ECE650 PROJECT REPORT

Yingru Pan 21063619 Xiaomeng Su 21118864

November 2024

1 Introduction

A vertex cover of a graph $G = (V, E)$ is a subset of vertices $C \subseteq V$, such that each edge in E has at least one vertex in C.

There exist a minimum vertex cover for an non-empty graph. Define the approximation of a vertex cover as the size of itself divided by the size of minimum vertex cover.

In this report, we are going to solve Vertex Cover Problem in three different algorithms, and analysis the performance of each algorithm, including time-efficiency and approximation ratio of the results.

2 Implement of Algorithms

2.1 CNF-SAT

This algorithm could be used for looking for a minimum vertex cover. Initialize an integer $k = 1$, increment the value of k by 1 at a time, and observe if there exist a vertex cover of size k.

A SAT-Solver is set to check if there is a vertex cover of size k. Detailed implementation of the algorithm is same as Assignment 4.[1]

2.2 APPROX-VC-1

This algorithm add the vertex with highest degree to the vertex cover, and delete all edges incident on it, repeat the operation till no edges remain.[2]

Assume each edge is encoded by a structure consist of two vertices u, v, the input of function should be number of vertices V, and the list E of all edges in G.

Algorithm 1 APPROX-VC-1

```
1: function VC_1(V, E)
2:     int result ← ∅
3:     int d[V]
4:     while true do
5:         for int j from 0 to V − 1 do
6:             [j] ← 0
7:         for i in E do
8:             d[i.u] ← d[i.u] + 1
9:             d[i.v] ← d[i.v] + 1
10:        int max ← index of maximum element in d
11:        if d[max] = 0 then
12:            break
13:        for e in E do
14:            if e.u = d[max] or e.v = d[max] then delete e from E
15:        result ← result ∪ {max}
16:    return result
```

2.3 APPROX-VC-2

This algorithm randomly pick an edge $< u, v >$ from E, add both u and v to the vertex cover, and delete all edges incident on u or v. Repeat until no edges remain.[2]

Algorithm 2 APPROX-VC-2

```
1: function VC_2(V, E)
2:     int result ← ∅
3:     while true do
4:         randomly pick x from E
5:         result ← result ∪ {x.u, x.v}
6:         for e in E do
7:             if e.u = x.u or e.u = x.v or e.v = x.u or e.v = x.v then
8:                 delete e from E
9:     return result
```

3 Performance and Analysis

3.1 Time-efficiency

To analyze the running time of three algorithms independently, three threads are created to run three algorithms simultaneously. Use pthread_getcpuclockid() and clock_gettime() to measure running time of each thread.

The graphs of running time analysis are presented as follows, where the y-axis represents the running time, the x-axis represents the number of vertices $|V|$, and the errorbars indicate the standard deviation.

Figure 1 shows the running time of CNF-SAT. The graph clearly demonstrate the exponentially increase of running time with the increase of number of vertices. When there are fewer than 10 vertices, the running time is noticeably short. However, when $|V|$ reaches 15, the running time increase significantly, and the standard deviation also becomes larger.

The main reason for this is that, as the number of vertices increases, the SAT-Solver needs to process a larger value of k, which means it has to handle more literals and clauses. More specifically, if k increases by 1, according to four rules of reduction[1], number of clauses is going to increase by $|V|^2/2 - |V|/2 + |V|k + 1$. This significantly increases the complexity of finding a vertex cover, resulting in longer running times and greater standard deviation.

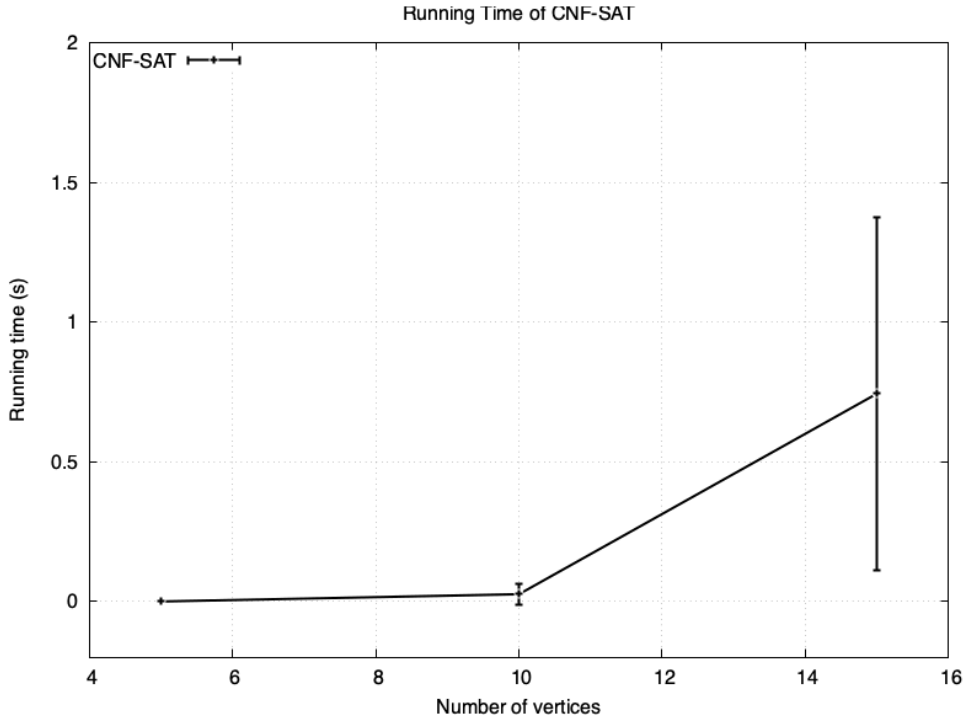

Figure 1: running time of CNF-SAT

Figure 2 compares the running time between APPROX-VC-1 and APPROX-VC-2 when the number of vertices rise from 5 to 50. These two algorithms both have an upward trend in running time as the number of vertices increase. APPROX-VC-1 performs slightly slower than APPROX-VC-2 when there are 5 to 15 vertices. However, as the number of vertices exceeds 15, the running time for both algorithms begins to diverge, with APPROX-VC-1 showing a more significant increase and larger standard deviation compared to APPROX-VC-2.

The primary distinction between these two algorithms lies in their approach to selecting vertices. APPROX-VC-1 applies an additional greedy choice strategy, by selecting the vertex with the highest degree in each iteration. In contrast, APPROX-VC-2 takes a simpler approach by randomly choosing a vertex in each iteration. This difference accounts for the longer running time of APPROX-VC-1.

The smaller standard deviation of APPROX-VC-2 can be attributed to

its simpler, randomized approach to selecting edges. Its running time is less sensitive to variations in graph structure. This reduces the variability in performance across different inputs, leading to a smaller standard deviation. In contrast, APPROX-VC-1 relies on a greedy choice strategy that involves additional computations to evaluate and sort vertex degrees at each step.

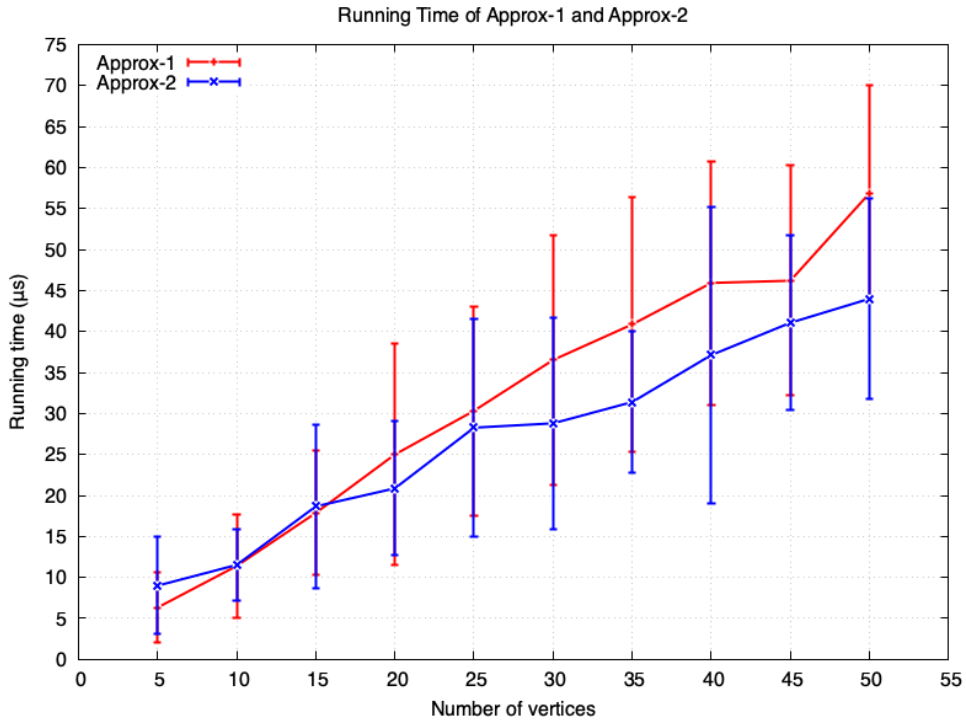

Figure 2: running time of APPROX-VC-1 and APPROX-VC-2

3.2 Approximation Ratio

To calculate the approximation ratio for APPROX-VC-1 and APPROX-VC-2, we compare their outputs against those generated by CNF-SAT, which guarantees the optimal vertex cover. We demonstrate the comparison in Figure 3, where the y-axis represents the approximation ratio, the x-axis represents the number of vertices $|V|$, and the errorbars indicate the standard deviation.

It is clear that APPROX-VC-1 have better performance in terms of approximation ratio. The outputs generated by APPROX-VC-1 is quite close to the optimal vertex cover, with a small standard deviation as well. However, the approximation ratio of APPROX-VC-2 deviates significantly from the optimal solution, and its standard deviation is also much larger.

The reason for better performance of APPROX-VC-1 than APPROX-VC-2 can be attributed to the distinct approaches of the two algorithms. APPROX-VC-1 selects the vertex with the highest degree in each iteration, ensuring that the selected vertex is a relatively optimal choice at that iteration. While the greedy choice strategy does not always guarantee the globally optimal solution, it can also deliver a good performance. However, APPROX-VC-2 selects vertices randomly in each iteration, which does not contribute meaningfully to improving the approximation quality.

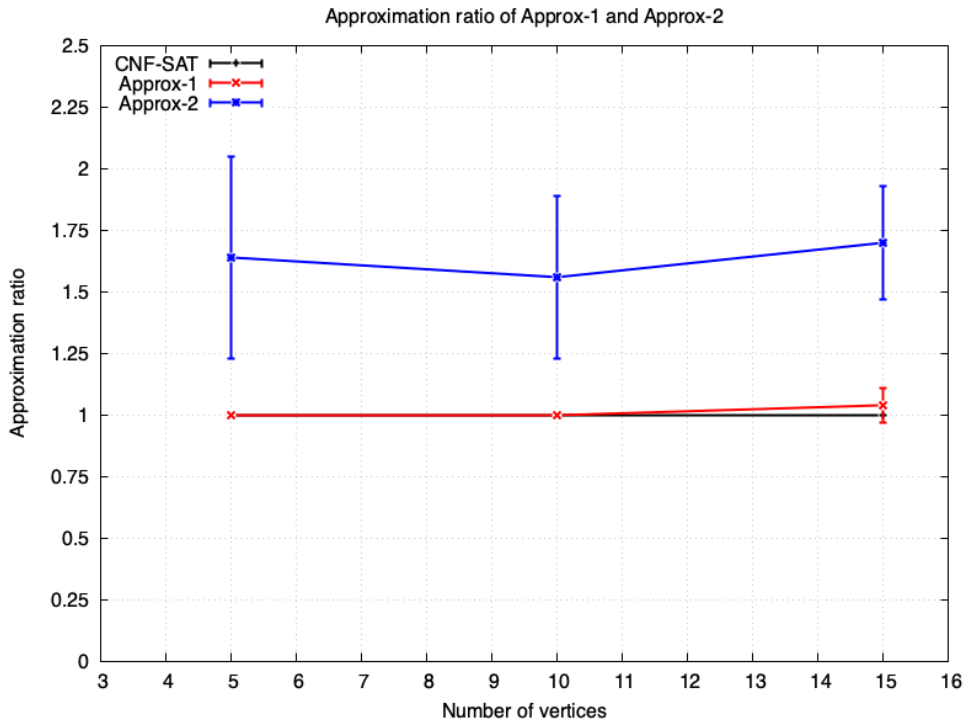

Figure 3: Approximation ratio of APPROX-VC-1 and APPROX-VC-2

4 Conclusion

CNF-SAT guarantees to generate the optimal vertex cover, serving as the benchmark for evaluating the approximation of other vertex cover algorithms. However, this precision comes at the cost of significantly longer computation times. To prevent excessively long runtime, we set a timeout limit of 10 seconds for each graph. We observe that once the number of vertices reaches 15, the timeout is frequently triggered. When the number of vertices is equal to or greater than 20, the algorithm consistently exceeds the timeout limit. Thus, while CNF-SAT is highly accurate, its efficiency is relatively low, especially for larger graphs.

APPROX-VC-1 demonstrates a relatively balanced performance. Its running time is significantly shorter than that of CNF-SAT, while its approximation is notably better than that of APPROX-VC-2. Therefore, if a relatively accurate minimum vertex cover result is needed without spending excessive computation time, APPROX-VC-1 is a suitable choice. The main advantage of APPROX-VC-2 lies in its minimal computation time. However, it can only provide a vertex cover that is highly unlikely to be the optimal solution, especially in graphs with a large number of vertices.

References

[1] Albert Wasef. *ECE 650, Fall 2024, Section 01: A Polynomial-Time Reduction from VERTEX-COVER to CNF-SAT*. `https://git.uwaterloo.ca/ece650-f24/assignments_pdf/-/raw/master/a4_encoding.pdf`, October 2024.

[2] Albert Wasef. *Methods and Tools for Software Engineering: Final Course Project*. `https://git.uwaterloo.ca/ece650-f24/assignments_pdf/-/raw/master/project.pdf`, November 2024.