# Analysis of Three Methods of Solving MIN-VERTEX-COVER Problem

Ye Wang and Yuqing Zhao

Dept. of Electrical and Computer Engineering, Student number 20730349

Dept. of Electrical and Computer Engineering, Student number 20702958

## ABSTRACT

In the mathematical discipline of 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. A minimum vertex cover is a vertex cover of smallest possible size. The problem of finding a minimum vertex cover is a classical optimization problem in computer science. In this report, we will use three different approaches to solve the minimum vertex cover problem by using C++ program which has 4 threads, one for I/O, and one each for the three different algorithms. The result shows the most effective way to solve minimum vertex cover.

**Keywords:** minimum vertex cover

## 1. INTRODUCTION

There are three methods. As the following, 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. 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. 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.

After implementing those three algorithms, we analyze those the efficiency of those three algorithms by two factors: (1) running time (2) approximation ratio. We characterize the

approximation ratio as the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover.

## 2. RELATING METHOD

MiniSat: 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 (perhaps with thousands of variables), so SAT solvers have a variety of uses.

Multithreading: multithreading is the ability of a central processing unit (CPU) or a single core in a multi-core processor to execute multiple processes or threads concurrently, appropriately supported by the operating system. If a thread gets a lot of cache misses, the other threads can continue taking advantage of the unused computing resources, which may lead to faster overall execution as these resources would have been idle if only a single thread were executed. If several threads work on the same set of data, they can share their cache, leading to better cache usage or synchronization on its values.

## 3. EXPERIMENTAL DESIGN

At the first step, we design the thread architecture of the project. In the main function thread, the program creates the I/O thread which can accept the input from the keyboard. The main thread dies and the I/O thread still run and always waiting 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 print them out.

At the second step, we implement the three algorithms and realize these functions more efficiently. Considering the two traditional ways of presenting graphs, we select the adjacency list to store the information of the input graph and only use this adjacency list to store information. After parsing the input V E pairs, the program initializes the graph by 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.

At the third step, we consider more specifically. How to pick edges randomly in algorithm 3 is the difficult part for us to think about. In our program, we use a random number to pick a vertex in adjacency list to pick an edge randomly. Because of the while loop to find the random vertex, we are afraid of the way we pick edges will influence the running complexity of the algorithm. In the following part, we will analyze the efficiency results of those three algorithms.

## 4. EXPERIMENTAL RESULTS AND ANALYSIS

In the first part, we will analyze running time of these three algorithms. Because there is a huge difference between CNF-SAT and other two algorithms in running time, we separate this to two figures 4.1 and figure 4.2. As can be seen in figure 4.1 and figure 4.2, running time of all increases as the number of vertices rises. This may be because CPU has to process more data and cost more time.

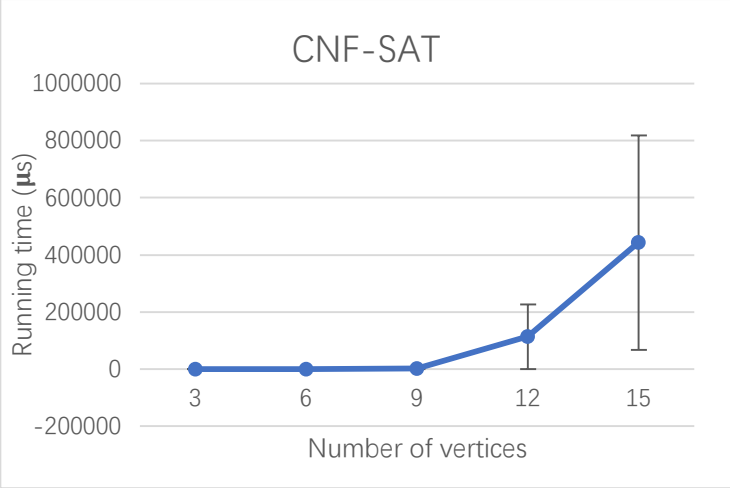


figure 4.1 the relationship between running time and Number of vertices in CNF-SAT algorithm

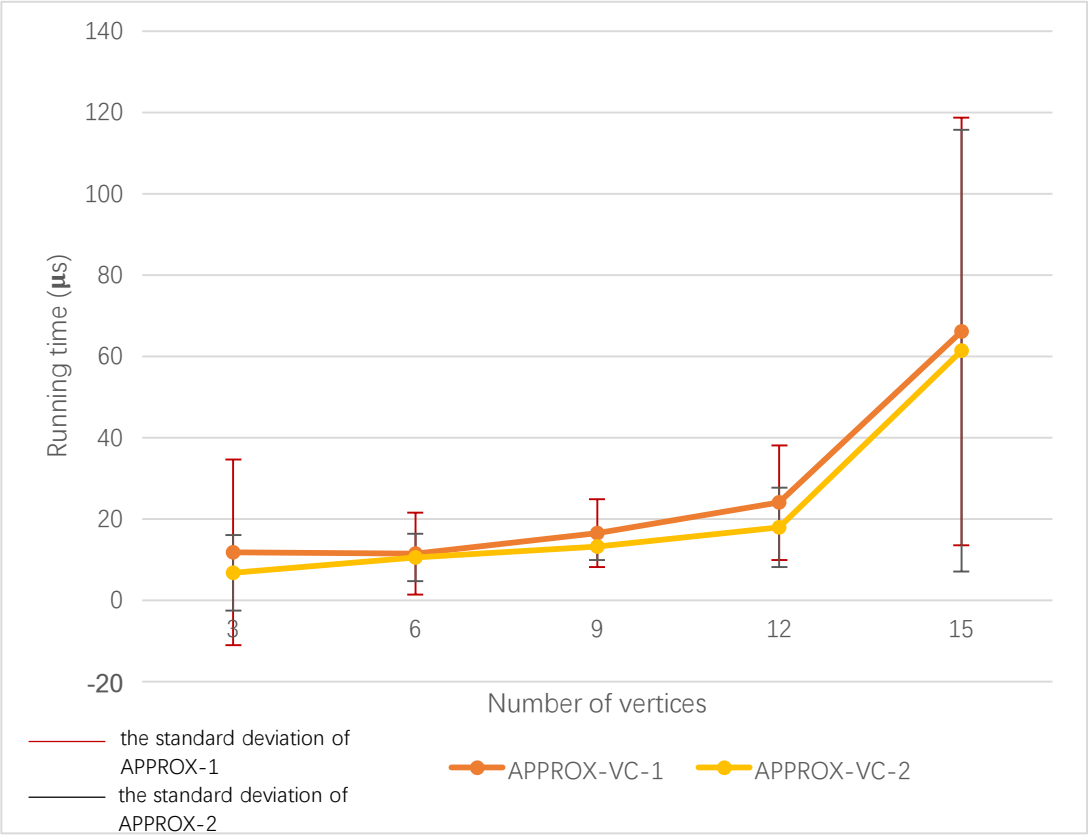


figure 4.2 the relationship between running time and Number of vertices in APPROX-VC-1 algorithm and APPROX-VC-2 algorithm

Overall, compared to the three algorithms, the running time of CNF-SAT is the highest and much higher than other two algorithms. The reason may be, in the implement of CNF-SAT algorithm, in order to add clauses to finish the reduction to CNF-SAT and get the minimum vertex cover, there are lots of for loops and one while loop. Except for the running time taking by the SAT solver, the time complexity is $\Theta(n^4)$ which is much bigger than the other two algorithm functions. This cause the running time more complexed. At the same time, SAT is NP problem, which means that the result from the SAT solver cannot be gotten in polynomial time. The running time of APPROX-VC-1 is a little higher than the running time of APPORX-VC-2. The reason may be finding the highest degree vertex cost more time compared to only picking edge without restriction. It seems that the complexity of APPROX-VC-1 is much higher.

In CNF-SAT algorithm, the data fluctuates greatly when the number of vertices goes up. In the all three algorithms, when the number of vertices is up to 15 which is that biggest figure, the running time of all fluctuate dramatically. A big characteristic is in the APPROX-VC-1

and APPROX-VC-2 is that the fluctuation in APPROX-VC-1 is much bigger than that in APPROX-VC-2. This is probably because APPROX-VC-1is a greedy algorithm which implies in finding a highest degree vertex.

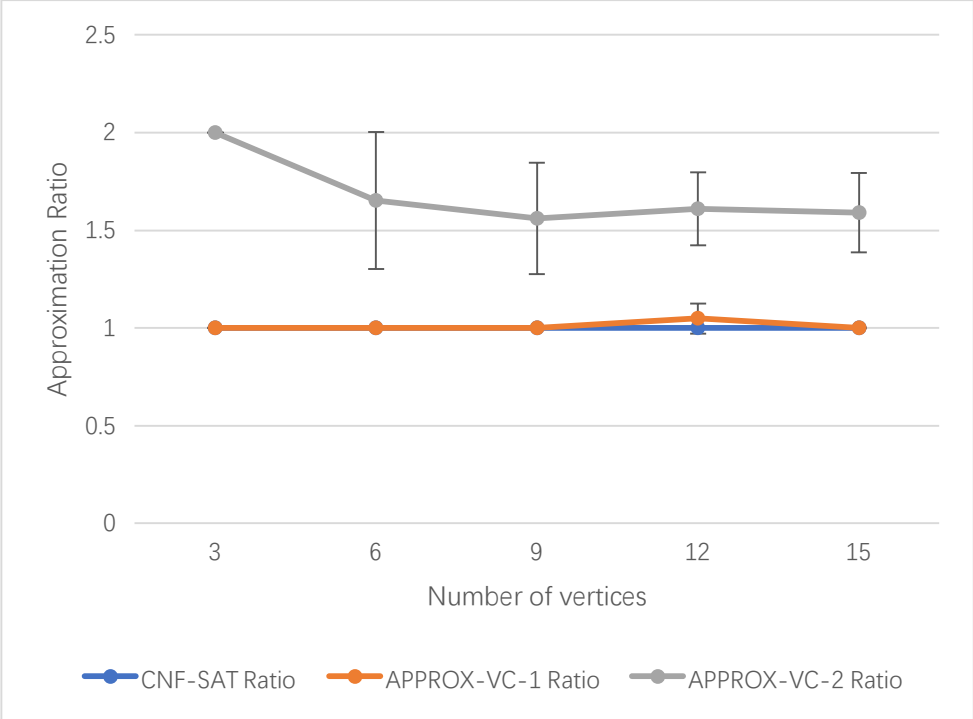


figure 4.3 the relationship between approximation ratio and the number of vertices in the three algorithms

In this part, we will analyze the difference between these three algorithms in approximation ratio (the ratio of the size of the computed vertex cover to the size of a minimum-sized vertex cover). We regard the result of CNF-SAT as the minimum-sized vertex cover. Figure 4.3 shows apparently that the ratio of APPROX-VC-1 is 1 at the most of time, which means this algorithm likely give the correct minimum size of vertex cover at the most of time. The ratio of APPROX-VC-2 is much higher than APPROX-VC-1. It shows that the probability of giving a correct minimum vertex cover for APPROX-VC-2 is much higher than APPROX-VC-1.

## 5. CONCLUSION

In this work, we studied and compared the three different approaches of solving the minimum vertex cover problem.

It was found that the running time of each approach was increased by the incremental numbers of vertices. If we just concentrated on the section of running time, we can find that the total running time of CNF-SAT-VC method is larger than that of other two approaches, and then followed by APPROX-VC-1, the least running time is in APPROX-VC-2 method. It means that the APPROX-VC-2 approach is the most effective way to compute the mini-vertex cover problem. But when we thought about the approximation ratio, it can easy find that APPROX-VC-2 algorithm maybe fail more unlikely to produce the precise answer to mini-vertex cover problem.

Future research will focus on finding the better way to solve the related vertex cover problem.

## REFERENCES

[1] Handout of Assignment 4

[2] Handout of Project

[3] Introduction to Algorithms. Third Edition. Thomas H. Cormen. Charles E. Leiserson. Ronald L. Rivest. Clifford Stein.

[4] https://www.dwheeler.com/essays/minisat-user-guide.html