ECE 650: Methods & Tools for Software Engineering

Final Project Report

Submitted by:

Allan Luu (a22luu) - 21051531

Zhen Zhang (z89zhang) - 21051768

2023-12-05

1 Introduction

This project employs a set of three algorithms to solve the minimum vertex cover problem. The initial approach is the REDUCTION-TO-CNF-SAT (referred to CNF-SAT-VC in the following report), reduction from VERTEX-COVER to CNF-SAT and solving by MiniSAT solver. The other two algorithms, identified as APPROX-VC-1 and APPROX-VC-2, will be explained in the following sections, accompanied by an examination of runtime and the approximation ratio.

2 Algorithms

2.1 CNF-SAT-VC

The CNF-SAT-VC algorithm was applied in assignment 4, operates by taking a reduction from vertex cover graph to propositional logic formula. Given inputs G and k generates a formula F in Conjunctive Normal Form (CNF) format. This CNF-formatted formula is then used as input for MiniSAT, a SAT solver. MiniSAT calculates the satisfiability of formula F, determining whether there exists a satisfying assignment for the given inputs within the context of the vertex cover problem.

2.2 APPROX-VC-1

This APPROX-VC-1 is a greedy algorithm that selects the vertex with the highest degree, adds it to the cover set, and subsequently removes both the vertex and all related edges from the original list. This process continues until there are no remaining edges.

Algorithm 1 APPROX-VC-1

```
1: let $R = \emptyset$
2: let $E' = E$
3: while $E' \neq \emptyset$ do
4:     find V with highest degree
5:     $R \leftarrow R \cup \{v\}$
6:     remove edges incident to V
7: end while
8: return $R$
```

2.3 APPROX-VC-2

Similar to APPROX-VC-1, the APPROX-VC-2 algorithm diverges in the selection approach. Instead of choosing the highest degree vertex, it randomly selects an edge and add the vertices $< u, v >$ to the set. Subsequently, all edges incident to these two vertices are removed, and continue the process until no remaining edges left.

Algorithm 2 APPROX-VC-2

```
1: let $R = \emptyset$
2: let $E' = E$
3: while $E' \neq \emptyset$ do
4:     randomly choose an edge $\{u, v\} \in E'$
5:     $R \leftarrow R \cup \{u, v\}$
6:     for Edge in Graph do
7:         if {u,v} in Edge then
8:             remove the edge
9:         end if
10:    end for
11: end while
12: return $R$
```

3 Implementation

The input data is generated using the **graphGen** program. This program takes the number of vertices as input and produces results in the format V x and E $< y, z >$, ... In this context, x represents the number of vertices in the graph, and $< y, z >$ denotes two vertices for that edge.

The **run.sh** script located in the current folder is designed to automatically execute the graphGen program and store the results in the data folder. This Bash file generates vertices ranging from 5 to 50, i.e., $|V| \in [5, 50]$, with an increment of 5. For each vertex, 10 graphs are generated, and measurements are conducted 10 times for each graph.

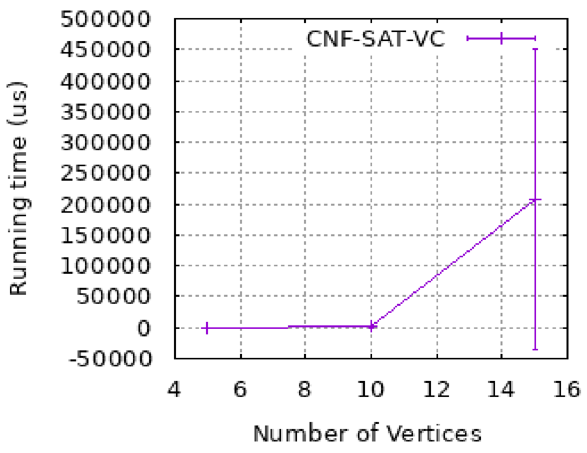

Figure 1: Mean computation time with error bar for CNF-SAT-VC algorithm thread.

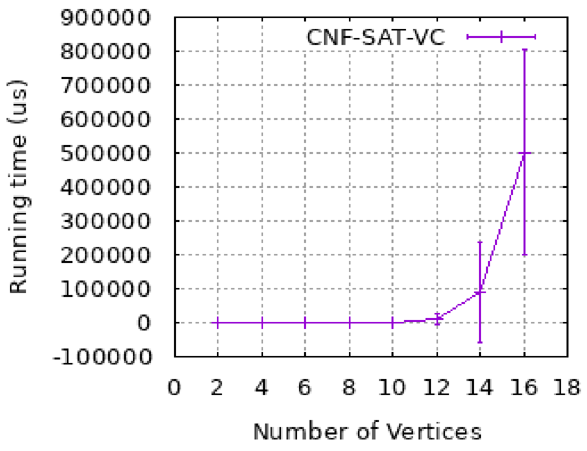

Figure 2: CNF-SAT-VC mean computation time in range [2,16]

4 Results and Analysis

4.1 Running time

As one of the methods to analyze the efficiency of algorithms, the mean (average) computation time and standard deviation are calculated using system clock across different threads for each algorithm. The Figure 1 illustrates the average running time for the graph with vertex number ranging from 5 to 15 in increments of 5 in CNF-SAT-VC. The running time shows a rapid increase from vertex number 10 to vertex number 15. According to the provided documentation *a4_encoding*, the number of clauses in the reduction is expressed as $k + n \cdot \binom{k}{2} + k \cdot \binom{n}{2} + |E|$ which exponentially increasing time complexity.

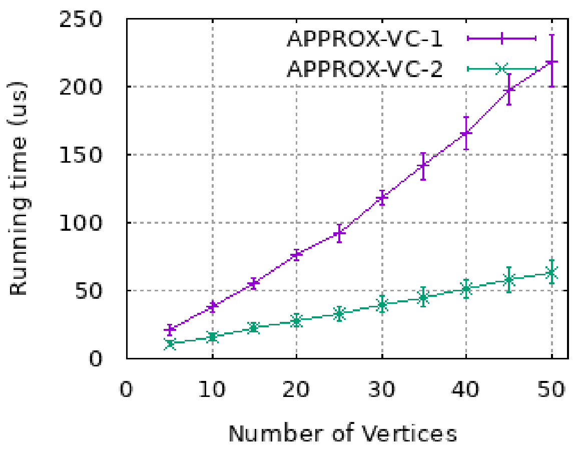

Figure 3: APPROX-VC-1 & APPROX-VC-2 mean computation time in range [5,50]

To manage this, the timeout mechanism was implemented associated with the thread, which is predominantly triggered when $|V| \geq 17$. In order to enhance the analysis of CNF-SAT-VC efficiency, the collected data reduced the number of vertices from 2 to 16 with an increment of 2 for a more comprehensive examination. As shown in Figure 2, the mean time exponential growth similar to Figure 1. Additionally, the error bars significantly increase along the x-axis refers the varied calculation times for different graphs even with the same number of vertices.

The Figure 3 above shows the average computation times for APPROX-VC-1 and APPROX-VC-2, and the computation times are significantly lower than those in CNF-SAT-VC. In comparison, APPROX-VC-1 takes more time because of the selection of the highest degree vertex, while APPROX-VC-2 simply employs a random edge selection strategy. The running time increases linearly for both algorithms, and the error bar is inconspicuous due to the small standard deviation.

4.2 Approximation ratio

The approximation ratio is as another metric for analyzing algorithms, providing a basis for comparison with the output of CNF-SAT-VC, which is assured to be optimal. The formula is given by

$$ApproxRatio = \frac{Approximate}{Optimal}$$

. As shown in Figure 4, for vertex number 5, and 10, the output of APPROX-VC-1 are exactly the same as optimal algorithm with nearly imperceptible error bars. However, the fastest algorithm, APPROX-VC-2, performs poorly with the optimal solution.

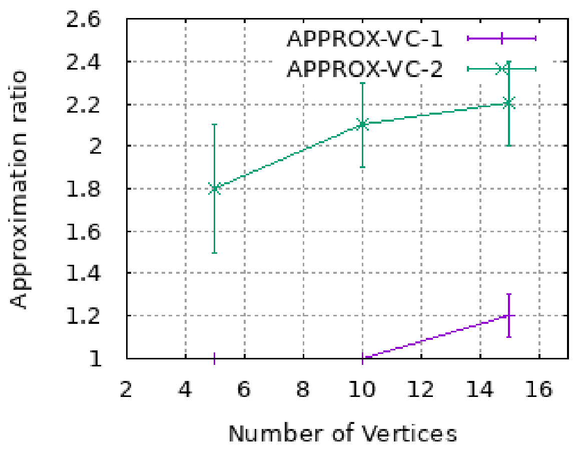

Figure 4: Approximation Ratio

5 Conclusion

The three algorithms shows a varied performance on the same hardware device with the same input. Specifically, the CNF-SAT-VC algorithm experiences an exponential increase in running time as the number of vertices rises, and APPROX-VC-2 has the fastest run time, albeit with a trade-off between running time and accuracy. APPROX-VC-1 maintains a linear running time as APPROX-VC-1, and its accuracy rate closely to CNF-SAT-VC.