ECE-650 - METHODS AND TOOLS OF SOFTWARE ENGINEERING

UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Project Report on: Analysis of Three Algorithms for Solving MIN-VERTEX-COVER Problem

Authors:
Jannat Kaur (ID: 20779763)
Meet Raval (ID: 20833793)

Abstract

In this report, we consider the classic NP-complete problem to find the minimum vertex cover. We have used three different Algorithms (CNF-SAT-VC, Approx-VC1 and Approx-VC2) to find out the Minimum Vertex cover of a given graph. For input, we use the input graph generated by graphGen executable available at the ecelinux servers and then the output of each algorithm is compared with the other two on various inputs to compute the most efficient algorithm. Efficiency is based on their Running Times and Approximation ratio(the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover).

Keywords— NP-Complete, Vertex-Cover problem, Satisfiability problem.

1 Introduction

Vertex cover of a graph is defined as the set of vertices in a graph such that each edge is incident to at least one vertex of the set. A vertex cover of a graph G = (V, E) is a subset of vertices $C \in V$ such that each edge in E is incident to at least one vertex in C. Vertex-Cover is the following problem:
Input **:** An undirected graph G = (V; E), and an integer $k \in [0, |V|]$.
Output **:** True, if G has a vertex cover of size k, false otherwise.

Computation of the smallest vertex cover of any given graph is an NP-hard optimization problem. Three different approaches were used to find the minimum vertex cover, which are as follows:

1. CNF-SAT VC: This approach uses a polynomial-time reduction from VERTEX-COVER to CNF-SAT, and a SAT solver is used to get the output.

2. APPROX VC 1: In this approach, a vertex with the highest degree is picked and is added to the vertex cover. All edges incident on this vertex are thrown and this is repeated until no edges remain.

3. APPROX VC 2: A random edge (u, v) is picked and both u and v are added to the vertex cover. All edges attached to u and v are thrown. This process is repeated until no edges remain.

2 Implementation

We have used 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. In our case, it takes as input G, k and produces a Formula F with the property that G has a vertex cover of size k if and only if F is satisfiable. In the CNF SAT Satisfiability Problem, the Boolean formula is specified in the Conjunctive Normal Form (CNF), that means that it is a conjunction of clauses, where a clause is a disjunction of literals, and a literal is a variable or its negation.

We have utilized multi-threaded programming approach in which 4 threads were created: one for input/output, one for CNF SAT algorithm, one for Approx VC 1 algorithm and one for the Approx VC 2 algorithm. The user input from the I/O thread is passed to the other threads.

For the SAT solver, we have used MiniSAT solver. MiniSat is a minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT. It is easy to modify, highly efficient and is designed for integration. MiniSAT determines if it is possible to find assignments to boolean variables that would make a given expression true, if the expression is written with only AND, OR, NOT, parentheses, and boolean variables. If it is satisfiable, MiniSAT returns a set of assignments that make the expression true. Many kinds of problems can be broken down into a large SAT problem. All the threads run concurrently and we ensure that there is no racing condition between them. Moreover, mutex is applied on threads to ensure that the critical section is accessed by a single thread at a time.

We implement the three algorithms and analyze these functions in terms of efficiency, which is characterized in two ways: 1. Running time and 2. Approximation ratio. The final results were collected cumulatively and GnuPlot was used to analyze the obtained data. Analysis is given below.

3 Analysis

After implementing all three algorithms Approx-VC-1, Approx-VC-2 and CNF-SAT we have collected Runtime of the input of different Vertice sizes, and for each number of vertice we took 10 different Vertices and respective Edges generated by graphGen and for each vertices and edges generated by graphGen we ran the algorithm 10 times and took average and Standard Deviation to get the data. And we also calculated Mean Approximate Ratio. Now, let's analyze each plot obtained from the data we collected.

3.1 Run-time Analysis of CNF-SAT

In the CNF-SAT we only took number of vertices from 5 to 18, because after 18 number of vertices the CNF-SAT took more than 40 minutes to compute a solution and our program would time-out in that time period. The CNF-SAT algorithm is NP complete problem, and as we can see from the Figure 1 that run-time of the algorithm increases exponentially after $V = 14$ because number of clauses in SAT solver increases and because of that CPU has to process each clauses. From analysis of the Figure 1 and Figure 3 CNF-SAT algorithm we get the optimal output but the run-time of this algorithm is good for $V < 14$ but for number of vertices greater than 14 the run-time increases exponentially. Not only that as the number of vertices increases the Standard Deviation also increases by huge margins showing the inconsistency in the run-time as the number of vertices increases.

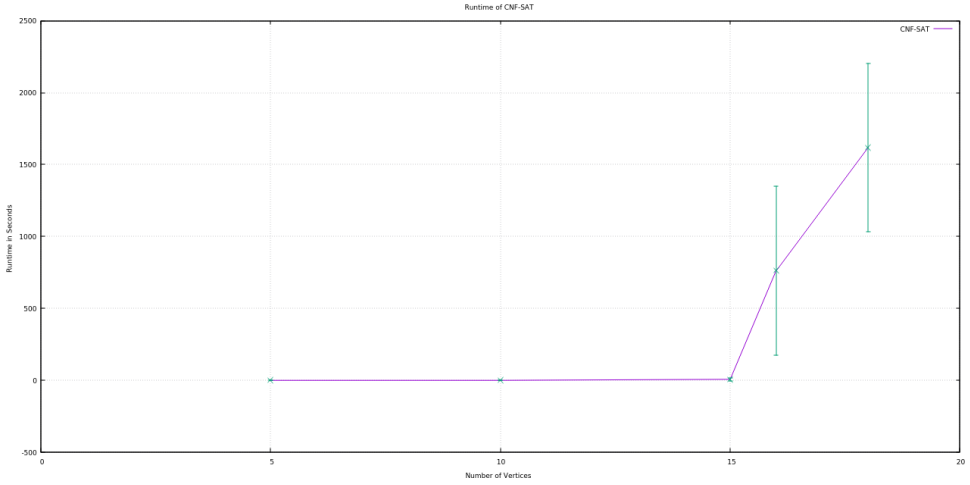

Figure 1: Run-time Plot of CNF-SAT

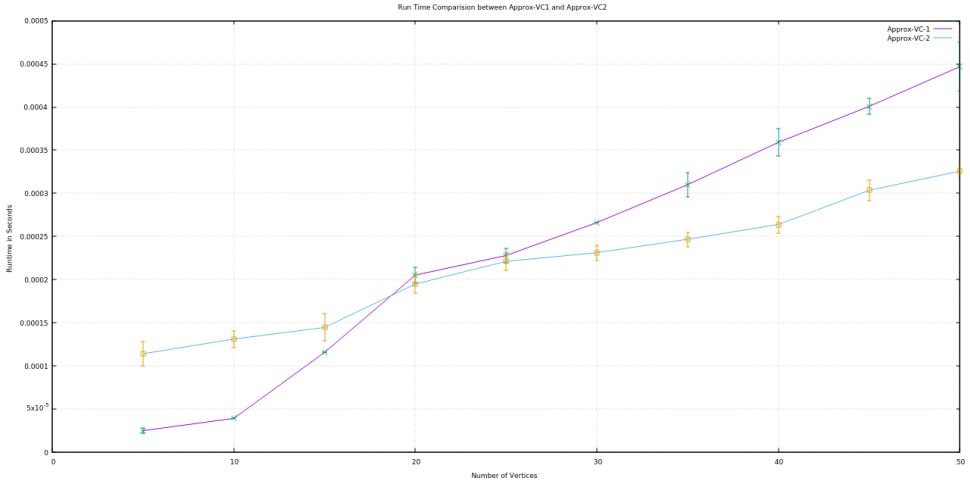

Figure 2: Average Run-time plot for Approx-VC-1 and Approx-VC-2

3.2 Run-time Analysis of Approx-VC-1 and Approx-VC-2

By analysing Figure 2, we can say that run-time of both approximation algorithm increases as the number of vertices increases. If the number of vertices are less than 25 then both the algorithms takes almost similar time to compute the Vertex Cover. But if number of vertices are more than 25 then Approx-VC-2 is faster compared to Approx-VC-1. But, Approx-VC-2 algorithm provides very low accuracy compared to Approx-VC-1. We don't see exponential increase in the runtime as seen in the CNF-SAT algorithm. Approx-VC-1 has slower run time compared to Approx-VC-2 as the number of vertices increases because as the number of edges increases it has to count the vertice with with the highest number of edges are connected and then add the vertice. So, Approx-VC-1 takes more time but it is good because it also provides better results compared to Approx-VC-2. Running time of the Approx-VC-1 algorithm can be written as O(V∗E) which is

polynomial to size of the input. Approx-VC-2 algorithm works faster because it will remove two edges each time it selects any random vertices, so we get vertex cover faster than the other two algorithm. In the Approx-VC-1 the standard deviation is almost none when number of vertices are less than 25 but as the number of vertices increases the standard deviation also increases. But, in the Approx-VC-2 algorithm the standard deviation remains almost the same overall.

3.3 Run-time Analysis of Mean Approximation Ratio of Approx-VC-1, Approx-VC-2 and CNF-SAT

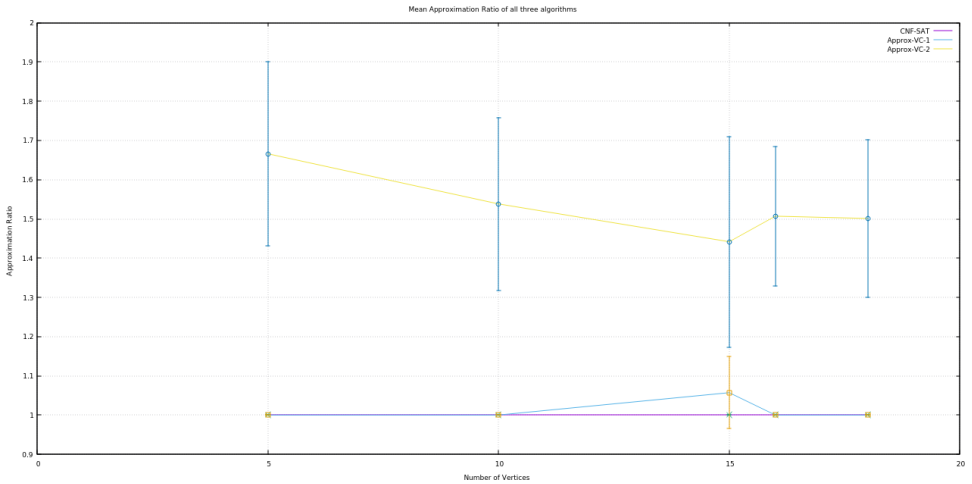

Figure 3: Mean Approximation Ratio of Approx-VC-1, Approx-VC-2 CNF-SAT

After analysing all three algorithms using rum-time as main parameter. we can say that Approx-VC-2 algorithm is the best algorithm because it gives us vertex-cover using the minimum amount of run-time. But, if we analyse Figure 3 then CNF-SAT being the optimal solution provides us with the true vertex cover everytime. Approx-VC-1 algorithm doesn't provides us with optimal solution always. and Approx-VC-2 algorithm has the worst approximation ratio compared to both the algorithm and the standard deviation of the same is very high, In the worst cases Approx-VC-2 provides output which is twice the size of the output provided by the CNF-SAT algorithm as we can see in the Figure 3. We can conclude that Approx-VC-1 algorithm has the best approximation ratio over Approx-VC-2 algorithm as Approx-VC-1 provides us near optimal solution near 1 in the polynomial time.

4 Conclusion

In terms of running time, APPROX-VC-2 performs best followed by APPROX-VC-1 and CNF-SAT-VC respectively. CNF-SAT-VC, being a NP-COMPLETE problem, fails to finish in polynomial time for larger inputs. Although it is the most optimal solution, its running time is exponential with the size of the input.

In terms of Approximation Ratio, APPROX-VC-1 is almost comparable to the CNF-SAT-VC. APPROX-VC-2, however, results in high mean approximation values and produces a larger vertex cover than the optimal solution in the vertex cover result. Thus, as far as Approximation Ratio is concerned, APPROX-VC-1 performs better than APPROX-VC-2.

When we compare efficiency both in terms of Running Time and Approximation Ratio, APPROX-VC-1 outperforms the other algorithms as it gives an almost optimal vertex cover in a very less time. APPROX-VC-2 algorithm has a good performance in terms of time but never generates an optimal vertex cover. CNF-SAT-VC approach guarantees an optimal vertex cover solution but fails to finish in polynomial time for higher values of vertex count.

Hence for any input graph, selecting APPROX-VC-1 to solve the vertex cover problem is preferable as it is better in terms of running time and in Mean Approximation Ratio but if we need a highly efficient and correct output, CNF-SAT VC methodology should be selected.