A Quantitative Analysis of 3 methods to solve the Minimum Vertex-Cover Problem

Vyas Anirudh Akundy
Student ID:20765080

December 8, 2018

Contents

Abstract

This report presents a detailed analysis of 3 different approaches i.e 3 implementations to solve the *Minimum Vertex Cover* Problem using *C++* programming language.
The *Vertex Cover* Problem is stated in Section **1**. The aim is to get a minimum sized **Vertex Cover** The 3 approaches used to solve the *Vertex Cover* Problem are,

- A Polynomial time Reduction from VERTEX COVER to CNF-SAT
- A Greedy Algorithm
- An Approximate algorithm

Each of the approaches have been implemented using *C++* and have been analyzed for their efficiency using two characteristics: (a) Running Time and (b) Approximation ratio. A plot of these two characteristics against the number of vertices of the graph G has also been presented for each approach and a reasoning has been presented for each plot.

1 Vertex-Cover Problem

The *Vertex Cover* problem is described as follows:

A **vertex cover** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both. That is, each vertex "cover" its incident edges, and a vertex cover for G is a set of vertices that covers all the edges in E. The size of a vertex cover is the number of vertices in it[1].For the *Minimum Vertex Cover* Problem, we have to minimize the size of the subset V'. For example,
No. of Vertices $V = 5$, and the set of edges are given as,
$E = \{(2, 1), (2, 0), (2, 3), (1, 4), (4, 3)\}$.
Here the minimum vertex-cover is $V' = \{2, 4\}$ because vertices 2 and 4 cover all the edges of the graph. The following sections present a brief summary of the 3 different approaches.

2 A Brief description about the methods used

2.1 Reduction to CNF-SAT

A polynomial-time reduction from **Vertex Cover** to **CNF-SAT** has been done. The reduction takes as input the graph G and an integer k and produces a formula F. Then F is presented in the *Conjunctive Normal Form* as input to a **SAT**-solver. In this case, I have used the *Minisat* [2] **SAT**-solver. This **SAT** solver will give me a satisfying assignment for the formula F, that is does the graph G have a vertex cover of size k. This assignment will give me the answer to the instance of the Vertex Cover. In

the algorithm, I minimize the value of k, so that the output of the **SAT** solver will give me the solution to the *Minimum Vertex Cover* problem.

2.2 Approximate Algorithm(ApproxVC-2)

This algorithm is simple and straightforward, but may not always produce the optimal minimum vertex cover set. The algorithm is as follows:

- Pick an edge $(u, v) \in E$, and add both u,v to the vertex cover. Throw away all the edges incident on u and v.
- Repeat till $|E| = 0$

2.3 Greedy Algorithm(ApproxVC-1)

This is a greedy approach to the problem. This algorithm can be explained in two steps.

- Pick a vertex $v \in V$ with highest degree. Add it to the vertex cover set $V^{'}$ and throw away all the edges incident on v.
- Repeat till $|E| = 0$

The algorithm always greedily picks the vertex with the most incident edges and updates the set of edges continuously till there are no more edges. This algorithm can produce an optimal minimum vertex cover set for a higher number of cases when compared to the approximate algorithm, but it is not guaranteed to provide an optimal solution for every graph, as opposed to the CNF-SAT version which is always optimal.

3 Analysis of Graphs generated

The following graphs have been generated for each algorithm. Each graph depicts the Running time and the Approximation Ratio for each of the 3 algorithms mentioned above, versus the number of vertices. With the help of these graphs, we can quantitatively compare the efficiency of each method. The graph in Figure 1 depicts the running time for the **CNF-SAT** version. Since the **CNF-SAT** is an NP-Complete problem, for higher number of verties(in this case $V = 15$ and $V = 17$), the running time is very high when compared to the other two methods. I have not given any input beyond $V = 17$ as the CPU time is being exceeded for a size of $V > 17$. Though the output is always optimum, the SAT-solver will take time exponentially higher than the other two methods.

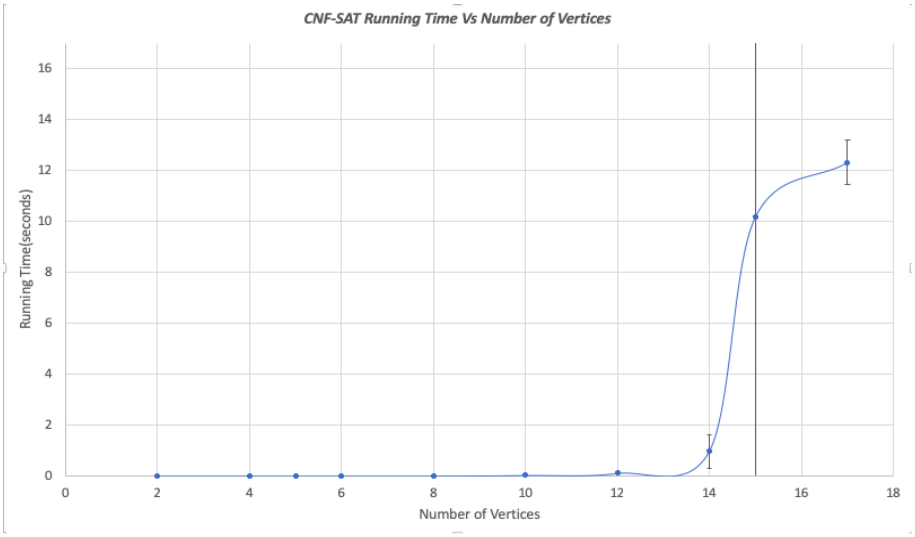

Figure 1: Running Time of CNF-SAT

The graph in Figure 2 depicts the running times for the Greedy and Approximate algorithms.

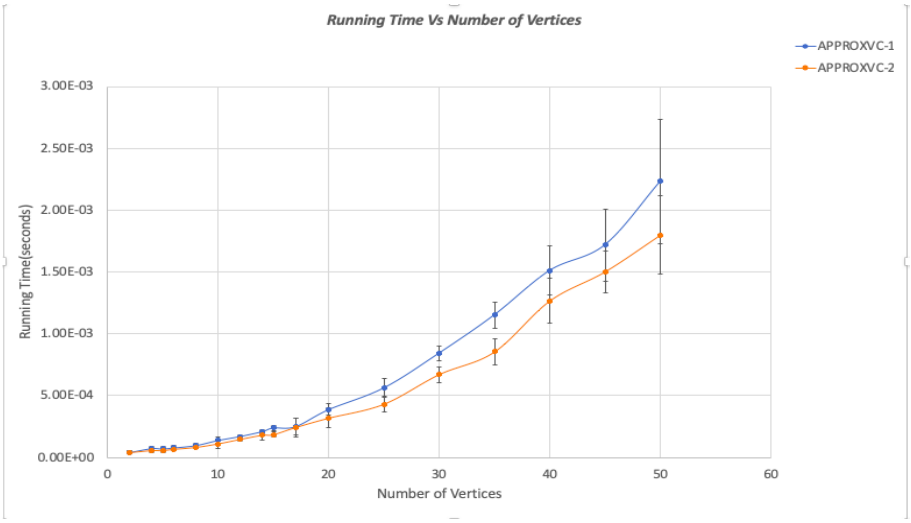

Figure 2: Running Times of Approximate Algorithms

Looking at the above trends for the Greedy approach(Approx VC-1) and the Approximate algorithm(Aprox VC-2), the Approx VC-1 takes slightly higher time than the Approx VC-2 method.
The way Approx VC-1 works is first it searches for the vertex of highest degree and then removes all its edges, i.e removes all the vertices adjacent to this vertex. Searching for the highest degree vertex runs in time Linear in size of the input. It has to scan the entire list of vertices in each run.
Next in my implementation, I have recursively called the function after deleting all the adjacent vertices, i.e I have reduced the size of the edge set E.
The running time can be expressed as follows:
If n represents the size of the input then,

$$T(n) = T(n - m) + \Theta(n)$$

Here $\Theta(n)$ is due to the time taken to search for the highest degree vertex. $T(n-m)$ represents the recursive call on the reduced size of the input. The size of the input is characterized as $|V|+|E|$. In the worst-case if $m = 1$ then the solution to the recurrence and the running time for my implementation comes out to be $\Theta(n^2)$, which explains why the trend for Approx VC-1 is not Linear.
However, in case of Approx VC-2, due to the absence of the "Search" for highest degree, it is slightly better than Approx VC-1.
The graph in Figure 3 shows the Approximation Ratios of the 3 techniques

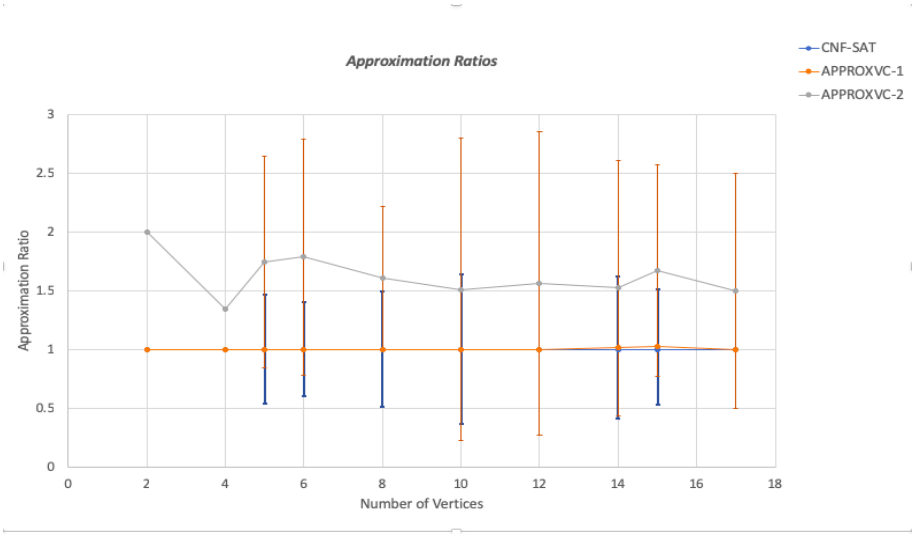

Figure 3: Approximation Ratios

Figure 4 shown here is just a closer look at the Approximation Ratio of the Greedy Algorithm as compared to the CNF-SAT version. From Figure 3, it can be said that, the ApproxVC-1 is more accurate in producing

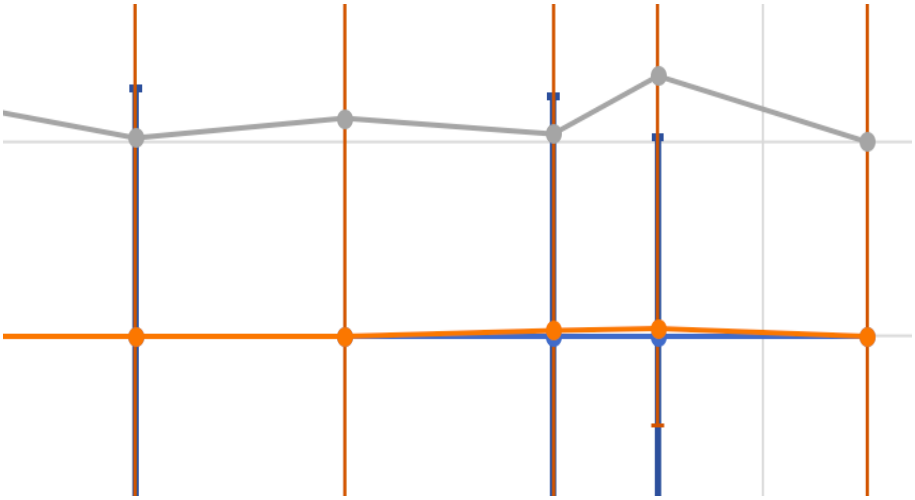

Figure 4:

the minimum vertex cover set as compared to the ApproxVC-2. The reason is that, in ApproxVC-2 we do not search for the highest degree vertex, rather a random edge is picked each time and both the vertices are added to the set. This random edge can result in picking the vertices with least degree thus resulting in the higher Approximate ratio as compared to the other two methods. For example in case the edges are $E = (0, 1)(2, 3)(4, 5)$, ApproxVC-2 will give all the vertices as the output when the optimal solution is $0, 2, 4$

4 Conclusion

In comparison, out of the 3 techniques, Reduction to CNF-SAT is yielding an optimal minimum vertex cover each time. Though at a higher number of vertices(higher input size), it is taking a huge amount of time to run as compared to the other methods since the SAT is an NP-complete problem. The Greedy algorithm(ApproxVC-1) is one that is near optimal as can be seen from the Approximation Ratio plots, whereas the Approximate algorithm(ApproxVC-2) is not as accurate as the ApproxVC-1 approach.
The two ApproxVC algorithms are faster than the CNF-SAT version since they run in polynomial-time in the size of the input. Between these two, ApproxVC-2 seems to run slightly faster than ApproxVC-1, because of the absence of the additional task of finding the highest degree vertex, however this depends on the representation or the encoding of the graph(as an adjacency list or adjacency matrix) and data structures used in the program.

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition.* 3rd. The MIT Press, 2009. ISBN: 0262033844, 9780262033848.

[2] *Modern SAT solvers: fast, neat and underused (part 1 of N).* URL: https://codingnest.com/modern-sat-solvers-fast-neat-underused-part-1-of-n/.