ECE650 Project Report

Presented by

Zhehao Xu
20762577
z352xu@uwaterloo.ca

Yongjian Zeng
20773103
y83zeng@uwaterloo.ca

9th Dec 2018

University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

- **Introduction**

In this project, we focus on solving the vertex cover problem using three algorithms and compare them in respect of running time to see which one is the most efficient. In order to achieve the goal, we firstly build CNF-SAT-VC, APPROX-VC-1, and APPROX-VC-1 independently. Then, 4 threads are created to make them run the input graph at the same time while recording time it takes for each algorithm to output the result. One for the general data input and output communication, another three for three individual algorithms. Finally, plots that compare the average time can be generated. We use the following 5 cases for the input tests which are 5, 10, 15, 17, 18 vertices graphs. Furthermore, we have also included one 20 vertices graph sample in the project.

- **Approaches**

1. CNF-SAT-VC:

The CNF algorithm converts the vertices cover problem into a conjunctive normal form (CNF) and then determine whether this graph is satisfiable or not. Selecting different vertices cases to find the minimum vertices required to satisfy the CNF.

2. APPROX-VC-1:

This approximation method selects the vertex that has most incident edges first. Add it to the vertex cover set and eliminate all incident edges. Repeating this above process until no edges remain.

3. APPROX-VC-2:

The algorithm takes two vertices in the graph randomly, adding them to the vertex cover set and throw all the edges containing these two vertices. Repeating the above process until no edges remain.

- **Result & Discussion**

i.General running time result

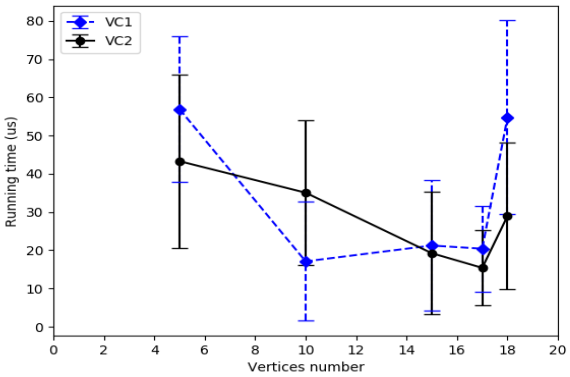


Figure 1. Running time plot for VC-1 & VC-2

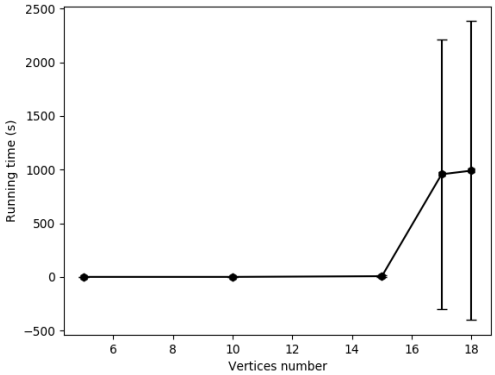


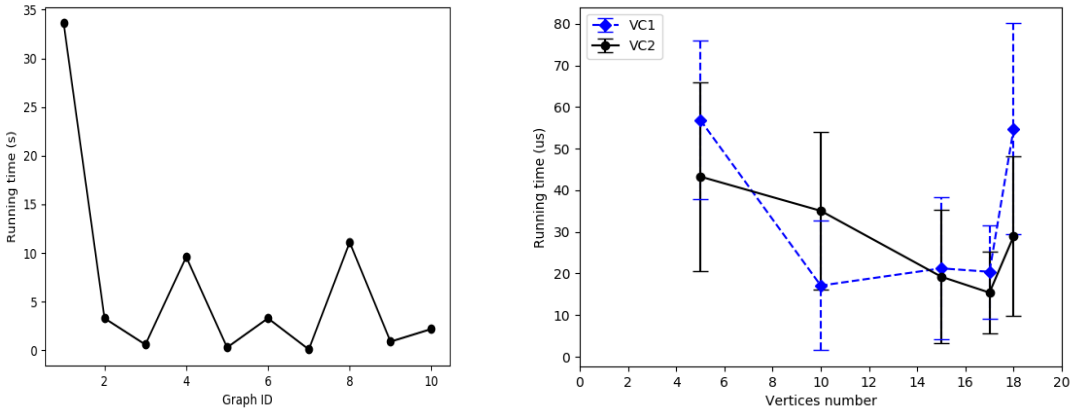
Figure 2. Running time plot for CNF-SAT-VC

The above figures are plots of the running time for three algorithms. Due to the long running time of CNF-SAT solver, we separate the other two approximation algorithm from it to get a clearer view. Observing figure 1, both APPROX-VC-1 (VC1) and APPROX-VC-2 (VC2) have similar behaviors. For the first case, 10 sets of 5 vertices graph are fed and VC1 uses 0.055 seconds on average while it only takes VC2 0.045 seconds. With the increasing number of vertices in the graph, both of them actually spend less time to process. For 10 vertices graph cases, VC1 decreased its processing time significantly

down to 0.02 seconds and VC2 decreases to near 0.037 seconds. For 15 and 17 vertices cases, the running time of VC2 continues to decrease near to 0.017 seconds. On the other hand, VC1's running time increases to 0.022 seconds. Finally, 18 vertices graphs produce a large increase in running time which VC1 spends almost the same amount of time as processing 5 vertices graphs. Comparing to VC1, VC2 does not get increased that much which reaches a value of 0.028 seconds.

Figure 2 is the running time plot of the CNF-SAT-VC, it is separated from VC1 and 2 due to its long processing time. For cases 5 to 15 vertices graphs, the running time is quite short, so they are close to zero showing on the plot. In fact, the average running time for 10 vertices is 0.007 seconds and 6 seconds for 15 vertices. Beyond 15 vertices, the time is increased significantly. When there are 17 vertices in the graph, the solver requires in average 16 minutes with a deviation of 16 minutes also. The same pattern is found when the number of vertices is increased to 18, the average time remains at 16 minutes while the deviation becomes even larger.

ii. CNF-SAT-VC analysis



(a) (b)

Figure 3. (a) Running time for graphs with 15 vertices (b) Running time for graphs with 17 & 18 vertices

The above figures give the details of running time of the CNF-SAT-VC algorithm. Figure 3 shows the running time of graphs with 15 vertices and the cases of 17 vertices and 18 vertices. In figure 3(a), we can see that some graphs need less than 1 second to calculate while some graphs need more than 30 seconds. Comparing to graphs with less vertices, the time difference between different 15-vertices graphs become much larger. If we increase the vertices number to 17 or 18, the standard deviation will increase dramatically. And some graphs need more than 3000s to calculate while some graphs only need several seconds.

Table 1 show the standard deviation of running time. It is obvious that standard deviation increases greatly as the number of vertices rise. There are two reasons for it:

1. As the graph become more complicated, the complexity of the SAT formula will improve too. So, it takes more time to work out the assignment. Because the average running time increase, the standard deviation will also increase.

2. Minisat use DPLL algorithm to solve the SAT problem. In DPLL algorithm, it firstly tries to apply unit propagation and pure literal rule to calculate the value of some atomic formula. And then if no atomic formulas can get value by these two methods, it starts to guess the value. So most of the atomic formulas in graph that need very little time (e.g. Graph 3) can be calculated by unit propagation and pure literal rule, then the algorithm just needs to do very little guessing. However, in some graphs, unit propagation and pure literal rule can only play a small role and most of atomic formulas need to be guessed. Assume that after applying unit propagation and pure literal rule, there are n atomic formula that don't know the value. If DPLL guess the answer for these formulas, it needs to try 2^n assignments in the worst case. It increases exponentially. So, in graphs with few vertices, it needs less time for guessing even if the unit propagation and pure literal rule is useless. However, in complex graph, the running time will grow significantly if unit propagation and pure literal cannot work.

From above, we can conclude that because time complexity of DPLL algorithm is O(2^n), the running time of CNF-SAT-VC will rise more significantly as number of vertices grow comparing to APPROX-VC-1 & 2. And for different graphs, even with same number of vertices, the running time difference may be huge.

Number of vertices	5	10	15	17	18
Standard deviation(s)	0.0001457	0.00538	9.77599	1254.57462	1392.3526

Table 1. Standard deviation of the CNF-SAT-VC

iii. APPROX-VC-1 & 2 analysis

Figure 1 shows that in most of the case, APPROX-VC-1 approach needs more time than APPROX-VC-1. It is normal because in APPROX-VC-1, it needs to find out the vertex that with biggest degree, and then remove all its related edges. Assume the graph with E edges, in the worst case, each time it finds the biggest-degree vertex, it can only remove one edge. In order to find out the biggest-degree vertex, we need to traverse all the edges, then it need to traverse all the edges again to remove them. So, we can calculate that the running time it needs is:

$$n = (E + E - 1 + E - 2 + \ldots + 1) + (E - 1 + E - 2 + \ldots + 1)$$

$$= O(E^2)$$

APPROX-VC-2 approach randomly picks an edge <u,v> and remove all the edges that related to u and v. So, in the worst case, after it pick a edge, it can remove only one edge. So, the running time is:

$$n = (E - 1 + E - 2 + E - 2 + \ldots + 1)$$

$$= O(E^2)$$

The time complexity of APPROX-VC-1 and APPROX-VC-2 are both O(E^2) so they both need less time than CNF-SAT. However, it is obvious that for same graph, APPROX-VC-1 is likely to spend more time than APPROX-VC-2. Because in APPROX-VC-2, it doesn't need to find out the biggest-degree vertex and it remove edges related to two vertices in each iteration.

However, it is abnormal that the graph with 5 vertices cost more time than graphs with more vertices. Table 2 shows some detailed APPROX-VC-2 running time of several graphs. We can see that even consider same graph, the running time will be very different. For example, when calculate the graph

with 5 vertices, sometimes it needs more than 60 us while sometimes it needs only 23 us. And the it takes more time to calculate the 5-vertices case.

#vertices running time(ns) count	1	2	3	4	5	6	7	8	9	10
5	23145	69863	69899	70451	64935	24612	65106	72236	69095	23242
10	10258	6646	6468	41857	47625	48291	6914	49720	7050	49648
15	54061	51755	46267	14170	11378	9997	10413	11403	56150	55135

Table 2. Detailed APPROX-VC-2 running time of several graphs.

Table 3 shows some detailed APPROX-VC-2 running time of same graphs without creating CNF-SAT thread. The data is more stable than the data above.

#vertices running time(ns) count	1	2	3	4	5	6	7	8	9	10
5	18746	15755	15893	15416	15673	15524	15770	15502	15730	15680
10	27453	18537	17956	17795	18041	17958	17966	18102	17875	17805
15	20800	19570	20429	18606	18722	18475	18625	18653	18492	18657

Table 3. Detailed APPROX-VC-2 running time of same graphs without creating CNF-SAT thread

So, we can see that if we run the CNF-SAT thread, it will influence the running time measurement. We do not know the exact reason. But we think the reason for this are:

1. When running the code, the RAM, numbers of thread running in the background, Os scheduling always change. All of these influence the measurement of running time. So, if the code run for too long, the measurement may be not accurate.

2.The processor supports dynamic frequency adjustment and will downgrade if the load is not heavy.

3.The value of running time is too short so it is more easily be influenced.

iv. Approximation ratio

CNF-SAT approach always find the minimum vertex cover. And APPROX-VC-2 will remove an edge each time and add two vertices. We analyzed that in the worst case it can remove one edge in each iteration. Then it is obvious that the size of vertex cover calculates with APPROX-VC-2 is at most twice the size of that calculate with CNF-SAT. Because consider the edges selected by the algorithm, we denote it as A. In order to cover this edge, the minimum vertex cover must include at least one endpoint of each edge in A. And no two edges in A share same endpoint because we will delete all related edges once we pick it. Thus, no two edges in A are covered by same vertex in the minimum vertex cover. So, the approximation ratio of APPROX-VC-2 ≤ 2. It is a 2-approximation algorithm.

APPROX-VC-1 approach find the biggest-degree vertex and add it to vertex cover set, we can prove that it is O(logn) approximation.

v. 20-vertices graph

We tried to compute the vertex cover of two graphs with 20 vertices by CNF-SAT-VC. One graph with 30 edges ran for more than 15 hours and has no result. However, another graph with 30 edges ran for 2409 s computed with CNF-SAT-VC and the running time for APPROX-VC-1 and APPROX-VC-2 is 35.7 us and 50.1 us.

The outcome shows that the growth rate of running time of CNF-SAT-VC is huge because it is an exponential-time algorithm. And The running time for APPROX-VC-1 and APPROX-VC-2 have hardly changed because they are polynomial-time algorithm. And the outcome shows again that for different graphs, even with same number of vertices, the running time difference may be huge.

- ## Conclusion

In a conclusion, there are two patterns found in this project regarding the growth rate and minimum vertices found. For CNF-SAT-VC, the rate of growth in respect of running time is exponentially increasing, the larger the input graph, the longer it requires in order to generate the outcome. Furthermore, time even for the same number of vertices input can vary a lot which results in a large deviation value. On the other hand, APPROX-VC-1 and 2 have a polynomial growth rate instead of exponential and this sacrifices the accuracy as a trade-off. Those two algorithms may not find the minimum vertices required to cover all edges but at maximum not exceeding twice of the minimum number of vertices.