-1 (3) APPROX-VC-2. Here we analyze the efficiency of the algorithms by considering a comparison of their running times and approximation ratios. Looking at the results, we observed that CNF-SAT-VC produces the optimal solution, but comparatively has the largest runtime (exponential in worst case). APPROX-VC-2 has the shortest runtime but does not provide a minimum vertex coverage, whereas APPROX-VC-1 is the most feasible in terms of accuracy and efficiency, despite a slight deviation from providing an optimal solution.

# 1 Overview

This project aims to solve the problem of installation of minimum number of security cameras at traffic intersections to help the police department. We map this optimization problem to the Vertex Cover problem and present three different possible solutions with their analysis.

# 2 Definitions

## 2.1 Vertex Cover

A vertex cover of a graph G = (V, E) is a subset of vertices $C \subseteq$V, such that each edge $e \in$E is incident to at least one vertex in C. In other words, the vertex cover is a set of vertices that completely covers all edges of a given graph.

## 2.2 MiniSAT

SAT solvers are tools that are used to check and evaluate the satisfiability of Boolean Satisfiability problems [3]. The evaluated satisfying assignment from the SAT solver is the solution to an NP-complete problem which is represented as a Boolean logic formula.

MiniSAT is a minimalistic SAT solver that we have used in our project. This tool is highly modifiable, and through variable and clause elimination it is very efficient [2].

## 2.3 CNF-SAT

The CNF Satisfiability Problem (CNF-SAT) is a version of the classic combinatorial Satisfiability Problem - Boolean Satisfiability, where the propositional logic is specified in the Conjunctive Normal Form (CNF). In this representation, a given formula of variables is written as a conjunction of clauses. Each clause is a disjunction of literals, and a literal is a variable or its negation [1].

We reduce the vertex cover problem for graph G, to CNF-SAT formula F with four reductions, in polynomial time [1]. This formula F is provided as an input to the SAT-solver MiniSAT. If F is satisfiable, then the satisfiability assignment from the MiniSAT will be our minimum vertex cover.

## 2.4 APPROX-VC-1

This greedy approach aims at finding the vertex cover by picking up a vertex with the highest degree in each run and eliminates all its adjacent edges. The process is repeated until all the edges are covered.

**Algorithm 1:** APPROX-VC-1(G)

$C \leftarrow \Phi$;
**while** $E \neq \Phi$ **do**
    Find vertex v from V with highest number of incident edges;
    $C \leftarrow C \cup \{v\}$;
    Delete all edges incident to v;
**end**
return C;

### 2.5 APPROX-VC-2

This is a naive algorithm, where we pick an arbitrary edge and insert both its endpoints in the vertex cover. Then, we throw away all the edges attached to these two endpoints and keep going till no uncovered edges are left.

**Algorithm 2:** APPROX-VC-2(G)

$C \leftarrow \Phi$;
**while** $E \neq \Phi$ **do**
    Pick an edge $\{u, v\} \in E$;
    $C \leftarrow C \cup \{u, v\}$;
    Delete all edges incident to either u or v;
**end**
return C;

### 2.6 Efficiency Metrics

As part of this experiment, the efficiency of all three algorithms are evaluated by using two quantitative metrics, namely: Mean Running Time and Approximation Ratio.

#### 2.6.1 Mean Running Time

An algorithm's mean runtime represents the average time taken to solve a problem by the algorithm across multiple runs. The three algorithms we use, APPROX-VC-1, APPROX-VC-2, and CNF-SAT-VC, are run over multiple inputs of a graph of fixed size, and the average time is gathered.

#### 2.6.2 Approximation Ratio

The approximation ratio here is defined as the ratio of cardinality of a vertex cover generated by various approximate algorithms (APPROX-VC-1 and APPROX-VC-2), to the cardinality of vertex cover by optimal (minimum-size) algorithm CNF-SAT-VC.

## 3 Implementation details

The concept of multithreading is used for this experiment to run algorithms concurrently. In total four threads are used to perform the analysis. One thread (main) is used for IO operations like reading the input, parsing it and then creating and calling the other three threads. These three threads run their corresponding algorithms over the parsed input, and generates the result. The required methods for dealing with threads are available in the pthread and time library in C++, which are used here. Methods like `pthread_getcpuclockid()` and `clock_gettime()` are used to get the clock id of the CPU assigned to the thread and extract the instance of time from that clock respectively.

Here for this experiment we selected 23 different sizes of vertex $|\boldsymbol{V}|$ ranging from [2, 17] in increments of 1 and [20, 50] with increments of 5. Now for each vertex size $|\boldsymbol{V}|$, we generate 10 graphs of that size using ***graphGen*** - a random generator for generating graphs. Each such graph is run

10 times against all three algorithms. This results in a total 100 runs of 10 graphs for a given vertex size.

The output data generated by these runs is of the order (100 x 6), where each row value represents the output from one run and 6 columns represents different metrics computed in form of tuple

*(RT(CNF-SAT-VC), RT(APPROX-VC-1),RT(APPROX-VC-2), VCSize(APPROX-VC-1), VCSize(APPROX-VC-2), VCSize(CNF-SAT-VC))*

*where RT(A) is running time of Algorithm A in microseconds and VCSize(A) is the vertex cover size produced by Algorithm A.*

When the results are generated for each input vertex size, the average and standard deviation of running time for all three algorithms are calculated. Moreover, we also calculate the approximation ratio of APPROX-VC-1 and APPROX-VC-2 for efficiency analysis.

# 4 Results and Analysis

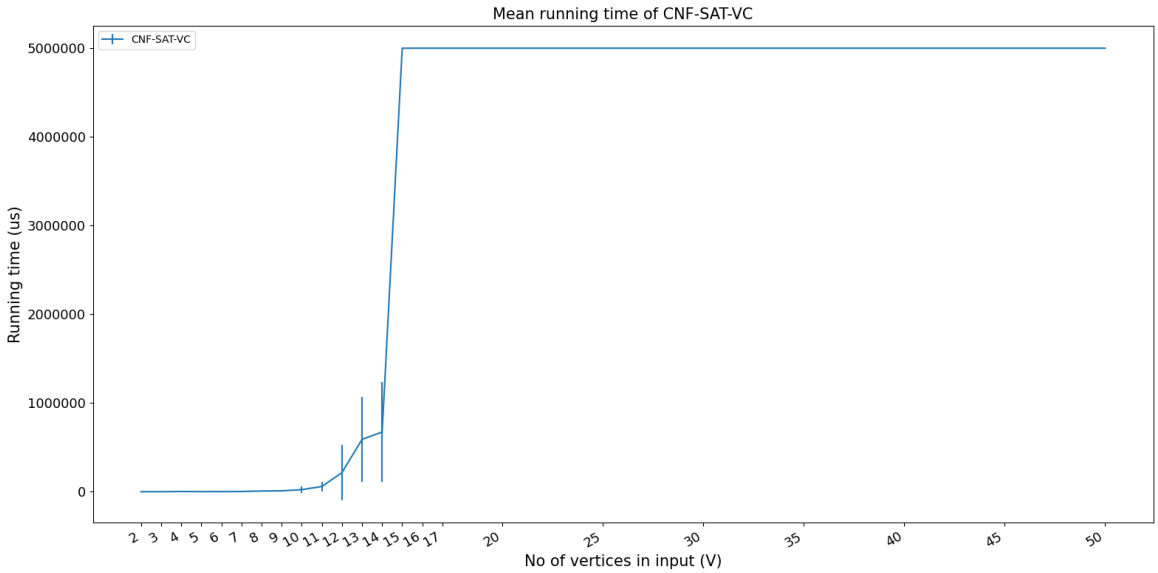


Figure 1: Mean Running Time plot for all input vertices of CNF-SAT-VC algorithm thread.

The average running time of CNF-SAT-VC algorithm is shown in ***Figure*1**, which contains all the values of vertex size used as part of analysis. It can be seen that with the increase in the number of vertices, the time taken by CNF-SAT-VC to generate the solution increases exponentially. The time taken by graphs with vertices of size $|\boldsymbol{V}| \geq 15$ is so high that in order to have the feasibility of the experiment, the timeout of 5 seconds is associated with the CNF-SAT-VC thread. As a result, a flat line ranging for vertex input from [15, 50] is seen to be clipped at 5 seconds because of timeout. In order to have the proper resolution of exponential behavior for runtime of this algorithm, ***Figure*2** is plotted with the range of vertices [2,14] where the algorithm is successful in finding a solution before timeout.

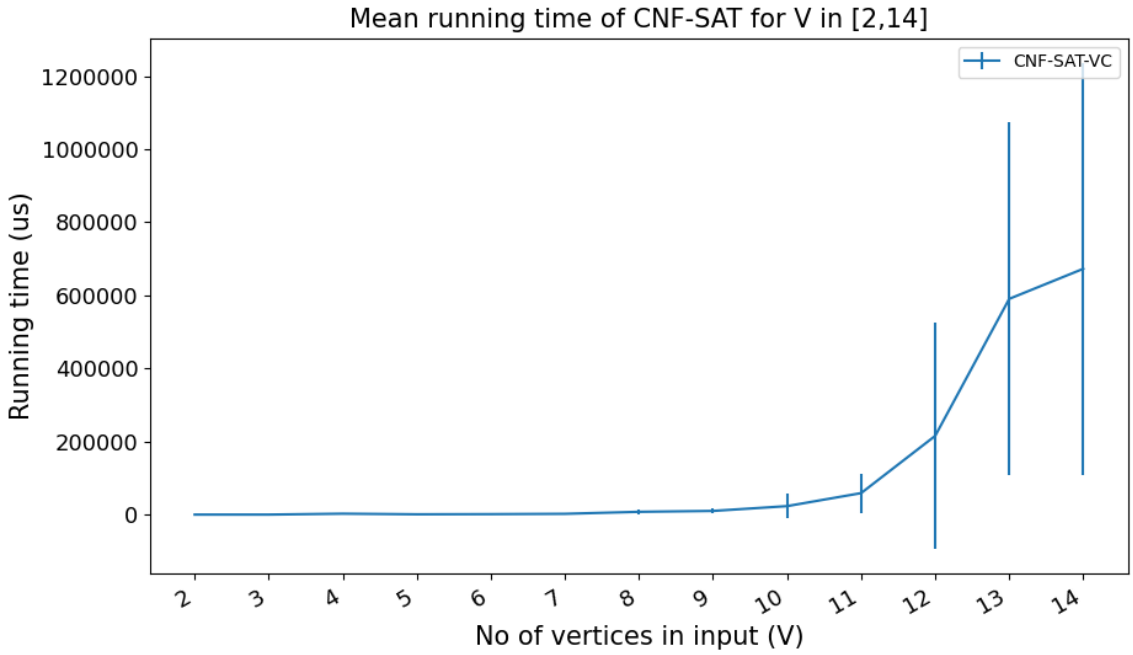


Figure 2: Mean Running Time plot for vertices in range [2,14] of CNF-SAT-VC algorithm thread.

The reason for the exponential rise in time can be justified by looking at the formula stating number of clauses and literals involved in reduction from VC to CNF-SAT which is given by:

$$Numberofreductions : k + k * \binom{V}{2} + V * \binom{k}{2} + |E|$$

$$Numberofliterals : V * k$$

*where V is the input vertex size, k is the vertex cover size and $|E|$ is the number of edges.[1]*

As V increases, the literals and the clauses generated by reduction also increase exponentially. This in turn makes the propositional logic formula F also large. SAT solver (here MiniSat) needs a lot of time to generate a satisfiable assignment for such a formula in order to solve the problem of vertex cover.

The ***Figure3*** below, shows the average running time of both approximate algorithms, APPROX-VC-1 and APPROX-VC-2 against all input vertices. It is very clear that the runtime of both these algorithms is very less compared to that of CNF-SAT-VC algorithm. Till input in range [2,9] the average runtime is almost equal for both algorithms. However, from $|\boldsymbol{V}| \geq 10$, the graph shows divergence and hence it can be observed that APPROX-VC-2 takes less time compared to APPROX-VC-1 in this range.

In APPROX-VC-2 the vertices associated with the first edge in edge list is added to vertex cover list and then the edge list is traversed to remove edges having these vertices. Whereas APPROX-VC-1 visits the edge list to find the highest degree vertex in order to include it in the vertex cover list. After that the list is traversed to remove edges having that highest degree vertex. So, APPROX-VC-1 makes extra iteration over the edge list which increases the time for computing the solution.

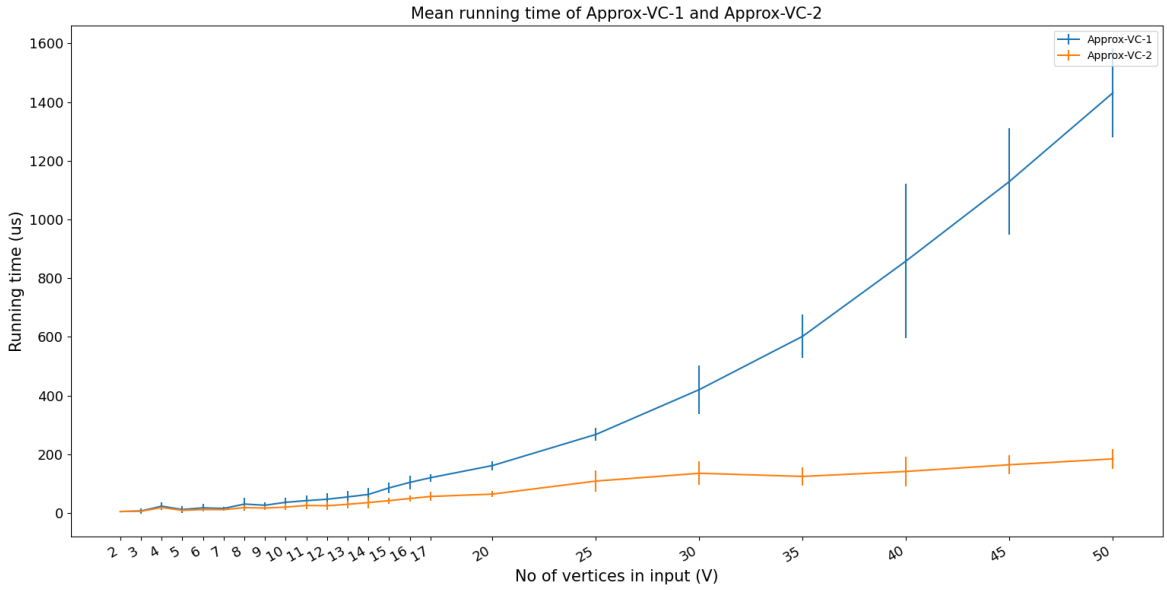


Figure 3: Mean Running Time plot for all input vertices APPROX-VC-1 and APPROX-VC-2 algorithm thread.

The mean approximation ratio for both APPROX-VC-1 and APPROX-VC-2 is plotted in ***Figure4***. Here the range of input vertices [2,14] is used, because for $|\boldsymbol{V}| \geq 15$, CNF-SAT-VC timeouts and as a result size of optimal vertex cover is not obtained resulting in no calculation of Approximation Ratio. It is evident from the result that the accuracy in the minimal size of vertex cover is more for APPROX-VC-1 than its counterpart APPROX-VC-2. In fact, the output of cardinality of vertex cover for APPROX-VC-1 is almost equal to that of the optimal CNF-SAT-VC algorithm. This can be deduced by looking at Mean Approximation Ratio for APPROX-VC-1 which is almost equal to 1, with every less deviation over given input range.

The Mean Approximation Ratio for APPROX-VC-2 is around 1.5 which suggests that vertices in vertex cover produced by APPROX-VC-2 is significantly more than CNF-SAT-VC. This is due to the fact that this naive algorithm does not pick a vertex that covers maximum edges of the graph, which results in high deviation from optimal size of vertex cover.

The APPROX-VC-1 chooses the vertex with the highest degree as vertex cover, hence more number of edges connected with that vertex gets covered. Thus output size is almost same as that of optimal one and also deviation is less which can be seen by small or almost no errorbar for APPROX-VC-1.

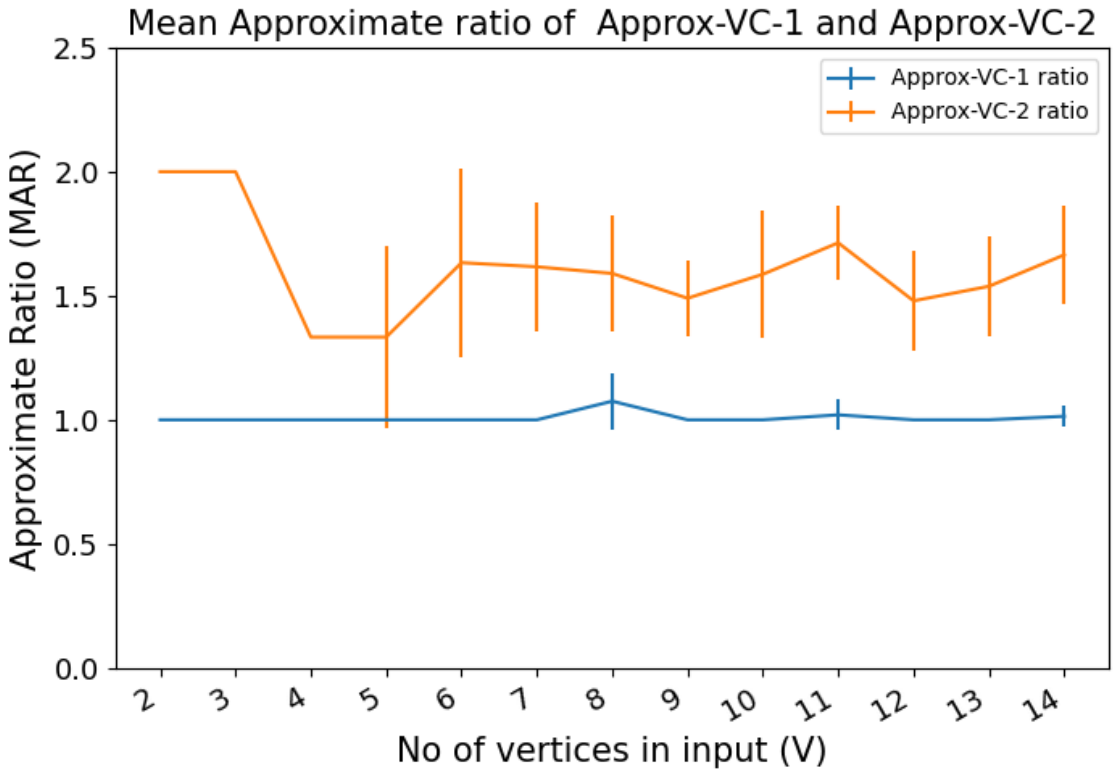


Figure 4: Mean Approximation Ratio for vertices in range [2,14] for APPROX-VC-1 and APPROX-VC-2.

## 5 Conclusion

The CNF-SAT-VC algorithm produces the optimal solution for the minimum-sized vertex cover problem. But, from analysis, it's clear that this algorithm's running time increases exponentially with the increase in vertices size and as a result, solutions cannot be obtained in a reasonable amount of time for large input graphs. APPROX-AC-V2 runs fastest among all three algorithms but this speed comes at the cost of accuracy, as the output of APPROX-VC-2 deviates much from the optimal output.

Time and accuracy are both important factors, in which case the APPROX-VC-1 algorithm is the best option as its output is close to the minimum vertex cover size. Furthermore, the algorithm runs in polynomial time and produces results much faster than CNF-SAT-VC, even for large input graphs.

## References

[1] Arie Gurfinkel. Assignment 4: Methods and tools for software engineering. https://git.uwaterloo.ca/ece650-1221/pdfs/-/blob/master/uw-ece650-1221-a4.pdf, February 2022.

[2] Niklas Sörensson Niklas Eén. The minisat page. http://minisat.se/.

[3] Atri Rudra Sanchit Batra. Satisfiability and sat solvers. https://cse.buffalo.edu/~erdem/cse331/support/sat-solver/index.html.