THREE *Vertext* COVER ALGORITHMS ANALYSIS

Quanyu Wang
Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada
q426wang@uwaterloo.ca

Run Zeng
Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada
r24zeng@uwaterloo.ca

December 1, 2019

ABSTRACT

Our project mainly implements three algorithms to solve *minimal Vertex Cover* problem, one algorithm is reduced to boolean satisfiability problem (SAT) called **CNF-SAT-VC**, another two are approximate algorithms called **APPROX-VC-1** and **APPROX-VC-2** respectively. We collected results and analyzed their performance according to running time and approximation ration. From the tread-off strategy's aspect, APPROX-VC-1 perform the best.

***K*eywords** Vertex Cover · Reduction · CNF-SAT-VC · APPROX-VC-1 · APPROX-VC-2

1 Introduction

Vertex Cover is a classical graph problem in algorithms. In detail, given a undirected graph, it can find a minimum size set of vertexes to cover all edges[1].

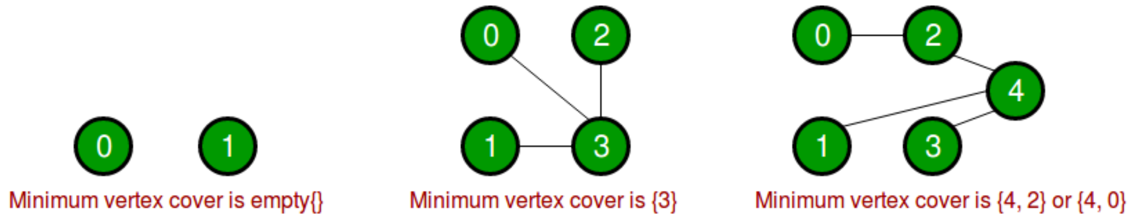

0
1
Minimum vertex cover is empty{}
0
2
1
3
Minimum vertex cover is {3}
0
2
4
1
3
Minimum vertex cover is {4, 2} or {4, 0}

Figure (1) Vertex Cover examples [1]

It is known as a NP complete problem, which means that its instance can be verified in poly-nominal time and can be reduced to a NP-hard problem. In our project, we reduced it to SAT problem. It is easier to regard SAT as a black box, we transfer vertex cover to a few propositional logic clauses as input of SAT and output the result represented as vertexes with value 1. Given a graph $G =< V, E >$, k is the size of vertex cover set. Create a $V \times k$ matrix, $x_{ij} = 0$ means vertex V_i is in the result set. The constrains are described as follows [2]:

Table (1) Vertex cover matrix $V \times K$ example, $k = 3, V = 5$, result set $=\{V_1, V_2, V_3\}$

V/K	K_1	k_2	k_3
V_1	1	1	0
V_2	0	1	1
V_3	1	0	1
V_4	1	1	1
V_5	1	1	1

- Exact one value 0 in each column.
- At most one value 0 in each row.
- Every edge is incident to at least one vertex of the vertex cover.

CNF-SAT-VC (mentioned above) can get the correct result but is not efficient. Except that, we will represent two approximate methods, APPROX-VC-1 and APPROX-VC-2. The former adopts greedy algorithm. It picks vertex of the highest degree all the time. The latter adopts random algorithm, it picks vertex randomly whose at least one incident edge is not covered by the previous picked vertexes. The result of them may not be correct but approximate. We used graphGen to create 10 instances for each $|V| \in \{5, 10, \ldots, 50\}$ as input and collected average running time and the size of vertex cover sets on every graph to do further exploration.

We will organize our paper as follows: In section 2, we will analyze the expected running time for each algorithm, and visualize the running time collected to verify our think and analyze the plots' trend. In section 3, we will give the definition of approximate ratio and visualize the approximate ratio collected to do comparison between three approaches. In section 4, we will combine running time and approximate ratio to compare each approach's advantages and disadvantages, then make a brief conclusion about this project.

2 Running time analysis

2.1 Time complexity expectation

- CNF-SAT-VC

  In our coding, we take SAT tool [3] as a black box, format constrain clauses as input. The only variable under our control is k, which is the size of vertex cover. We don't know the value of k, so we set $k \in \{1, 2, \ldots |V|\}$ for SAT tool until find minimal vertex cover. The worst case is that vertex cover contains all vertex so as to correct $k = |V|$. In other words, we should invoke SAT tool V times in the worst case. The running time is $O(|V| \times T(SAT))$. According to our knowledge, T(SAT) is exponential time in the worst case. So T(CNF-SAT-VC) is exponential time in the worst case as well.

- APPROX-VC-1

  The principle behind the algorithm is to pick a vertex of highest degree(most incident edges). Add it to vertex cover and throw away all edges incident on that vertex. Repeat till no edges remain. [4] Our input is an adjacency list like E = [[1,2], [0,2], [0,3], [2]]. E[0] = [1,2] means vertex 0 incidents to vertex 1 and vertex 2. Pick the vertex of highest degree takes $\theta(|V|)$. Assume we pick vertex x, there exist $x \rightarrow a$, we should add x to vertex cover but throw a. a is a list with at most size of $(|V| - 1)$. When we find each a in adjacency list, we should find x to delete, the process takes $\theta(|V|)$. In total, the running time is $O(|V|^3)$ of the input size.

- APPROX-VC-2

  The algorithm is to pick an edge $< u, v >$ and add both u and v to vertex cover. Throw away all edges attached to u and v. Repeat till no edges remain.[4]

  Our input is a list of edge [1, 2, 4, 5, 7, 0, 2, 5, 5, 1], which means edge pairs $\{(1, 2), (4, 5), (7, 0), (2, 5), (5, 1)\}$. The retrieve time all vertex attached to a certain edge is $O(|E|)$. The total running time is $O(|V| + |E|)$ of the input size.

2.2 Running time analysis

The figure (2) shows the running time when $|V| \in \{5, 10, \ldots, 50\}$. Obviously, the running time of CNF-SAT-VC is much longer than approximation approaches. When any one algorithm can't get the result in 5 seconds (we set), program exits and records the running time as ∞. It is easy to see, CNF-SAT-VC times out when number of vertexes is over 15, but other two algorithms still keep working till to 50 within a super short time.

In order to see the running time trend more easily, we plot figure (3). (a) shows the trend under the vertexes increasing by 2, $V \in \{5, \ldots, 13, 15\}$. When $|V|$ increases from 1 to 11, the curve looks like an exponential function, which fits our expectation. After $|V| = 11$, running time decreases and is Fluctuate, then goes up to more than 5 seconds sharply after |V| increases to 20. The reason of the fluctuation between $|V| = 11 \rightarrow 15$ is because, when there are more vertexes, there are possible more edge pairs. So as to there is higher possibility for SAT to judge quickly if an instance is false. So time decreases. Once number of vertexes exceeds 20, too many possible instances result to much longer time to search the true instances. Therefore, the total trend of running time will go up sharply when $|V|$ is big enough. When $|V| = 11$, running time reaches the peak of local area and the standard deviation reaches the biggest. Because $|V| = 11$ is a turning point, SAT keeps testing the instances, it may test many false instances then reach a

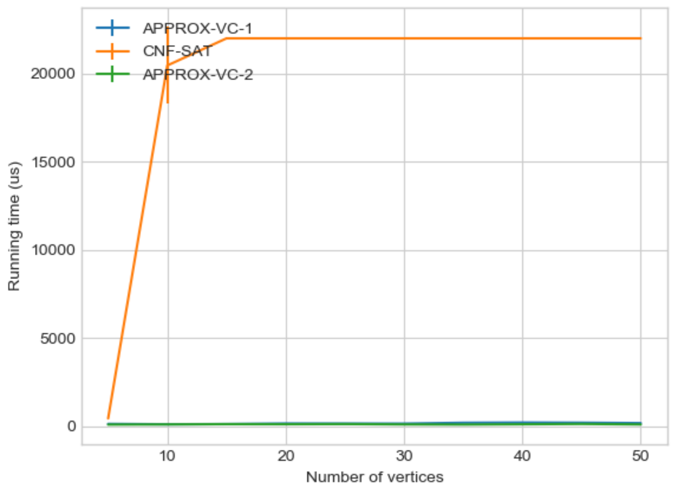
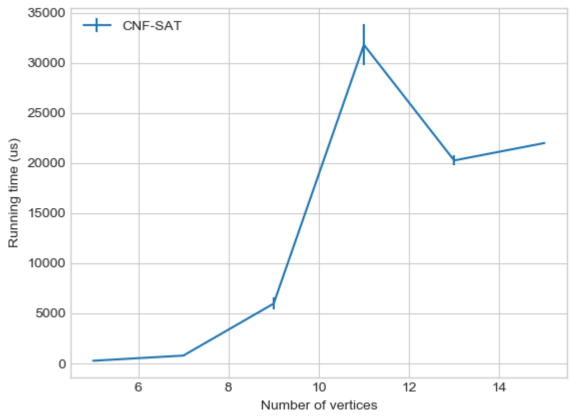
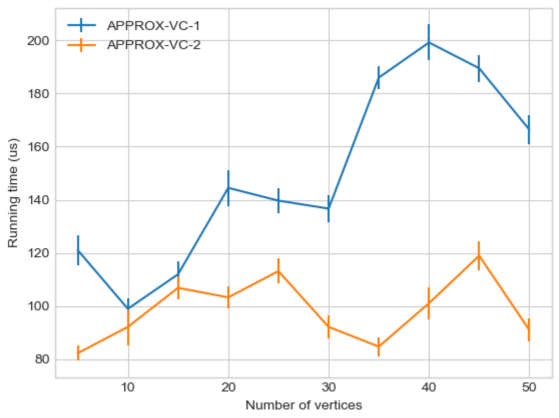


Figure (2) Group running time of CNF-SAT-VC, APPROX-VC-1, APPROX-VC-2. X aix represents $|V|$ and Y aix represents running time (μs). Set sleep 5 seconds to let program exits.



(a) Running time of CNF-SAT-VC. X aix represents $|V|$ and Y aix represents running time (μs). Set sleep 5 seconds to let program exit.



(b) Running time of APPROX-VC-1 and APPROX-VC-2. X aix represents $|V|$ and Y aix represents running time (μs). Set sleep 5 seconds to let program exits.

Figure (3) Plot running time respectively

correct instance or reach a correct instance very early. Therefore, it may find a correct instance very fast or late but the running time is still less then 5 seconds. The case is still in the tolerance of CNF-SAT-VC algorithm.

figure (3) (b) gives a clear view of two approximation algorithms. The running time is fluctuate, but the total trend of APPROX-VC-1 goes up and of APPROX-VC-2 keeps around 100 μs. For APPROX-VC-1, the worst case is the graph is full connected. By the increase of $|V|$, time increases naturally. The trend doesn't look like T=$|V|^3$ (which is our expectation), because our graphGen doesn't generate full connected graph. For APPROX-VC-2, it picked edge randomly, although $|V|$ increases, the chance of more edges increases, which means less running time. The balance between time and lucky chance is good so running time is stable. Why the trends for both algorithms don't meet our expectation completely is because the scale of $|V|$ is not big enough to see.

Another view is, running time of APPROX-VC-1 is longer than APPROX-VC-2 obviously. Because latter picks edge randomly but former picks vertex after comparing incident degrees, APPROX-VC-1 must take longer time.

3 Approximation ratio analysis

In our project, *approximation ratio* is defined as the ratio of size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover. Because there may be not only one correct vertex cover set for one graph. But the size of minimal vertex cover is fixed. CNF-SAT-VC's result must be the optimal solution, but APPROX-VC-1 and APPROX-VC-2 only refer to arbitrary vertex cover. If and only if the size of vertex cover equals to the optimal solution, the solution is optimal.

Approximation ratio represents how closed the result is to the correct answer. In our analysis, the approximate ratio of optimal solution (CNF-SAT-VC) is 1, and other two algorithms' approximate ratio ≥ 1. The bigger the approximate ratio is, the worse the solution is.

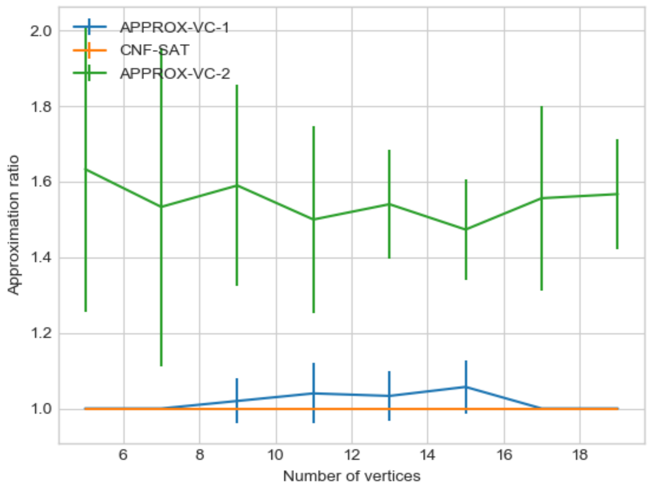


Figure (4) Group approximation ratio of CNF-SAT-VC, APPROX-VC-1, APPROX-VC-2. X aix represents $|V|$ and Y aix represents approximation ratio. Set sleep 5 seconds to let program exits.

As CNF-SAT-VC always gets the correct result, the approximation ratio is always 1, the best. Approximation ration of other two algorithms always bigger or equal than 1 because they are approximate algorithms. There are two obvious trends. APPROX-VC-1's result is very closed to correct solution but APPROX-VC-2 is always far behind. The ratio for both of them are very stable, the former keeps around 1.5 and the latter keeps around 1. Which indicates APPROX-VC-1 has much higher correctness than APPROX-VC-2. It is reasonable, because APPROX-VC-1 pick vertex by assuming the vertex with higher incident degree can cover more edges so as to minimize vertex cover. But it is not always true. For example. given a graph $G =< V, E >$, $V = \in \{0, 1, 2, \ldots, 9\}$, edge list is $E = \{< 0, 6 >, < 9, 4 >, < 0, 9 >, < 5, 9 >, < 1, 4 >, < 5, 6 >, < 9, 2 >, < 5, 3 >, < 1, 3 >, < 0, 8 >, < 7, 1 >, < 8, 2 >, < 6, 3 >, < 5, 4 >, < 1, 2 >\}$, true instance is $\{1, 5, 6, 8, 9\}$, the result of APPROX-VC-1 is $\{0, 1, 2, 3, 4, 5\}$. The other apparent trend is APPROX-VC-2 has very big standard deviation, because it picks edge randomly, the chance to be correct may be half to half. It also meets our expectation.

4 Comprehensive analysis and conclusion

We grouped three algorithms' results when $|V| \in \{5, 10, 15\}$ in Figure (5). In order to see running time and approximation ratio in one figure, when CNF-SAT-VC times out, the histogram of running time will reach top. In this figure, we can compare what the algorithms' performance in the case of different number of vertexes.

When $|V| = 5$, APPROX-VC-1 can achieve almost 100% correctness as CNF-SAT-VC but APPROX-VC-2 can't. CNF-SAT-VC's running time is generally longer than other algorithms. The running time of approximate approaches is very similar.

When $|V| = 10$, APPROX-VC-1 has a small deviation compared to CNF-SAT-VC. APPROX-VC-2 still has a very high deviation. The running time of CNF-SAT-VC is still far beyond other two algorithms, which takes similar running time. Compared to $V = 5$, running time decreases obviously.

When $|V| = 15$, APPROX-VC-1's correctness is being worse apparently, but still not too much far from CNF-SAT-VC's solution. APPROX-VC-2's correctness raises as well. Running time of the approximation algorithms are similar, which is increased rationally but is still shorter than CNF-SAT-VC's.

According to the analysis above, the solution of APPROX-VC-1 is very closed to CNF-SAT-VC but running time is as short as APPROX-VC-2. Therefore, considering the trade-off strategy, APPROX-VC-1 is the best choice in the three algorithms when exact correctness is not required.

From another perspective, with number of vertexes raising, running time of two approximation approaches doesn't change too much, which means they can keep efficiency in our test. However, the deviation of results keeps increasing, which means the two approaches will be more and more unreliable in a large size of vertexes.

Simply to say, if emphasize correctness, both of APPROX-VC-1 and CNF-SAT can be used in small scale of vertexes and APPROX-VC-2 should never be used in any case. If emphasize running efficiency, only APPROX-CV-1 and APPROX-CV-2 should be considered. If emphsize the balance between correctness and efficiency, APPROX-CV-1 is the optimal solution.

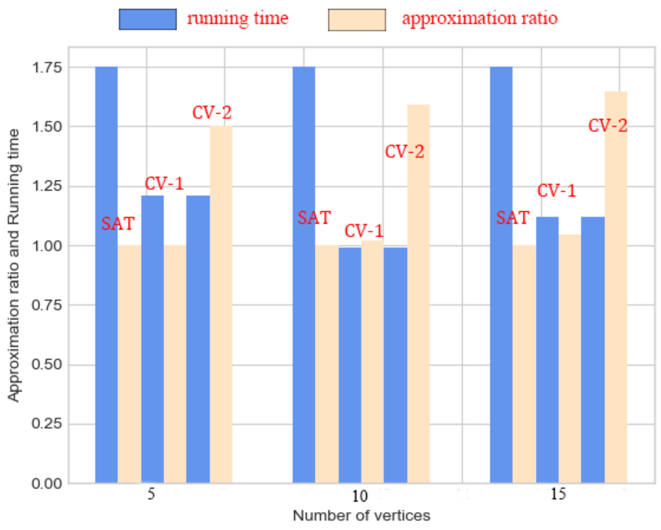


Figure (5) Group approximation ratio of CNF-SAT-VC, APPROX-VC-1, APPROX-VC-2. X aix represents $|V| \in \{5, 10, 15\}$. Set sleep 5 seconds to let program exits.

In conclusion, our project did a comprehensive analysis about the three approaches to minimal vertex cover problem based on visualizing the comparison of approximation ratio and running time. Three algorithms' performance are apparent. However, CNF-SAT-VC programming can be optimize to improve its efficiency so as to we have more data to compared the correctness of other two algorithms in a bigger amount of vertexes. Based on data we collected until now, APPROX-VC-1 is a good trade-off example.

References

[1] GreeksForGreeks. Vertex cover problem | set 1 (introduction and approximate algorithm). [Online]. Available: https://www.geeksforgeeks.org/vertex-cover-problem-set-1-introduction-approximate-algorithm-2/

[2] P. A. Gur.nkel, "A polynomial-time reduction from vertex-cover to cnf-sat," ECE 650, Fall 2019, Section 01, Waterloo, CA, Tech. Rep., 2019 fall.

[3] mElBalkini and agurfinkel. Sat solver. [Online]. Available: https://github.com/agurfinkel/minisat

[4] P. A. Gur.nkel, "Final course project," ECE 650, Fall 2019, Section 01, Waterloo, CA, Tech. Rep., 2019 fall.