ECE650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

Final Project: Effieciency and Accuracy Analysis of Different Algorithms in Solving the Vertex Problem

Authored by:

1. Minh-Quan Quach (ID: 21050664)
2. Bethelhem Nibret (ID:20979837)

I Introduction

The primary objective of this project is to optimize the installation of CCTV cameras at street intersections, with the aim of reducing the number of cameras required while maintaining effective monitoring. To address this real-world challenge, we propose mapping the problem to the Vertex Cover problem. By formulating the problem in this way, we can leverage existing theoretical and practical solutions to identify the optimal locations for cameras, leading to more efficient and cost-effective installation strategies for police departments.

The Minimum Vertex Cover problem, which is known to be NP-complete, involves finding the smallest possible subset of vertices in an undirected graph G(V,E) such that for every edge (u,v) in the graph, at least one of the vertices 'u' or 'v' is included in the vertex cover. This report presents six different methods for solving this problem, and analyzes each approach in terms of its running time and approximation ratio. The second part of the report provides an overview of these methods, while the next one will offer the implementation of the program which is used for running the algorithms. a detailed comparison of their performance will be in the next section.

II Algorithms

1 CNF-SAT

The CNF Satisfiability problem (CNF-SAT) is a problem where Boolean formula is given in the Conjuntive Normal Form (CNF). As in the assignment 4, we reduce the graph G to CNF formula F by using 4 reductions in the polynomial time, and use this formula as a input to SAT Solver to determine its satisfiability. If F is satisfiable, the assignments are the Vertex Cover set.

Algorithm 1 Finding vertex cover using CNF-SAT

```
function CNF-SAT(Graph $G = (V, E)$, integer $k$)
    Initialize a SAT solver $S$
    /* Step 1: */
    for $i = 1$ to $k$ do
        Add clause $\bigvee_{v \in V} x_{v,i}$ to $S$
    end for
    /* Step 2: */
    for $i = 1$ to $n$ and $p, q = 1$ to $k$, $p < q$ do
        Add clauses $(\neg x_{i,p} \vee \neg x_{i,q})$ to $S$
    end for
    /* Step 3: */
    for $i = 1$ to $k$ and $p, q = 1$ to $n$, $p < q$ do
        Add clauses $(\neg x_{p,i} \vee \neg x_{q,i})$ to $S$
    end for
    /*Step 4*/
    for $(u, v) \in E$ and $i = 1$ to $k$ do
        Add clause $(x_{u,i} \vee x_{v,i})$ to $S$
    end for
    Use a SAT solver to solve $S$
    if Solution found then
        Extract the vertices assigned true
```

This approach will use k + nk2 + kn2 + /E/ clauses

2 3-CNF-SAT

The 3-CNF-SAT problem is the special case of SAT, where each clause is a conjunction of three literals.

Algorithm 2 Finding vertex cover using 3-CNF-SAT

```
function 3-CNF-SAT(Graph $G = (V, E)$, integer $k$)
    Initialize a SAT solver $S$
    for $i = 1$ to $k$ do
        if $|C| \leq 3$ then
            Add clause $\bigvee_{v \in V} x_{v,i}$ to $S$
        else
            Create auxiliary variables $B_0, B_1, \ldots, B_{|C|-2}$
            for $i = 1$ to $|C| - 2$ do
                Add clause $\neg B_{i-1} \vee C_i \vee B_i$ to $S$
            end for
            Add clause $\neg B_{|C|-2} \vee C_{|C|}$ to $S$
        end if
    end for
    /*same as Step 2*/
    /*same as Step 3*/
    for $(u, v) \in E$ and $i = 1$ to $k$ do
        if $|C| \leq 3$ then
            Add clause $(x_{u,i} \vee x_{v,i})$ to $S$
        else
            Create auxiliary variables $B_0, B_1, \ldots, B_{|C|-2}$
            for $i = 1$ to $|C| - 2$ do
                Add clause $\neg B_{i-1} \vee C_i \vee B_i$ to $S$
            end for
            Add clause $\neg B_{|C|-2} \vee C_{|C|}$ to $S$
        end if
    end for
    Use a SAT solver to solve $S$
    if Solution found then
        Extract the literals assigned true
```

3 ARRPOX-VC-1

The pseudo code for ARRPOX-VC-1

Algorithm 3 APPROX-VC-1: Greedy algorithm for vertex cover

```
function APPROX-VC-1($G = (V, E)$)
    $C \leftarrow \emptyset$
    $E' \leftarrow E$
    while $E' \neq \emptyset$ do
        Choose a vertex $v$ of maximum degree in $G$
        $C \leftarrow C \cup v$
        Remove all edges incident to $v$ from $E'$
    end while
    return $C$
end function=0
```

4 ARRPOX-VC-2

The pseudo code for ARRPOX-VC-2

Algorithm 4 APPROX-VC-2: Greedy algorithm for vertex cover

```
function APPROX-VC-2($G = (V, E)$)
    $C \leftarrow \emptyset$
    $E' \leftarrow E$
    while $E' \neq \emptyset$ do
        Choose an edge $e = \langle u, v \rangle$ from $E'$
        $C \leftarrow C \cup u, v$
        Remove all edges incident to $u$ or $v$ from $E'$
    end while
    return $C$
end function
```

5 REFINED-APPROX-VC-1 and 2

Algorithm 5 REFINED-APPROX-VC-1 and 2: Greedy refinement of vertex cover

```
function REFINED-APPROX-VC-1/2($G = (V, E)$)
    $C \leftarrow$ APPROX-VC-1($G$)
    for $v \in C$ do
        if $C - v$ is a vertex cover then
            $C \leftarrow C - v$
        end if
    end for
    return $C$
end function
```

III Implementation

We implemented multithreading in our project to execute the different algorithms we have in parallel. We have used a total of 7 threads: 1 thread for each of our 6 algorithms and 1 thread for IO. The input to each of our algorithm is fed from graphGen which takes number of vertices as an input and produces a graph with a certain number of edges.

For the vertex cover output of each algorithm, average run time, standard error for average run time and approximate ratio is calculated. For the CNF-SAT and CNF-3-SAT, vertices in the range of [5,15] with increments of 1 has been used. For the rest algorithms vertices in the range of [5,50] with increments of 5 has been used. For each vertex, 10 graphs are generated using graphGen and for each graph the running time is measured for 10 runs. Then the average and standard devidation across the 100 runs for each V is computed. The same procedure is applied to calculate the approximate ratio.

IV Result and Analysis

The average run time of CNF-SAT-VC and CNF-3SAT-VC is shown in Figure 1. We can see that the trend in the run time for both algorithms is increasing similarly to an exponential increase for higher values of V. Here we only reported time-out analysis for —V—¡15 because for higher number of vertices, the CNF SAT algorithms couldn't produce a vertex cover with the time out limit of 15 minutes. This happens because the number of literals and clauses that are generated by a reduction for the SAT solver increases significantly as the number of vertices increases.

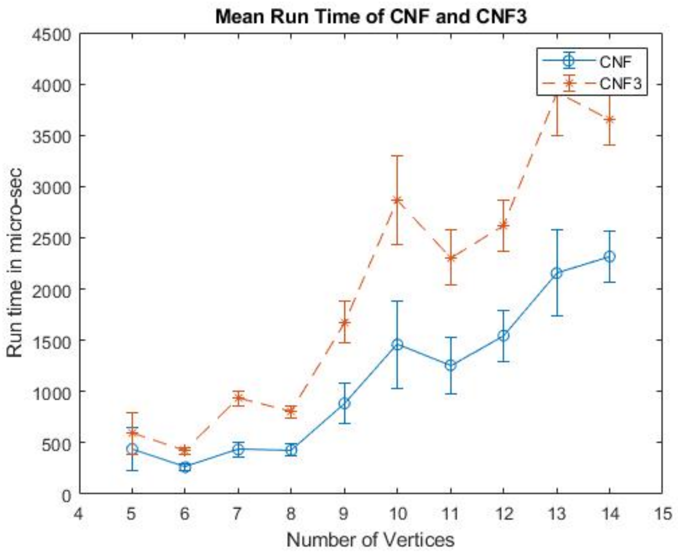

Figure 1: Mean run time for CNF SAT and 3CNF SAT algorithms

From the figure, we can also clearly see that the 3CNF SAT takes longer to run than the CNF SAT. This can be explained by the increased expressiveness of 3CNF formulas. In a 3CNF formula, each clause can contain up to 3 literals , whereas in a CNF formula, each clause can contain any number of literals. To change the CNF SAT to 3CNF reduction, we introduce additional variables. As the number of variables and clauses increases, the complexity for the SAT solvers to find a satisfying assignment is increased which leads to to longer runtimes for 3CNF SAT compared to CNF SAT.

Figure 2 shows the average run time for the approximate algorithms. APPROX1 stands for APPROX-VC-1 algorithm and APPROX2 stands for APPROX-VC-2 algorithm. We can see that the run time for both of these algorithms is considerably less than the CNF and 3CNF SATS. However, for almost all the inputs of V, the APPROX-VC-2 algorithms take less time to run. APPROX-VC-1 picks the vertex with the highest degree and removes edges associated with it until no edge remains. APPROX-VC-2 chooses a random edge and removes edges associated with that

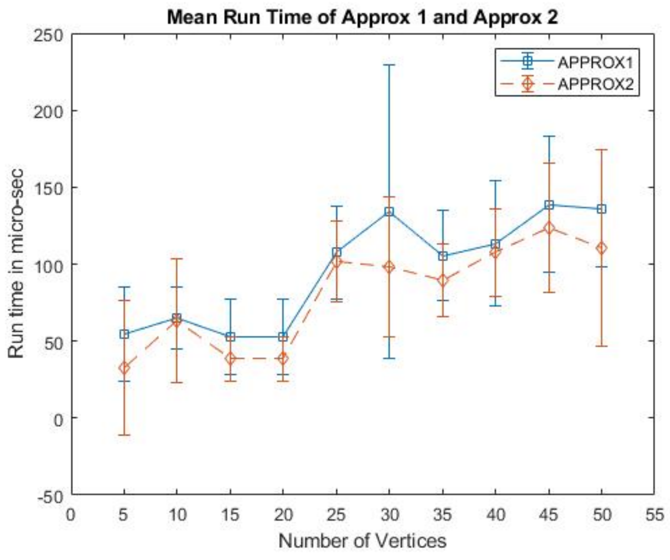

Figure 2: Mean run time for APPROX-VC-1 and APPROX-VC-2 algorithms

edge. Since it checks associated edges against 2 vertices at each iteration, it takes less time than APPROX-VC-1 which checks only against 1 vertex for each iteration.

Figure 3 and Figure 4 shows how the refined algorithms run time compares with their orignal

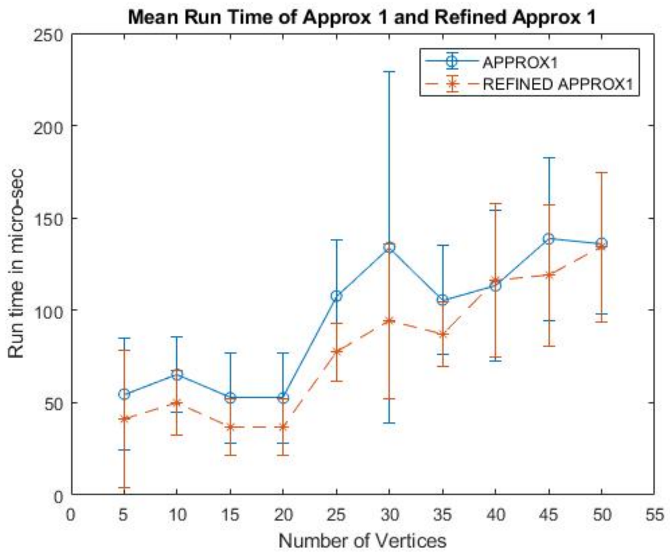

Figure 3: Mean run time for APPROX-VC-1 and REFINED APPROX-VC-2 algorithms

algorithms. Technically, the run time for the refined algorithms will be higher than the approximate algorithms. But in our case, the same thing can not be concluded because the inputs to the refined algorithms are fed from the vertex cover outputs of the approximate algorithms.

The refined algorithm for APPROX-VC-1 takes less time as shown in Figure 3. This is because the refined algorithm only iterates through the number of vertices in the vertex cover outputted by

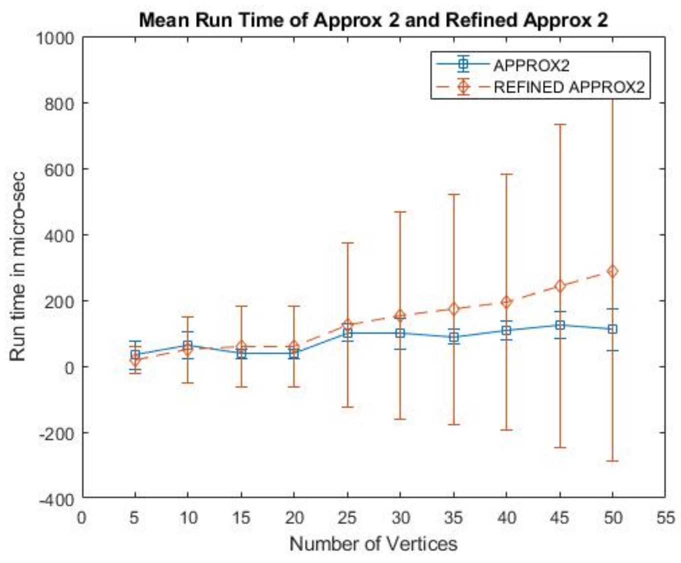

Figure 4: Mean run time for APPROX-VC-2 and REFINED APPROX-VC-2 algorithms

APPROX-VC-1, unlike APPROX-VC-1 algorithm which has to iterate through the input number of vertices. On the contrary, Figure 4 shows that the refined algorithm for APPROX-VC-2 runs longer than the original algorithm. Again, this can be attributed to the fact that APPROX-VC-2 is checking against two vertices (an edge) in each iteration while the refined algorithm iterates through each vertex in the vertex cover of APPROX-VC-2.

The mean approximation ratio for approximate and the refined algorithms is plotted in Figure 5. Input vertices in the range of [5,15] are used because for —V—¿15, the CNF SAT algorithms experience a time out which means approximate ratios can be generated. We can see from the graph that APPROX-VC-1 does considerably better in generating minimal vertex cover size than APPROX-VC-2. This is made possible by always choosing the vertex with the highest vertex degree first to eliminate associated edges. In fact, we can see that APPROX-VC-1 does almost as well as the CNF SATs in generating the optimal vertex cover.

As expected the refined algorithm for APPROX-VC-2 siginificantly imporves the approximate ratio for APPROX-VC-2. Adding two vertices at a time to a vertex cover doesn't help a lot in obtaining a minimal vertext cover so the refined algorithm helps remove a significant portion of the extra vertices in the vertex cover output of APPROX-VC-2. We can also see that REFINED APPROX-VC-1 improves the approximate ration for APPROX-VC-1 even though the improvement effect is not as large as REFINED APPROX-VC-2. In fact we can see that for —V—¡11, the APPROX-VC-1 was doing as well as its refined version.

V Conclusion

The aim of this project was to compare run time effeciency of six algorithms in solving the minimum vertex problem and how accurate they are. Even though CNF SAT and CNF3 SAT algorithms produce the optimal vertex cover, their run time duration increases exponentially with increasing vertex size and becomes infeasible to utilize for large number of vertices. The approximate algorithms help reduce the run time significanlty than the CNF SAT algorithms even though they don't

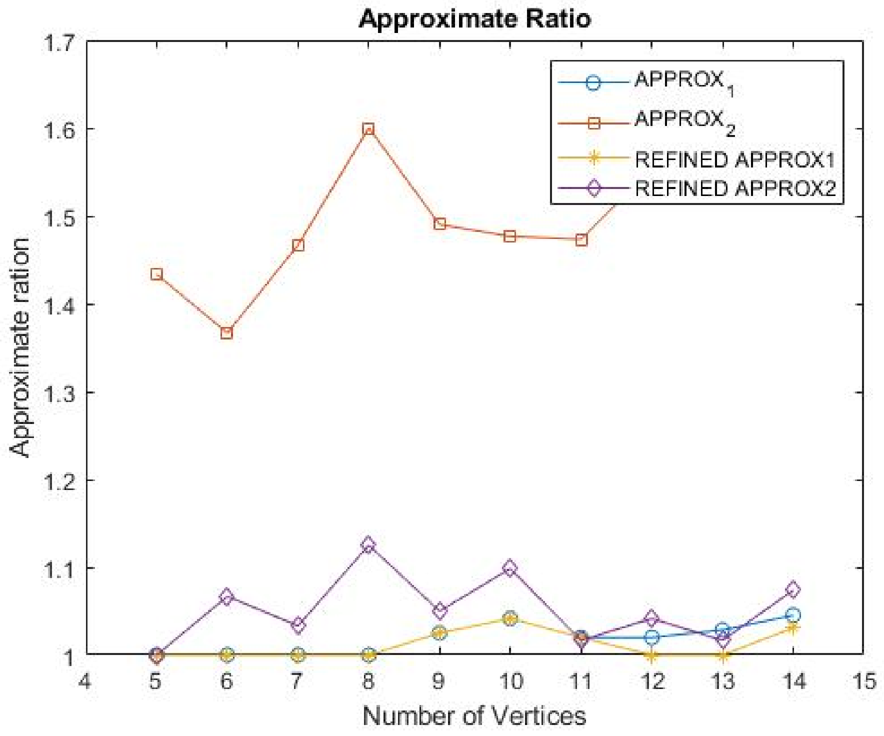

Figure 5: Caption for the image

produce optimal solution. While APPROX-VC-1 algorithm is more accurate in obtaining a minimal vertex cover, APPROX-VC-2 runs faster in producing the vertex covers. The refined algorithms help produce a vertex cover that is more accurate than the approximate algorithms.