University of Waterloo

ECE 650 - METHODS AND TOOLS FOR SOFTWARE ENGINEERING

Project : Analysis Of Six Approaches To Vertex Cover Problem

ARYAN KANWAR
JASPREET BHINDER

Department of Electrical And Computer Engineering

Supervised by
Prof. Dr. Reza Babaee

April 13, 2023

Abstract

A set of vertices is considered the vertex cover of a graph if each edge of the graph is incident to at least one of the set's vertices. A common example of an NP-hard optimization issue with an approximation approach is the challenge of finding a minimum vertex cover, which is a classic optimization problem in computer science. It will be implemented by six different methods in this study; relevant techniques and experimental information will be presented in sections 2 and 3. Section 4 will provide and discuss the running duration section 5 will provide approximation ratio in order to examine the effectiveness of each strategy. The results of the analysis are discussed in section 6.

Table of Contents

1 Algorithms

1. CNF-SAT: We present a polynomial time reduction from VERTEX-COVER to CNF-SAT. A polynomial time reduction is an algorithm that runs in time polynomial in its input. And then we use a Minisat SAT solver to find the solution.

2. 3-CNF-SAT:We present a polynomial time reduction from VERTEX-COVER to 3-CNF-SAT. A polynomial time reduction is an algorithm that runs in time polynomial in its input. And then we use a Minisat SAT solver to find the solution with at most 3 literals in each clause.

3. APPROX-1:
   Step 1: First, we find a vertex that has the highest degree.
   Step 2: Add it to the vertex cover and throw away all edges incident on that vertex.
   Step 3: Repeat these steps until no edges remain.

4. REFINED-APPROX-1: This algorithm implements a refined approximation algorithm (APPROX-1) for finding a minimum vertex cover in a graph. It uses a greedy approach to sort vertices based on their degree and then iteratively removes edges connected to the vertices in the sorted order until a vertex cover is found.

5. APPROX-2:
   Step 1: First, we pick an edge u,v.
   Step 2: Add both u and v to the vertex cover, and throw away all edges attached to u and v.
   Step 3: Repeat steps 1 and 2 till no edges remain.

6. REFINED-APPROX-2: This algorithm implements a refined approximation algorithm (APPROX-2) for finding a minimum vertex cover in a graph. It uses a greedy approach to sort vertices based on their degree and then iteratively removes edges connected to the vertices in the sorted order until a vertex cover is found.

2 Experimentation Specifics

The result of graphGen on eceubuntu, which creates graphs with the same amount of edges for a specific number of vertices, is what we utilize as our input. After that, the I/O thread should process the inputs and store the vertices and edges in two vectors. Each algorithm thread accepts the graph's data as input, and the parameters are the integer V for the number of vertices and the vector S for the number of edges for each graph. The output of the vertex cover program is the least vertex cover, which is a vector sorted from low to high value after each approach's individual computation in each of the three threads. Here, six distinct vectors were established to store the output vertex cover.

3 Evaluation of Experimental Findings

We calculated each approach's running time and approximation ratio in order to evaluate how effective it is. To measure them, we created 10 graphs for each value of V (V= 5, 10, 15, 20,..., 45, 50), ran the program 10 times for each graph, and then averaged the results of those 100 runs to find the mean and standard deviation for each value of V. Next, we shall examine these two features of the experimental findings:

4 Running Time Analysis

4.1 CNF VS 3 CNF

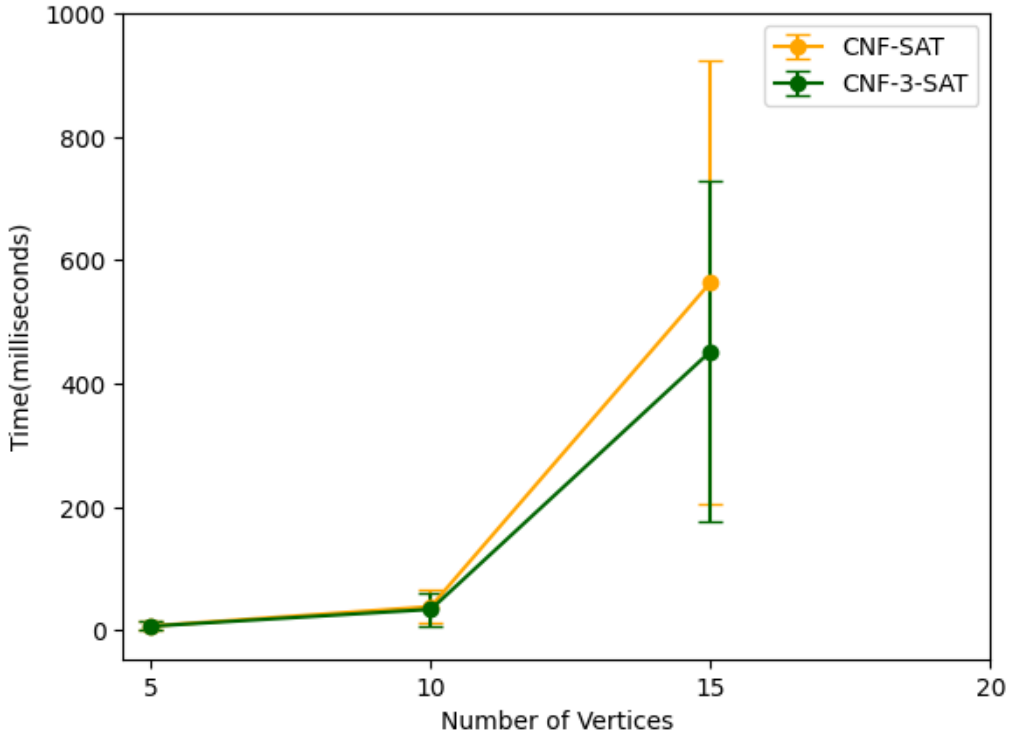

Figure 1: Running time of CNF-SAT and 3-CNF-SAT.

- CNF SAT: The minimum vertex cover problem is NP-hard, which means there is no known polynomial-time algorithm that can solve this problem. The CNF SAT reduction to the minimum vertex cover problem also runs in exponential time in the worst case.
- CNF-3-SAT-VC: The CNF-3-SAT-VC algorithm is a modification of the CNF-SAT-VC algorithm, which converts the original CNF-SAT instance into a 3-CNF instance with a maximum of 3 literals per clause. This reduction can be done in polynomial time. The overall time complexity of the CNF-3-SAT-VC algorithm will be the time complexity of the CNF-SAT to 3-CNF reduction, plus the time complexity of the vertex cover approximation algorithm used. Therefore, in terms of time complexity, the CNF-3-SAT-VC algorithm is faster than the CNF-SAT-VC algorithm.

-

Standard Deviation-

In comparison to smaller values of vertices, the standard deviation of running times has dramatically increased when the vertex is 15. This shows that, compared to the smaller vertex values, the running times for the CNF-SAT Algorithm at vertex 15 are significantly more dispersed and inconsistent.
In other words, vertex 15 has a wider range of running periods than smaller vertices. This shows that as the number of vertices increases, the CNF-SAT Algorithm becomes less predictable and more sensitive to changes in input parameters. It is also observed that for standard deviation 3 CNF-SAT Algorithm has similar behavior to that of the CNF-SAT Algorithm. -

CNF vs 3 CNF graph has data up to only V 15

The satisfiability problem for conjunctive normal form (CNF-SAT) and 3-conjunctive normal form (3-CNF-SAT) are both known to be NP-complete, which means that there is no known polynomial time algorithm that can solve them for all possible instances of the problem. As the input size and minimum vertex size increase, the number of possible variable assignments to be checked grows exponentially. This makes it computationally infeasible to solve these problems for large input sizes using brute force or exhaustive search methods. We displayed timeout if it takes more than 6 seconds to compute the vertex cover.

When there is a high number of vertices and edges in the graph, the CNF-SAT, 3 CNF-SAT method runs much more slowly. We discovered throughout the experiment that graphs with 20 or more vertices will take a very long time to run; the data is not displayed as timeout occurs before the minimum cover is computed by the program.

Can we avoid/improve it?
There are efficient algorithms for solving CNF-SAT and 3-CNF-SAT for some special cases or instances of the problem. For example, algorithms such as Davis-Putnam-Logemann-Loveland (DPLL) and Conflict-Driven Clause Learning (CDCL) are commonly used in SAT solvers to efficiently solve large instances of the problem.

In summary, CNF-SAT and 3-CNF-SAT may be computationally infeasible to solve for large input sizes using brute force or exhaustive search methods.

4.2 APPROX-1 vs APPROX-2

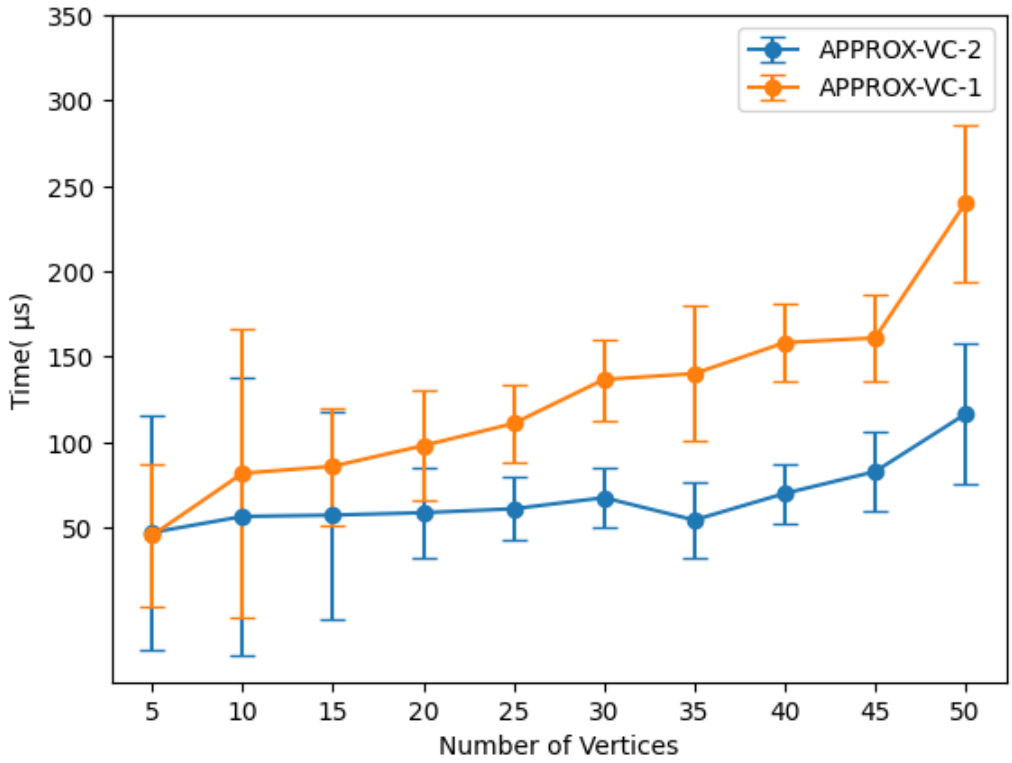

Figure 2: Running time of APPROX-1 and APPROX-2.

Time Comparison

When comparing these two algorithms, APPROX-1 requires more processing time than APPROX-2 because, when, choosing the vertex with the highest degree requires more sequential loops and calculations in APPROX-1, whereas APPROX-2 simply chooses the edge in random order.

- APPROX-VC-1: This algorithm has a time complexity of $O(m * n)$, where m is the number of edges and n is the number of vertices in the graph. The vertex cover size produced by this algorithm is guaranteed to be no more than twice the size of the optimal vertex cover.
- APPROX-VC-2:This algorithm has a time complexity of O(m log m), where m is the number of edges in the graph. The vertex cover size produced by this algorithm is also guaranteed to be no more than twice the size of the optimal vertex cover.

Standard Deviation

- For small graphs, APPROX-VC-1 may produce a smaller standard deviation compared to APPROX-VC-2. This is because APPROX-VC-1 tends to select vertices with high degrees, which cover more edges, resulting in a more uniform distribution of vertex cover sizes.
- For larger graphs, APPROX-VC-2 may produce a smaller standard deviation compared to APPROX-VC-1. This is because as the graph size grows, the number of high-degree vertices decreases, making it less likely for APPROX-VC-1 to select a vertex with a significantly higher degree than the other vertices. In contrast, APPROX-VC-2 selects vertices in pairs, which allows for a more balanced distribution of vertex cover sizes.

Conclusion-
In terms of time complexity, APPROX-VC-2 is generally faster than APPROX-VC-1 for most graphs.

4.3 REFINED-APPROX-VC-1 vs REFINED-APPROX-VC-2

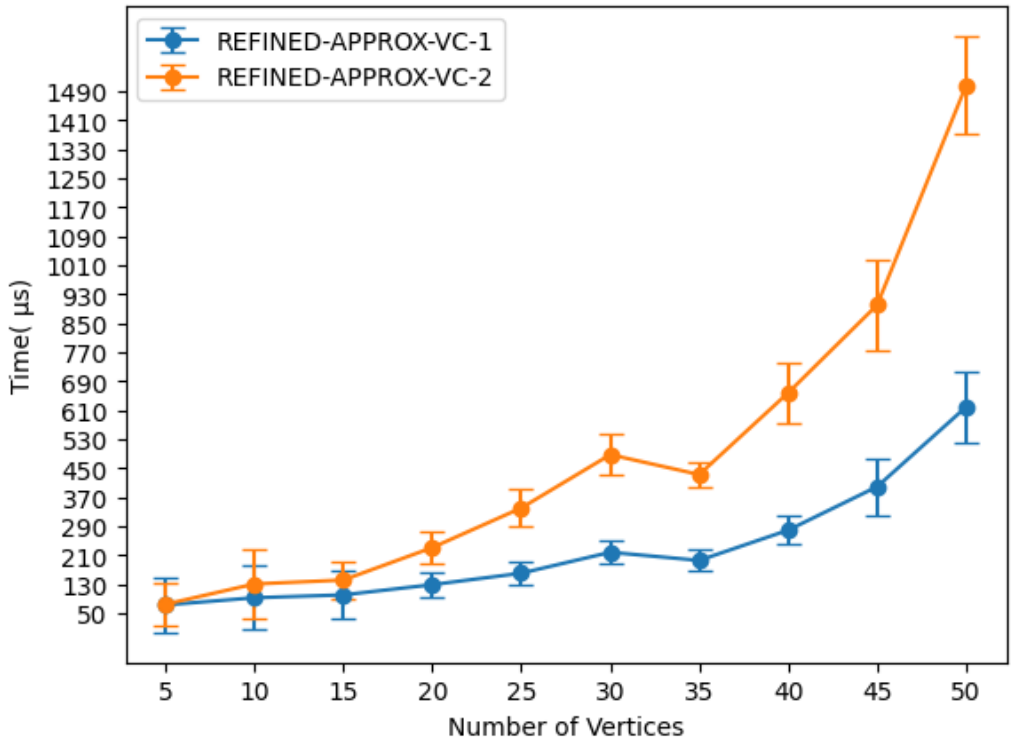

Figure 3: Running time of REFINED-APPROX-VC-1 and REFINED-APPROX-VC-2

REFINED-APPROX-VC-2 is generally faster than **REFINED-APPROX-VC-1** for most graphs

- The worst-case time complexity of REFINED-APPROX-VC-1 is the complexity of APPROX-VC-1 O(mn) + complexity of choosing vertices greedily $O(n^3)$, so overall $O(n^3)$.
- The worst-case time complexity of REFINED-APPROX-VC-2 is the complexity of APPROX-VC-2 O(mlogn) + complexity of choosing vertices greedily $O(n^3)$., so overall $O(n^3)$.

  where m is the number of edges and n is the number of vertices in the graph.

Conclusion -
The time complexity of both algorithms also depends on the specific graph structure and the size of the vertex cover produced by their basic counterparts (i.e., APPROX-VC-1 and APPROX-VC-2). In some cases, REFINED-APPROX-VC-2 may be faster than REFINED-APPROX-VC-1.

Overall, the choice between REFINED-APPROX-VC-1 and REFINED-APPROX-VC-2 depends on the specific graph properties and the desired trade-off between time complexity and solution quality.

5 Approximation Ratio

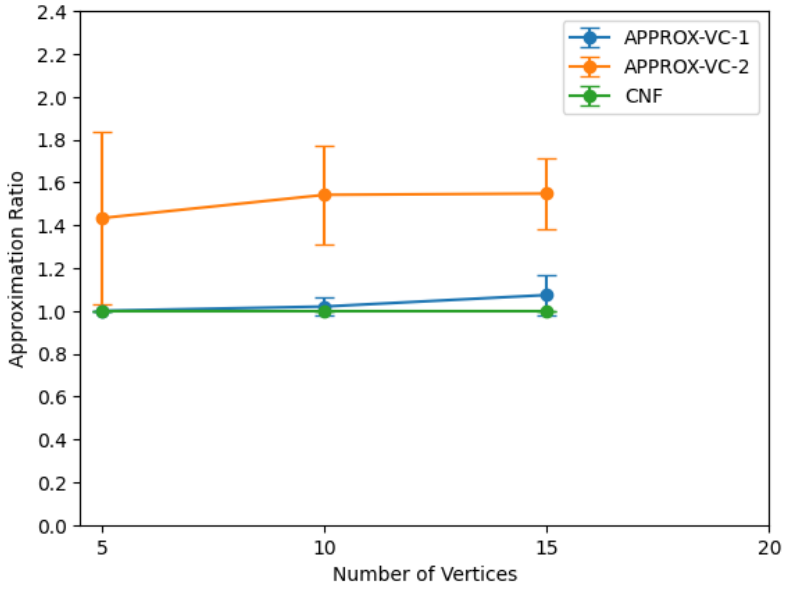

Figure 4: Approximate of APPROX-VC-1 and APPROX-VC-2 VS CNF

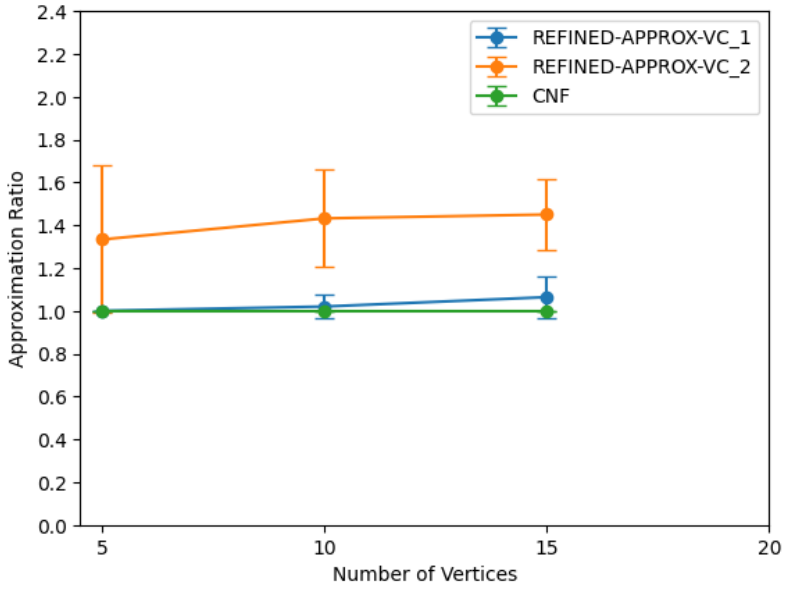

Figure 5: Approximate ratio of REFINED-APPROX-VC-1 and REFINED-APPROX-VC-2 VS CNF

We calculated the approximation ratios of APPROX-1, APPROX-2, REFINED–APPROX-VC-1, REFINED-APPROX-VC-2, 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 or 3 CNF SAT (both provide minimum cover)). The approximation ratio of these algorithms is shown in Figure 4 and Figure 5 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, 10, and 15.

- Since CNF-SAT-VC ensures finding the minimum number of vertices cover so 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/3 CNF-SAT.
- This is not the case for APPROX-VC-2. The approximation ratio shows it cannot always generate the minimum vertex cover accurately.
- The REFINED-APPROX-VC-1 further tries to be as close to CNF/3-CNF-SAT as compared to APPROX-VC-1 by removing vertices greedily.
- Similarly, The REFINED-APPROX-VC-2 further tries to be a little better and be further close to CNF/3CNF-SAT as compared to APPROX-VC-2 by removing vertices greedily.

6 Results

In general, the CNF-SAT, 3 CNF-SAT approach is the most effective of all to identify minimum vertex cover when the data scale is small,3 CNF-SAT is a little faster than CNF-SAT, but it is difficult to realize when there are more vertices and edges since the running time will increase greatly compared to the other four algorithms. The shortest running time is of APPROX-VC-2, although the approximation ratio reveals that it is not always correct in generating the smallest vertex cover. When the size of the data increases, APPROX-1 is also a more efficient option because it runs significantly faster and produces results that are extremely near the ideal ones in terms of approximation ratio. Using the refined approaches for APPROX-1, and APPROX-2, further lowers our vertex cover size and makes the approximate ratio close to 1 by greedily removing the vertices.

7 How we improved CNF SAT

Binary search is being used to find the minimum vertex cover in the graph. It is more efficient than linear search 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 **mid-point** 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 mid-point + 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.

8 References

1. Advanced Linux Programming, Mark Mitchell, Jeffrey Oldham, and Alex Samuel
2. Introduction to Algorithms, Second Edition ,Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest and Clifford Stein