A Project Report on

Analysis of Three Algorithms for Solving MIN-VERTEX-COVER Problem

Course: ECE 650 “Methods and Tools of Software Engineering”

Submitted By

Kunal Taneja[1] & Prateek Gulati[2]

[1]Student number 20790967
WatIAM: k5taneja
Email: kunal.taneja@uwaterloo.ca

[2]Student number 20764145
WatIAM: p3gulati
Email: p3gulati@uwaterloo.ca

Faculty of Engineering

Department of Electrical and Computer Engineering

Fall 2018

ABSTRACT

In this report, we consider the classical NP-complete VERTEX COVER problem. We assume that the size and the access to the input graph derived (from graphGen executable available at ecelinux servers) impose the following constraints: (1) the input graph must not be modified (integrity of the input instance), (2) the computer running the algorithm has a memory of limited size (compared to the graph) and (3) the result must be sent to an output console once a new piece of solution is calculated. Despite the severe constraints of the model, we have used three different Algorithms (CNF-SAT-VC, Approx-VC1 and Approx-VC2) to find out Minimum Vertex cover of a graph and are then each algorithm is compared with other two based on their Running Time (by analysing mean and standard deviation over several runs) and Approximation ratio [1].

1. Introduction

In a graph theory, a vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. The minimum vertex cover problem is the optimization problem of finding a smallest vertex cover in a given graph. A vertex cover of a graph G can also more simply be thought of as a set S of vertices of G such that every edge E of G has at least one of member of S as an endpoint. The vertex set of a graph is therefore always a vertex cover. The smallest possible vertex cover for a given graph G is known as a minimum vertex cover. We have used three different approaches to reach to the conclusion of minimum vertex cover problem using a C++ language code. Three methods are as follows:

1. **Algorithm 1 (CNF-SAT-VC):** the approach uses a polynomial time reduction to CNF-SAT, and then use a SAT solver to get the final results.
2. **Algorithm 2 (APPROX-VC-1)**: the approach is to pick a vertex of highest degree (most incident edges) and add it to "vertex cover" container then throw away all edges incident on that vertex and repeat those steps till no edges remain.
3. **Algorithm 3 (APPROX-VC-2):** pick an edge <u, v> and add both u and v to your vertex cover and throw away all edges attached to u and v and repeat till no edges remain.

2. Program Design

We use a polynomial time reduction of vertex cover problem to CNF-SAT. Polynomial-time reduction is an algorithm that runs in time polynomial in its input. It takes as input G,K and produces a CNF formula that is it reduces VERTEX COVER problem to CNF-SAT problem. Satisfiability Problem (CNF-SAT) is a version of the Satisfiability Problem, where 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. So, in our program we make four threads, one for the user input/output which takes the input from the user end and continues running wait for the input until it sees end of file. The other threads are dedicated for each of the three algorithms that is for CNF-SAT-VC, APPROX-VC-1 and APPROX-VC-2. The user input from input/output thread is then passed to each of these threads.

In the thread for CNF-SAT-VC, we use MiniSAT solver. MiniSAT is a SAT solver which can determine 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, most SAT solvers (including MiniSAT) can also show a set of assignments that make the expression true. Many problems can be broken down into a large SAT problem. All the threads run concurrently with respect to each other and we ensure that there is no racing condition between them. Moreover, we apply mutex on threads so as to ensure mutual exclusion such that critical section is accessed by a single thread at a time.

[1] Approximation Ratio = $\frac{\text{size(computed vertex cover)}}{\text{size(optimal vertex cover)}}$

We design a multi-threaded architecture for the project where the main function creates an I/O thread which accepts the input from the keyboard. The I/O thread runs and waits for the input. In the I/O thread, after getting the input, the program creates three algorithm threads. At that time, the four threads run concurrently. When these three algorithm threads finish, the I/O thread get the results and displays them correctly.

Next, we implement the three algorithms and analyse these functions in terms of efficiency. Considering the traditional ways of representing graphs, we choose the adjacency list to store the information of the input graph. After parsing the input <V, E> pairs, the program initializes the graph by initialising the adjacency list. We build three functions to realize the three algorithms including CNF-SAT-VC, APPROX-VC-1 and APPROX-VC-2. Each function returns a vector to pass the final results (vertex cover) to the I/O thread function.

3. Data Analysis of Three Algorithms

For analysis of algorithms, we are considering Running Time (time taken by an algorithm to output result i.e. vertex cover in our case) and approximation as the parameters to find the efficiency of our algorithms. We found that there was a huge difference between the running times for APPROX-VC-1/VC-2 and CNF-SAT, we made two different plots for run time. One plot shall compare running time between VC-1 and VC-2 approach. The other plot will show the mean time and standard deviation for CNF-SAT approach.

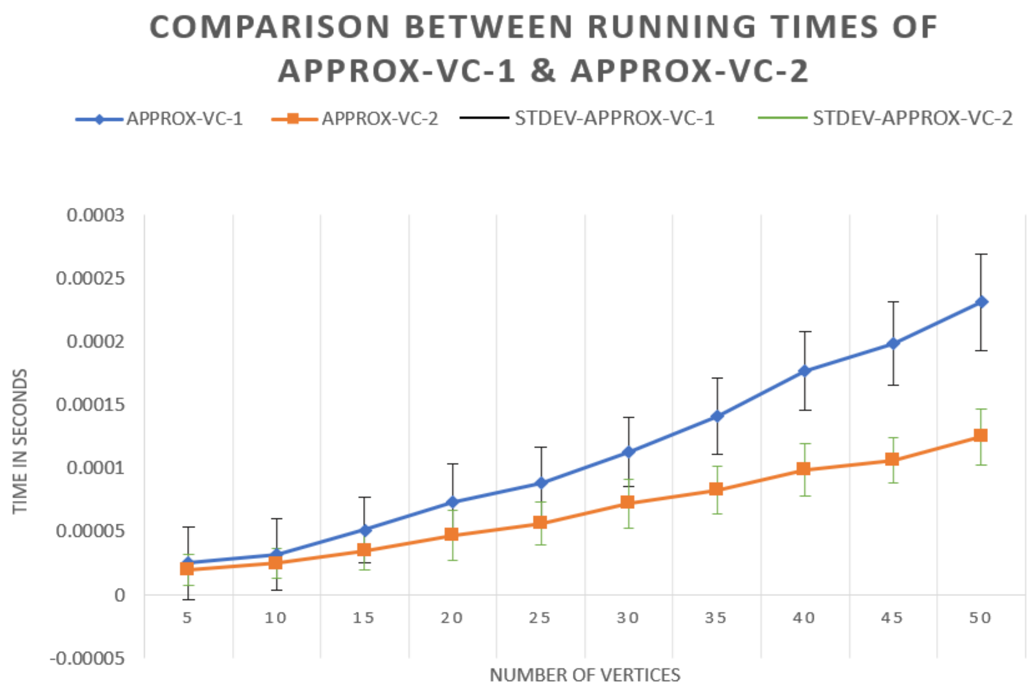

Figure 1: Mean Run time (in microseconds) plot for APPROX-VC-1 and APPROX-VC-2

As we can see in Figure 1, for each value of vertex count |V|, the performance of APPROX-VC-2 is better than that of APPROX-VC-1. This increased mean run time for APPROX-VC-1 can be attributed to the making greedy choice of the vertex with highest degree at each step for finding the vertex cover, which is an additional time intensive component of this algorithm. APPROX-VC-2, on the other hand, simply picks a random vertex at each step without any restriction and hence is faster.

In general, there is a polynomial increase in time for both APPROX-VC-1 and APPROX-VC-2 approaches. We observe a higher standard deviation for |V| < 30 in the mean running time for APPROX-VC-1 approach as compared to APPROX-VC-2 approach. The high standard deviation in the case of APPROX-VC-1 approach could be due to deviations in different run times (which includes time intensive computation of the highest degree vertices and which may vary with different graphs for the same vertex count) for each of the 10 distinct graphs for every vertex count. On the other hand, in the case of APROX-VC-2 approach, for smaller vertex counts, almost all edges in the graph would be removed in no time (as the vertices are randomly selected and removed without any restriction) causing the thread to end with almost same times. For |V| ≥ 30, both the algorithms have a high standard deviation because we have used completely random 10 graphs to compute the run times. Each graph might be having a different run time and hence contributes to standard deviation.

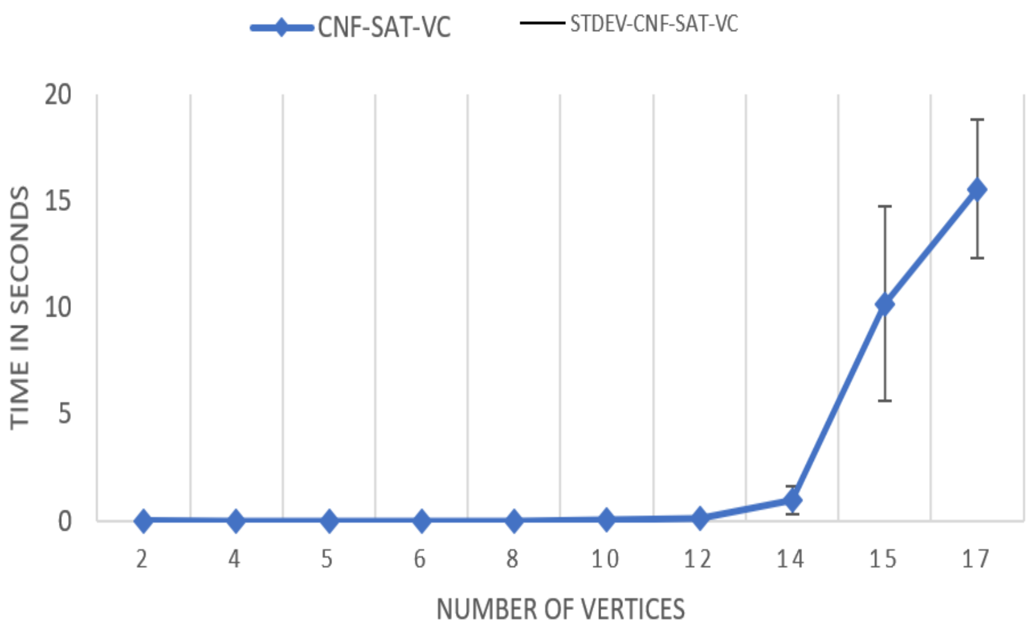

Figure 2: Mean Run time (in milliseconds) plot for CNF-SAT-VC

By analysing this graph, we can say that CNF-SAT-VC is efficient for less number of vertices and the running time is less as compared to the running time if we give more number of vertices. When number of vertices increase, we can see an exponential growth in running time. We expect this growth will continue if we run SAT solver for graphs with larger number of vertices. This is because as number of vertices(v) increases and it will increase the number of clauses for the CNF-SAT and with increased edge numbers it has to satisfy more clauses. Thus, it leads for higher running times as CPU must process more data and there will be dedicated cost for each operation. In case of small number of vertices, CNF-SAT-VC produces less number of clauses so, it takes less time as it must satisfy less number of combinations of variables. Since CNF-SAT is an NP-Hard problem, the run time increases a lot after |V| = 14. The algorithm was taking a lot of time to run for |V| > 17 and was leading to time outs. Hence, we performed analysis till a maximum |V| = 17. This algorithm takes a lot of time as compared to APPROX-VC-1 and APPROX-VC-2 approaches.

The huge difference in the time taken by CNF-SAT-VC as compared to the other two algorithms is because of the time taken by Minisat to solve the CNF-SAT polynomial reduction of the vertex cover. For |V| ≥ 15 we notice very high standard deviations. This may be because varying vertex cover size for the same vertex count will result in different run times. Since the run time for CNF-SAT-VC for higher vertex counts is high, it leads to higher standard deviations.

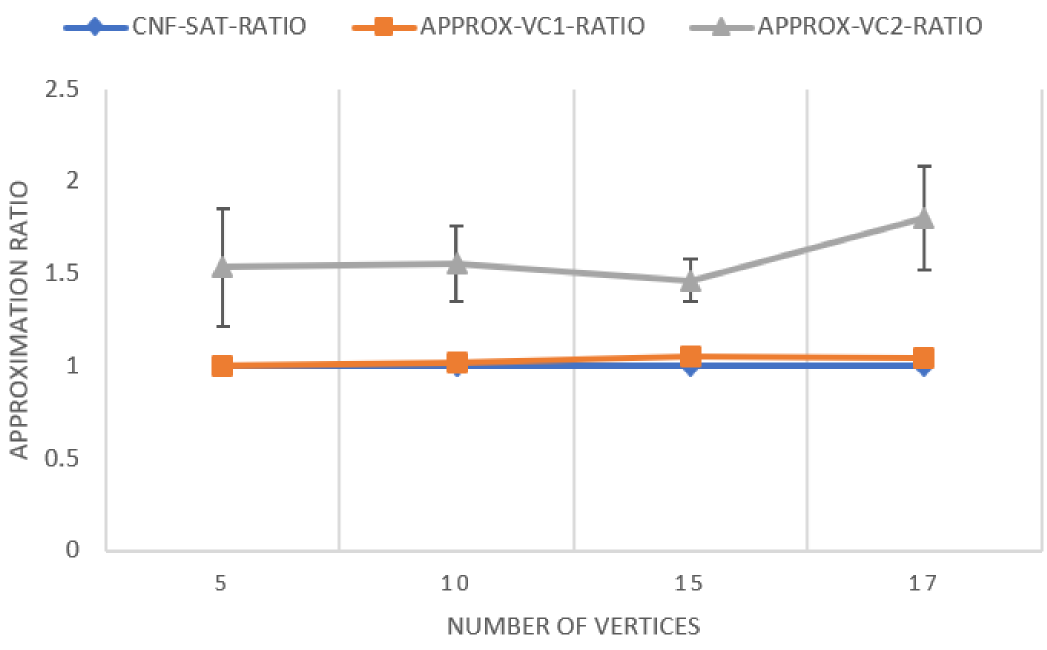

Figure 3: Mean Approximation Ratio plot for APPROX-VC-1 and APPROX-VC-2

Figure 3 plots the approximations ratio of APPROX-VC-1 and APPROX-VC-2 approaches as compared to CNF-SAT approach. We notice that the mean approximation ratios for APPROXVC-2 approach is always more as compared to that of APPROX-VC-1 approach. This is because APPROX-VC-2 approach depends on randomness in terms of Vertex selection and hence deviates from the optimal vertex cover solution. APPROX-VC-1 approach, on the other hand, computes the Vertex with the highest degree at each step and hence achieves a solution which is much closer to the optimal solution.

APPROX-VC-2 approach, however, always results in more vertices in the vertex cover as compared to the optimal vertex cover. We notice that APPROX-VC-2 has higher standard deviations as compared to APPROX-VC-1. This is because APPROX-VC-2 algorithm has a random component in vertex selection and provides vertex cover results based on the selection (which has no sense of optimality). This random selection in different graphs for the same vertex count may give different approximation ratios. Since, APPROX-VC-2 follows a probabilistic model, high standard deviation is expected.

4. Conclusion

According to efficiency 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 $|V| > 17$. With respect to efficiency in terms of Approximation Ratio, APPROX-VC-1 is almost comparable to the optimal solution (CNF-SAT-VC). APPROX-VC-2, however, always results in high mean approximation values and produces higher number of vertices than the optimal solution in the vertex cover result. Thus, as far as Approximation Ratio is concerned, APPROX-VC-1 approach performs better than APPROX-VC-1.

If we look at efficiency in terms of both Running Time and Approximation Ratio, APPROX-VC-1 approach outperforms both the other approaches as it gives almost optimal vertex cover solution in very less time. APPROX-VC-2 approach performs good in terms of time but never results in 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.