ECE-650 Final Project Report

Dip Patel
University of Waterloo
Waterloo, ON, CA
dv9patel@uwaterloo.ca

Anupam Rajanish
University of Waterloo
Waterloo, ON, CA
aranupam@uwaterloo.ca

Abstract

This report analyzes three algorithms for solving the minimum vertex cover problem: CNF-SAT-VC, APPROX-VC-1, and APPROX-VC-2. Using test graphs with 5 to 15 vertices, the study evaluates their performance in terms of efficiency, approximation accuracy, and scalability. While CNF-SAT-VC provides exact solutions, its exponential runtime limits scalability. APPROX-VC-1 and APPROX-VC-2 offer faster, nearly linear performance with trade-offs in accuracy. Optimization techniques for CNF-SAT-VC are also proposed to improve its scalability. The findings highlight the trade-off between accuracy and efficiency in solving vertex cover problems.

1 Introduction

This report presents a comparative analysis of three algorithms designed to solve the minimum vertex cover problem, namely CNF-SAT-VC, APPROX-VC-1, and APPROX-VC-2. To evaluate their performance, we tested the algorithms on a test-suite of graphs with an increasing number of vertices, where $|V| \in [5, 50]$ (V is number of vertices). The analysis examines how each algorithm handles varying graph inputs, measuring their ability to provide optimal vertex covers alongside relevant performance metrics. Key trends, such as linear or exponential growth in running time and spikes in approximation ratio, are explored to understand their behavior under varying conditions. Additionally, the report explores potential optimizations for improving the CNF-SAT-VC encoding, aimed at enhancing its scalability for larger graphs.

2 Task

approx-vc-1 :- The implementation of APPROX-VC-1 involves selecting the vertex with the highest degree, adding it to the vertex cover, and removing all edges connected to that vertex. This process is repeated until no edges are left.

approx-vc-2 :- The implementation of APPROX-VC-2 involves selecting an edge $< u, v >$, and both vertex u and v to vertex cover, while removing all the edges attached to u and v. This process is repeated until no edges are left.

Finally, the task is to make these two implementations, as well as the CNF-SAT method, multithreaded.

3 Input

The input is generated using the script file located at "/home/agurfink/ece650/graphGen/graphGen". For the analysis, graphs with sizes ranging from 5 to 15 vertices are created. This range is chosen because CNF-SAT takes an excessively long time to produce results when the number of vertices in the graph exceeds 15.

4 Result & Analysis

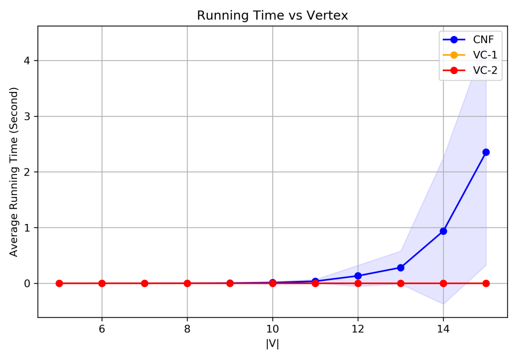

Figure 1: 'Running Time' Vs 'Number of Vertex' (measured in seconds)

The figure clearly illustrates that CNF-SAT exhibits exponential time complexity, while Approx-VC-1 and Approx-VC-2 show nearly linear time complexity relative to the number of vertices in the graph. As a result, although CNF-SAT provides an accurate solution, its runtime increases exponentially with the number of vertices, making it impractical for scaling to larger instances. To address this, approximation algorithms offer a viable alternative. While their solutions may not always be perfectly minimal, they are often close to optimal. Additionally, their major advantage lies in their linear time complexity, enabling scalability to larger input sizes.

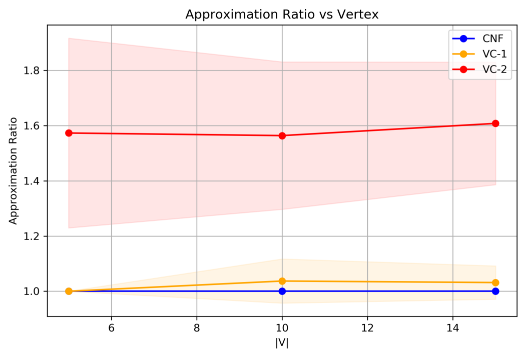

Figure 2: 'Approximation Ratio' Vs 'Number of Vertex'

Figure 2 presents a comparison of approximation ratios. It is evident from the figure that the approximation ratio of Approx-VC-1 closely aligns with the actual solution. In contrast, the approximation ratio of Approx-VC-2 ranges between 1.5 and 1.6. Therefore, Approx-VC-1 is the preferred choice for achieving more accurate results compared to Approx-VC-2.

The higher approximation ratio of Approximation-VC-2 can be attributed to its selection strategy. Specifically, it adds both vertices of an edge to the vertex cover, even though only one is guaranteed to belong to it, without knowing which one. As a result, including both vertices can lead to an approximation ratio of 2 in the worst case. However, this value represents the upper bound and cannot be exceeded. On the other hand, the greedy approach, which selects the vertex with the maximum number of connected edges, performs well as it generally results in a smaller approximation ratio.

Finally, Figure 3 compares the running times of Approx-VC-1 and Approx-VC-2. The graph clearly shows that Approx-VC-2 is faster than Approx-VC-1. This difference can be attributed to their respective approaches. In Approx-VC-2, two vertices are added to the vertex cover in each iteration. In contrast, Approx-VC-1 requires finding the vertex with the highest degree, which takes $\Theta(n)$, before adding it to the vertex cover.

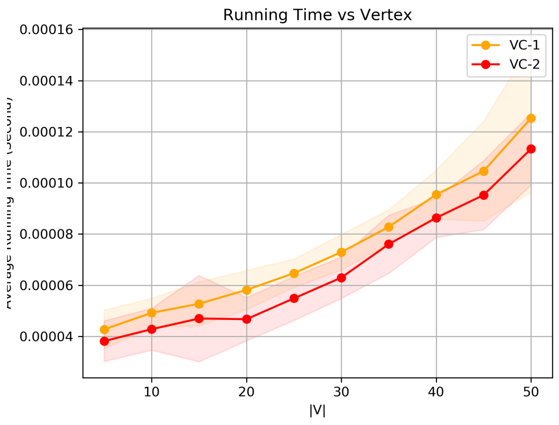

Figure 3: 'Running Time' Vs 'Number of Vertex' (Measured in Seconds)

5 Optimization

Given a graph G=(V,E), where V represents the set of vertices and E represents the set of edges, the task is to find the minimum vertex cover for the graph. To solve this problem, one can transform it into a CNF-SAT problem in polynomial time. By using an off-the-shelf CNF-SAT solver, the problem can be solved efficiently for smaller graphs. However, since the vertex cover problem is NP-complete, no known polynomial-time algorithm exists for solving it. As a result, the time required to solve the problem increases exponentially as the size of the graph grows.

In assignment-4, author proposed generic way to encode VERTEX-COVER problem into CNF-SAT.

In proposed solution, The reduction uses n x k different atomic propositions. Where, n cardinality of V and k represents the number of vertices included in the solution. Each atomic proposition is denoted as $X_{i,j}$, where i ∈ [1,n] and j ∈ [1,k]. The atomic proposition $X_{i,j}$ is true if the i^{th} vertex in the set $\mathcal{V}$ is selected as one of the k^{th} vertices in the vertex cover solution.

The reduction consists of the following clauses.

- At least one vertex is selected for the vertex cover:

$$\forall i \in [1,k], \quad (x_{1,i} \vee x_{2,i} \vee \cdots \vee x_{n,i})$$

- No vertex appears twice in the vertex cover:

$$\forall m \in [1, n], \forall p, q \in [1, k] \text{ with } p < q,$$
$$(\neg x_{m,p} \vee \neg x_{m,q})$$

- No more than one vertex in the m^{th} position:

$$\forall m \in [1, k], \forall p, q \in [1, n] \text{ with } p < q,$$
$$(\neg x_{p,m} \vee \neg x_{q,m})$$

- Every edge is incident to at least one vertex in the cover:

$$\forall \langle i, j \rangle \in E, (x_{i,1} \vee \cdots$$
$$\vee x_{i,k} \vee x_{j,1} \vee \cdots \vee x_{j,k})$$

For sake of comparison with other optimized reduction, let's calculate total number of clauses above method will generate.

$$|C| = 1 + \frac{|V| \cdot k \cdot (k-1)}{2}$$
$$+ \frac{|V| \cdot (|V| - 1) \cdot k}{2} + |E| \cdot k$$

The above expression can be further simplified based on the fact that $k < |V|$.

$$|C| \approx |V|^2 \cdot k + |E| \cdot k + k^2 \cdot |V|$$

If the graph G=(V,E) has a vertex set cardinality of n = 50, and it is assumed that the minimum vertex cover contains 10 vertices, the resulting formulation would include more than 25,000 clauses. Additionally, the consideration of the edge set cardinality exacerbates the complexity. As a result, the proposed naive reduction strategy does not scale well for larger problem instances.

Hence, an optimized approach is required.

5.1 Trick 1

To understand this optimization, a basic familiarity with the overall procedure is essential. The following pseudo-code provides a general overview

The pseudocode outlines an iterative approach to finding the minimum vertex cover of a graph G=(V,E). It converts the problem into a CNF-SAT formulation and iteratively checks for satisfiability with increasing values of k, representing the size of the vertex cover. Once a satisfiable solution is found, it converts the result back to a vertex cover and terminates.

To optimize, the above structure can be leveraged effectively.

Algorithm 1 Vertex Cover

```
1: Input: G = (V,E).
2: Output: List of vertices present in min-vertex-cover.
3: convert Problem into CNF-SAT.
4: for k in range(1,|V|) do
5:     if CNF-SAT for ‘k‘ is satisfiable then
6:         convert result of CNF-SAT to Vertex-cover.
7:         break;
8:     else
9:         continue;
10:    end if
11: end for
12: return Result of vertex-cover
```

Proof. **Claim :** For the pseudocode above, all clauses ensuring that a single vertex cannot occupy multiple slots can be eliminated.

Proof : In order to prove above claim, let's assume that we removed all clauses which ensures that; single vertex cannot occupy multiple slots. Which allows algorithm to choose one vertex in multiple slots.

Now, without loss of generality, let's assume that for k = m > 1, above code gets terminated with answer, v_1, v_2, v_3, ..., v_m, where v_i = v_j and, i < j ≤ m.

Which means that, min-vertex-cover has in total m-1 vertices. It comes from the fact that after the removal of clause; code generate answer v_1, v_2, v_3, ..., v_m, where v_i = v_j and, i < j ≤ m.

Which further implies that k = m - 1 problem is not satisfiable. Because if it is then code should have terminated at k = m - 1. Moreover, if problem is unsatisfiable at k = m - 1, then from answer above, we can generate new answer by removing duplicate vertex from the answer. This newly generated answer then must satisfy while k = m - 1. Which again contradict our assumption.

Thus, it can be concluded that the answer cannot contain duplicate vertices, as any such scenario would have been addressed in a previous iteration by generating a solution without duplication. Therefore, it is proven that removing all clauses ensuring "a single vertex cannot occupy multiple slots" does not affect the outcome.

□

Therefore, with above optimization, we can effectively reduce $k^2 \cdot |V|$ number of clauses.

5.2 Trick 2

Clauses ensuring that no more than one vertex occupies the m^{th} position can be further optimized using binary encoding. With naive encoding, $|V|^2 \cdot m$ clauses are required to enforce this constraint. However, binary encoding reduces the total number of clauses by introducing additional, more efficient clauses to achieve the same outcome.

Binary encoding for "At Most One"

To enforce that at most one variable in x_1, x_2, x_3, x_4 can be true:

1. **Assign Binary Codes:** Each variable gets a unique binary code using $\lceil log_2(4) \rceil$ auxiliary variable y_1 and y_2 :
   - $x_1 = 00, x_2 = 01, x_3 = 10, x_4 = 11.$
2. **Adding Encoding Constraints:** For each x_i, enforce that if x_i is true, y_1 and y_2 must match the binary code assigned to x_i.
   - $\neg x_1 \lor (\neg y_1 \land \neg y_2)$
   - $\neg x_2 \lor (\neg y_1 \land y_2)$
   - $\neg x_3 \lor (y_1 \land \neg y_2)$
   - $\neg x_4 \lor (y_1 \land y_2)$
3. **Ensure Consistency:** These constraints naturally enforce that at most one x_i can match the values of y_1 and y_2, ensuring that the "at most one" condition.

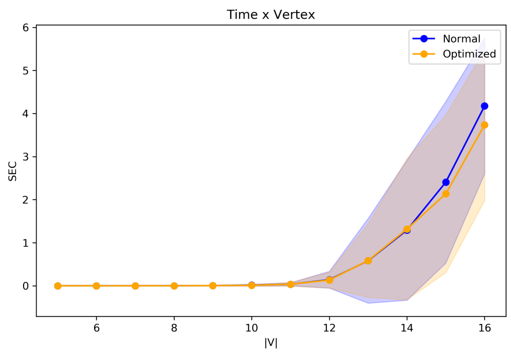

Figure 4: 'Approximation Ratio' Vs 'Number of Vertex'

5.3 Result

The graph in Figure 1 illustrates a comparison of the running time between the "Normal" and "Optimized" approaches for processing a problem as the input size ($|V|$) increases. From the figure, it is evident that the "Optimized" approach outperforms the "Normal" approach, particularly for larger input sizes. For smaller input sizes ($|V| \in$ [5,14]), the running times of both approaches are nearly identical, with minimal difference in performance. However, for larger input sizes ($|V| \geq 14$), the distinction becomes more pronounced.