UNIVERSITY OF WATERLOO

ECE 650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

Project : Quantitative Analysis of Mini-Vertex Cover Using Various Algorithms

SHREYAS BANGALORE SURESH
SHREYA PRASANNA

Supervised by Prof. Dr. Albert Wasef

Department of Electrical And Computer Engineering

Table of Contents

1 Introduction....3
2 Algorithms....4
2.1 CNF-SAT....4
2.2 APPROX- VC1....4
2.3 APPROX- VC1....4
3 Experimentation Specifics and evaluation of findings....5
4 Running Time Analysis....6-9
4.1 CNF....6
4.2 CNF in logarithmic time....6
4.3 Analysis....7
4.4 APPROX VC1....8
4.5 APPROX VC2....8
4.6 APPROX VC2 VS APPROX VC-1....9
4.7 Analysis....9
5 Approximation Ratios....10-11
5.1 Approximation Ratio for Approx VC1....10
5.2 Approximation Ratio for Approx VC1....10
5.3 Approximation Ratio for Approx VC1 vs Approx VC2 vs CNF....11
5. 4 Analysis....11
6 Refinement / Bonus....12
7 Conclusion....12

1 Introduction

Our central objective is to provide support to the local police department as they implement security cameras at traffic intersections. The primary aim is to optimize the installation of these cameras by reducing their number while ensur- ing maximum effectiveness in monitoring activities. The underlying challenge is rooted in the classical problem of finding a minimum vertex cover, a well-known NP-complete problem for which approximation algorithms have been developed. In the graph representation (G(V, E)), a vertex cover comprises a subset of vertices (V). This subset is carefully selected to guarantee that for every edge (u, v) in the graph (E), at least one of the vertices, either u or v, is included in the vertex cover. Our project delves into diverse techniques and algorithms employed to solve the parameterized Vertex Cover problem.

One of the initial approaches involves encoding the vertex cover problem into Conjunctive Normal Form-Satisfiability (CNF-SAT) clauses within the graph, utilizing the MiniSat solver. Additionally, we leverage two approximation al- gorithms, namely Approx-VC-1 and Approx-VC-2. These algorithms play a crucial role in calculating the vertex cover efficiently, allowing for a balanced and optimized deployment of security cameras at traffic intersections.

A vertex cover of a graph $G(V, E)$ is a subset of vertices V such that for every edge (u, v) is a subset of E, at least one of the vertices u or v is in the vertex cover. In this project we discuss different techniques and algorithms used to solve the parameterized Vertex Cover problem. The first approach is to encode vertex cover to CNF-SAT clauses in a graph (using MiniSat). We also have two approximation algorithms Approx-VC-1 and Approx-VC-2 that are used to calculate the vertex cover in polynomial time.

2 Algorithms

2.1 CNF-SAT:

The process of reducing the vertex cover to CNF-SAT is a polynomial-time operation. The graph is represented using literals and clauses, structured in Conjunctive Normal Form (CNF). A4 encoding is employed to construct these clauses. In CNF, each clause consists of literals connected by logical OR operations, and these clauses are connected by logical AND operations. The resulting CNF expression is then fed into the MiniSat SAT solver, which determines whether the given clauses are satisfied, indicating the existence of a vertex cover for the current graph.

2.2 Approx-VC-1:

This algorithm begins by identifying the vertex that frequently appears in the provided set of connections, specifically the one with the highest degree (number of connections). The chosen vertex is added to the list of covered vertices, and all connections associated with it are removed. This process is iteratively repeated until no connections remain. In essence, the algorithm systematically builds a list of vertices that collectively cover all connections in the set.

2.3 Approx-VC-2:

In this strategy, we start by choosing an edge (u, v) from the set of edges (E) and include both vertices in the vertex cover (VC). Subsequently, we remove all edges connected to these chosen vertices. This process is repeated, picking edges and eliminating connected edges, until no edges are left.

To tackle the minimum vertex cover problem, our solution employs four threads. The main function creates one thread for input/output (I/O) and three additional threads for each algorithm: CNF-SAT, Approx-VC-1, and Approx-VC-2. The I/O thread, managed by the main function, reads the user-provided input (V, E) and generates the adjacency list based on it.

3 Experimentation Specifics and evaluation of findings

In order to find the efficiency of these three algorithms for the number of vertices, we make use of two factors namely (1) Running time, and (2) Approximation Ratio. We generate 10 graphs for each vertex. With help of this, we compute the Running time and the Approximation ratio for each graph.

To measure the running time of an algorithm we make use of a clock() function which gives us the CPU execution time for a thread. Depending upon the number of vertices that is given as input we can calculate the running time for each algorithm and find the maximum CPU time taken by an algorithm.

As CNF sat solver takes exponential time to solve the literals we see a huge spike in time after 15+ vertex graphs. So we have calculated the metrics of the CNF algorithm for graphs with vertex numbers ranging from 5 to 15.

And for approx VC1 and approx VC2 we calculate the running time and approximation ratio for various values of V (number of vertices), for the graphs generated by graphGen. The graph ranges from V = 5 to V = 50 with an interval of 5 . Then, we compute the average and standard deviation across those 100 samples for each value of V.
We compare this with the running time of the CNF-SAT for those values of V ranging from 5 to 15 with an interval of 1.

4 Running Time Analysis

4.1 CNF

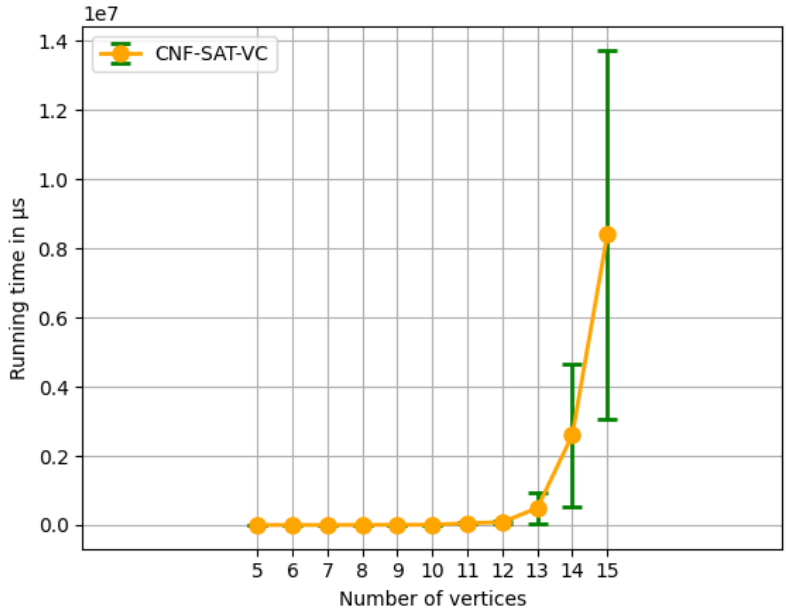

Figure 1: Running time of CNF-SAT

4.2 CNF in logarithmic time

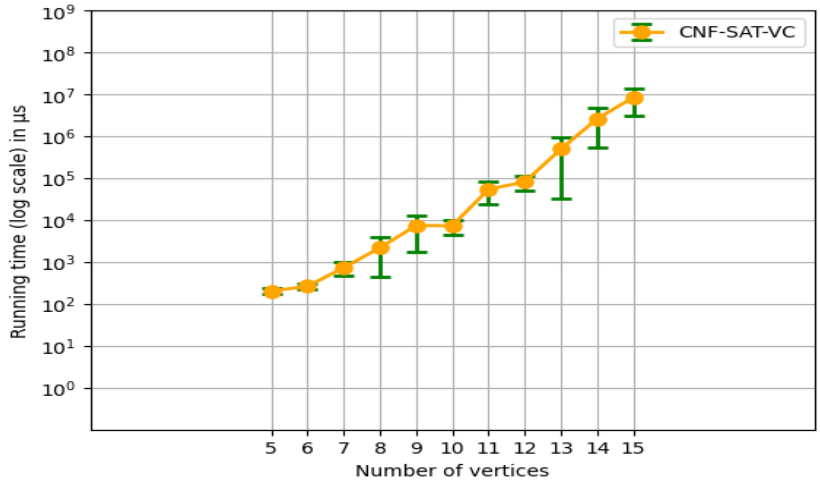

Figure 2: Running time of CNF-SAT in logarithmic time

4.3 Analysis:

The minimum vertex cover problem is known to be NP-hard, meaning there's no quick algorithm that can solve it for large inputs. So, when we reduce the CNF SAT problem to the minimum vertex cover problem, it can take a very long time to run, especially as the number of vertices increases. To address this, we've implemented a timeout feature using the pthread utility. If the CNF threads take more than 120 seconds to complete, the timeout function kicks in, interrupting and stopping those threads to prevent excessively long computation.

4.4 APPROX VC1

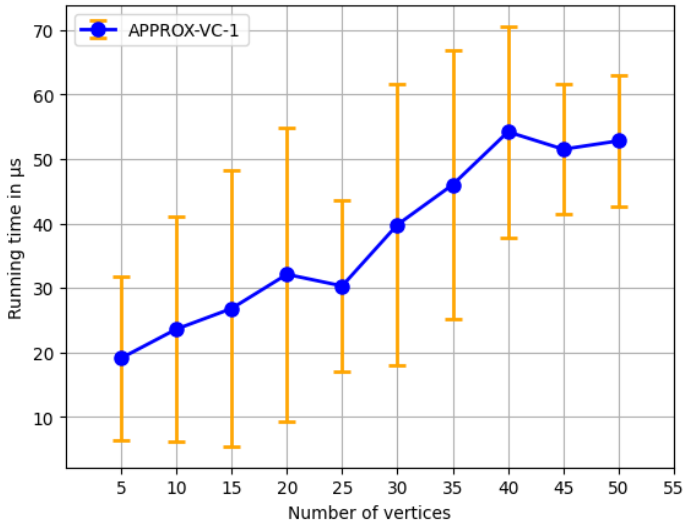

Figure 3: Running time of APPROX VC-1 in microseconds time

4. 5 APPROX VC2

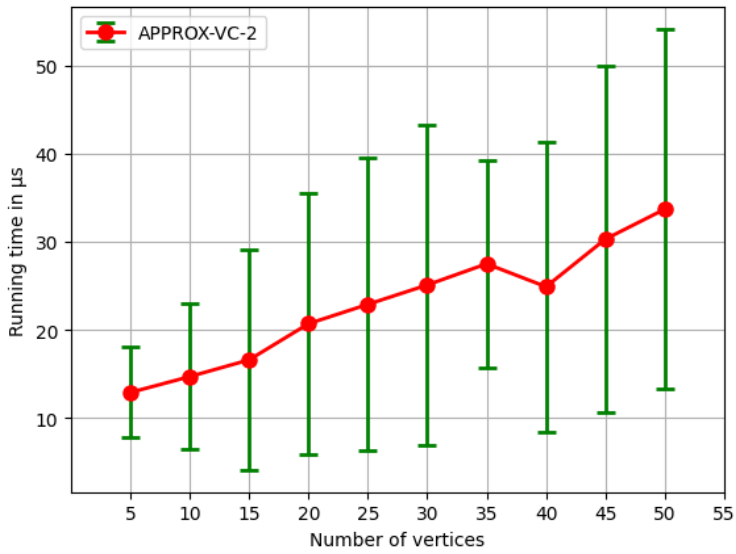

Figure 4: Running time of APPROX VC-2 in microseconds time

4. 6 APPROX VC2 VS APPROX VC-1

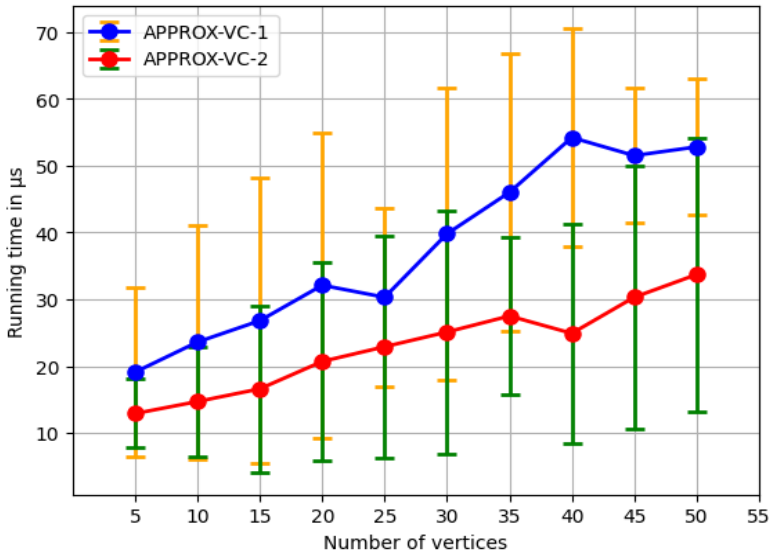

Figure 5: Running time of APPROX VC-1 vs APPROX VC-2 in microseconds time

4. 7 Analysis:

In Figure 5, we examined how long it takes for APPROX-VC-1 and APPROX-VC-2 to run, plotting the average (mean) and the spread (standard deviation) of running times for each set of vertices in the range of 5 to 50. The graph illustrates that as the number of vertices increases, both algorithms take more time to complete. However, it's noticeable that APPROX-VC-1 generally takes more time compared to APPROX-VC-2.This difference arises because APPROX-VC-1 is designed for optimization problems where it calculates the highest degree of a vertex and then iteratively builds a list of covered vertices by visiting highest degree vertex each time O(|V|) and then removing the edges surrounding it O(|E|). So the time complexity of this algorithm is O(|V|*|E|) where |V| is the number of vertices of the graph and |E| is the number of edges of the graph.
On the other hand, APPROX-VC-2 removes an edge first and includes both the vertices following that it removes all the edges belonging to that. Removing an edge of a vertex in the graph takes O(|E|) time complexity and removing all vertices edges will take O(|V|*|E|). So the overall time complexity will be O(|V|*|E|+|E|) where |V| is the number of vertices of the graph and |E| is the number of edges of the graph. However, we have optimized the brute force method even further and instead of explicitly removing edges which takes O(|E|) time for each vertices, we have a visited vertex list which increases the space complexity by O(|V|) and reduces the O(|V|*|E|+|E|) to O(|V|+|E|). So by using space-time tradeoff we have optimized the APPROX-VC-2 further. As a result, the time complexity of APPROX-VC-1 is a bit higher than that of APPROX-VC-2.

5 Approximation Ratios

5. 1 Approximation Ratio for APPROX VC-1

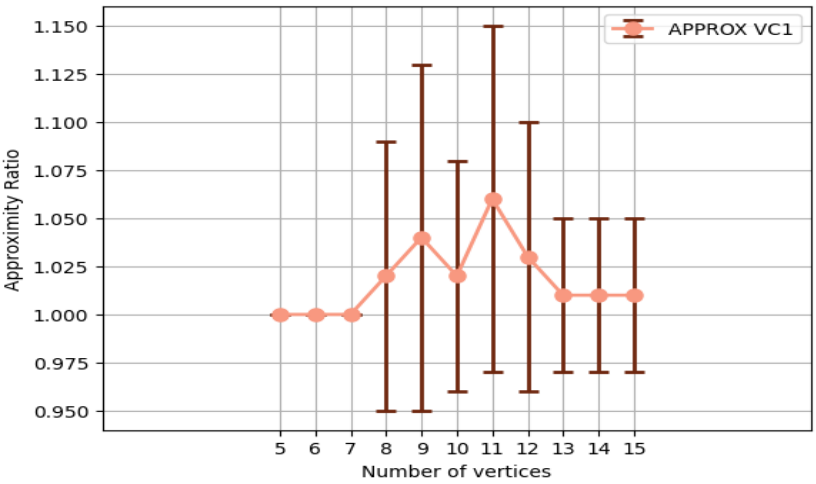

Figure 6: Approximate ratio for APPROX VC-1

5.2 Approximation Ratio for APPROX VC-2

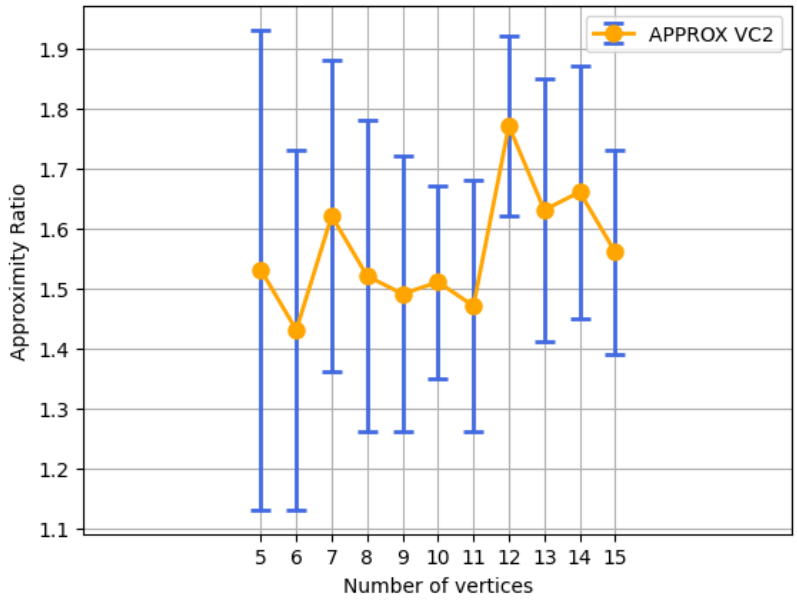

Figure 7: Approximate ratio for APPROX VC-2

5.3 Approximation Ratio for APPROX VC-1 vs APPROX VC-2 vs CNF

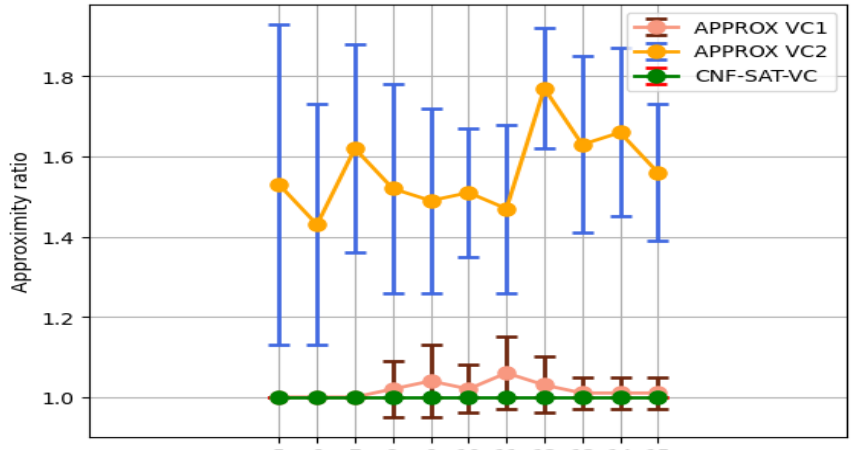

Figure 8: Approximate ratio for APPROX VC-1 vs APPROX VC-2 vs CNF

5. 4 Analysis

We calculated the approximation ratios of APPROX-VC-1, APPROX-VC-2 which represents the comparison between the size of the computed vertex cover and that of a vertex cover with minimal sizes(i.e. CNF SAT). The approximation ratio of these algorithms is shown in Figure 6 and Figure 7 where the x-axis denotes the number of vertices and the y-axis is the approximation ratio. We draw the graphs with values V = [5,15]. Seeing the comparison graph in Figure 8 we found out the following findings:

• Since CNF ensures finding the minimum number of vertices covered, the approximation ratio is equal to 1 in all vertex situations here.

• We can see that the vertex cover produced by APPROX-VC-1 is nearly identical to that of CNF-SAT as it greedly includes the vertices with highest degree.

• This is not the case for APPROX-VC-2. The approximation ratio of APPROX-VC-2 shows it cannot always generate the minimum vertex cover accurately. This is because we arbitrarily choose the edges and then include those vertices connected to the edge in the vertex cover. So though this algorithm might be time efficient, it is not the best algorithm in terms of approximation ratio metrics.

6.0 Refinement / Bonus

Binary search is being used to find the minimum vertex cover in the graph which has a time complexity of O(log |V|). It is more efficient than linear search(time complexity O(|V|)) for large graphs because it eliminates half of the search space at each iteration. The idea is to first calculate the maximum possible size of the vertex cover. This size is equal to the number of vertices in the graph. Then we set the upper limit of the search space to be this maximum size, and the lower limit to be 1 (since the minimum vertex cover size is at least 1). At each iteration of the binary search, we calculate the vertex cover size for the midpoint of the search space (i.e., the average of the upper and lower limits). If the vertex cover size is less than or equal to k (i.e., the size we are looking for), we update the lower limit of the search space to be the midpoint + 1. Otherwise, we update the upper limit of the search space to be the mid-point-1. We keep track of the minimum vertex cover seen so far. This way, we gradually narrow down the search space until we find the minimum vertex cover. This process takes logarithmic time in the worst case, which is much faster than a linear search for large graphs.

7.0 Conclusion

In the context of solving the vertex cover problem, the CNF-SAT approach proves highly effective for smaller datasets but encounters scalability challenges as the dataset size increases, resulting in prolonged running times. Although the APPROX-VC-2 algorithm boasts the shortest running time, its trade-off lies in a less consistent ability to generate the smallest vertex cover, as indicated by its approximation ratio.

As the dataset scales up, APPROX-VC-1 becomes a more pragmatic choice, exhibiting a commendable balance between processing speed and result accuracy. Despite its intermediate running time, APPROX-VC-1 outperforms CNF-SAT with larger datasets and offers results that closely approximate the ideal vertex cover in terms of the approximation ratio.

In essence, the algorithm choice should be tailored to the specific context, considering factors such as dataset size and the desired equilibrium between processing speed and accuracy. While CNF-SAT excels in smaller-scale scenarios, the dynamics shift with larger datasets, prompting careful consideration of the trade-offs between APPROX-VC-1 and APPROX-VC-2.