Analysis of different algorithms to solve minimum vertex cover problem

Vasu Jayeshkumar Patel
vj4patel@uwaterloo.ca

Jaitoon Dhanani
j2dhanan@uwaterloo.ca

---

1 Abstract

Our main objective is to help the local police department with their installation of security cameras at traffic intersections in such a way so, police to be able to minimize the number of cameras they need to install, and still be as effective as possible with their monitoring. The problem of finding a minimum vertex cover is a classical optimization problem in computer science and is a typical example of an NP-complete problem that has an approximation algorithm.

---

2 Introduction

In terms of graph theory, the vertex cover of a graph is a set of vertices such that each edge of the graph incident to at least one vertex of the graph. A minimum vertex cover problem consider as vertex cover of smallest possible size. In this report, we will solve vertex cover problem using three different approaches by using c++ program which consist of 4 threads one for I/O and one each for the three different algorithms. The result shows the most efficient way to solve minimum vertex cover problem. We used three different approaches to solve vertex cover problem, each approach is described below :

2.1 CNF-SAT

In this approach we present a polynomial time reduction from VERTEX COVER to CNF-SAT. The graph is implemented as a set of literals and clauses to be in CNF form. In this method encoding is used to create the clauses. Different literals that are conjunction of dis- junction forms the CNF. These clauses are then send to the MiniSat SAT solver to find solution.

2.2 APPROX-VC-1

In this approach system pick a vertex of highest degree which means vertex that most frequently occurs in give sets of edges, from a graph.This vertex is added

to the vertex cover list and we remove the edges that are attached to this vertex. This process is repeated until no edges remain.

2.3 APPROX-VC-2

In this approach system pick an edge ⟨u,v⟩ from the set of edges and add both u and v to your VC (vertex cover). Then it removes away all edges that are attached to u and v. This process repeat till no edges remaining in the list.

---

3 Methodology

Each approach has been attached to single thread in c++ program. The fourth (I/0) thread takes set of vertices and edges as an input and converts it to adjacency matrix which is further used by all three approaches. In order to find the efficiency of those three algorithms we used two factors: (1) Running time, and (2) Approximation Ratio.

3.1 Running Time

To measure the running time of an algorithm we used pthread utility "pthread getcpuclockid()" which gives us the CPU execution time for that particular thread. Depending upon the number of vertices that is given as input we can calculate the running time for each algorithm and find the maximum CPU time taken by an algorithm. The results are then plotted by computing the average and standard deviation of the running time of the algorithms for each value of $\|V\|$.

3.2 Approximation Ratio

The Approximation Ratio is defined as the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover. In this method we consider CNF-SAT-VC as an optimal algorithm. The approximation ratio for our experiment defined as :

$$approximationratio = \frac{Sizeofvertexcover(eachapproximationalgorithm)}{Sizeofvertexcover(CNF - SAT - VC)}$$

The average and standard deviation of the approximation ratio of various algorithms obtained across various executions are then plotted on a graph and analysed.

---

4 Result and Analysis

In this section, We calculate the running time and approximation ratio for various values of V (number of vertices), for the graphs generated by graphGen. The graph ranges from V=5 to V=50 with an interval of 5. Note that we computed running time from V=5 to V=15 due to timeout issue in CNF-SAT algorithm we decrease our analysis range.

4.1 Running Time

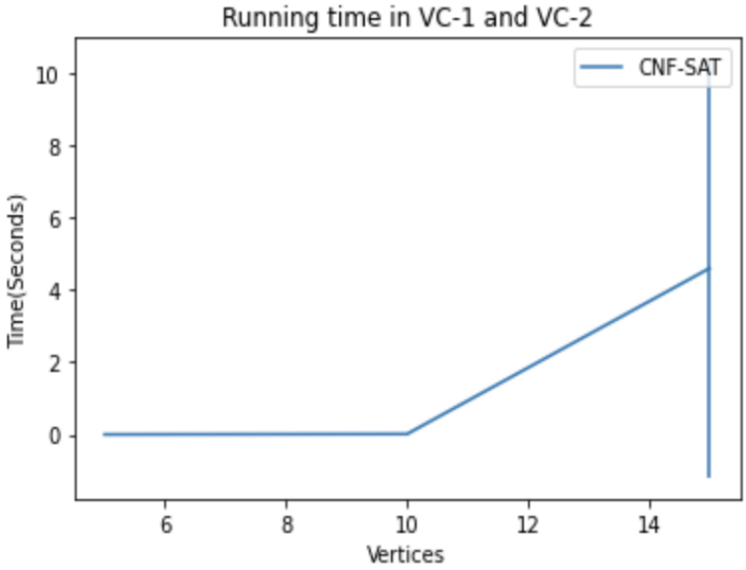

Fig 1. Relationship between running time and number of vertices in CNF-SAT-VC approach

In figure 1 we have plotted the running time of CNF-SAT Algorithm. As we can see that the running time increases exponentially along with the increased number of the vertices.Initially, for smaller values of vertices, it takes less time to find stability because less literals and less clauses and for larger values of vertices, larger the number of literals and clauses contributes to the increase in time consumption.

Similarly, the results obtained for the other two algorithms for the values of V are plotted. The following figure show the results:

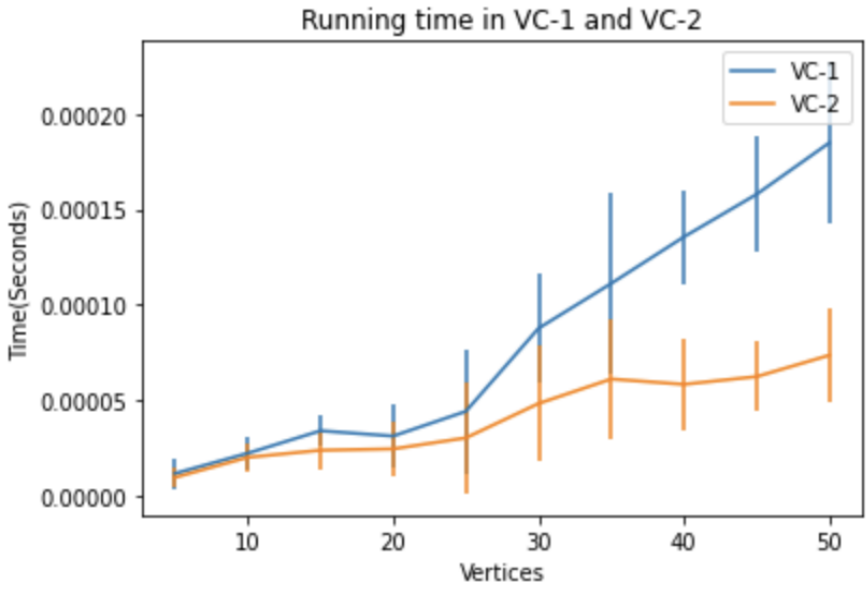

Fig 2. Comparison of APPROX-VC1 and APPROX-VC2 approach with respect to running time

In Figure 2, we have plotted the running time of APPROX-VC-1 and APPROXVC-2 with the standard deviation of the average running times for each value of V in [5,50]. Below are some remarkable observation from that :

- Both approximation algorithms APPROX-VC-1 and APPROX-VC-2 performs much faster than the optimal CNF-SAT algorithm in terms of running time. This is due to CNF-SAT algorithm converts the given problem into propositional literals and clauses and then looks for all possible combinations to find a solution. Meanwhile both the approximation algorithms solve the problem in polynomial time.
- From the above graph it also noticed that VC-2 is take less running time than the running time for VC-1 algorithm. Because VC-1 firstly finds the vertex with the highest degree and then selectively removing particular edges, whereas VC-2 randomly selects the edges and removes them, which results lesser computation.
- Both the algorithms shows linear increase in the running time with an increase in the number of vertices. Whereas in CNF-SAT-VC algorithm it increase in polynomial time. As a result these algorithms(VC1 and VC2) return results even for V higher than 15.

4.2 Approximation Ratio

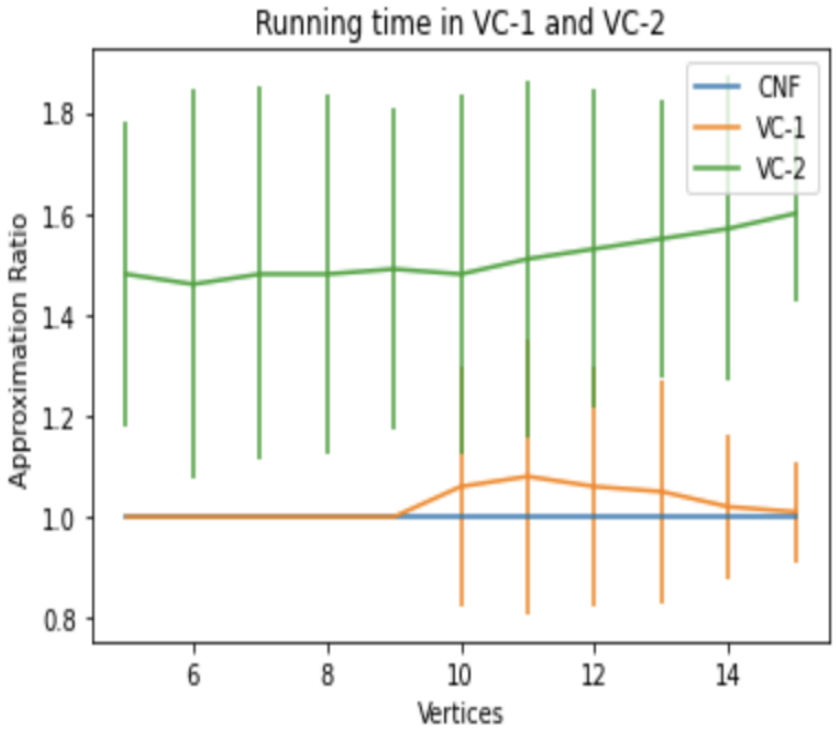

Fig 3. Comparison of CNF-SAT, APPROX-VC1 and APPROX-VC2 approach with respect to running time

The Figure 3 shows the approximation ratios of three algorithm: APPROX-VC-1, APPROX-VC-2 and CNF-SAT-VC. Below are some observation :

- As we consider CNF-SAT is optimal algorithm, it has the approximation ratio of 1 at all time, as represented by the blue line.
- The results of algorithm VC-1 are very nearly similar to the optimal solution of CNF- SAT irrespective of number of vertices. So, VC-1 algorithm provides us a near-optimal solution in polynomial time for the optimal solution provided by CNF-SAT in exponential time.
- One of the surprising change is to note that although VC-2 algorithm is much faster than VC-1 in running time, but it has a higher approximation ratio. Which shows that VC-2 algorithm is not able to provide an optimal solution in many cases.

5 Conclusion

From the above analysis and the comparison, we find out that the CNF-SAT-VC generates the best solution among all the algorithms. But when graph size increases the running time also increases. In our case, when the number of vertices is greater than 15, the computational time encounters a steep increase. Therefore if the running time is the main constraint then CNF is not a best choice.

In conclusion, when we consider Approximation ratio APPROX-VC-1 algorithm becomes more efficient than the APPROX-VC-2. When running time is the constriction of design APPROX-VC-2 should be the preferred algorithm to compute vertex covers because it takes the least time amongst all the three. But if the main focus is to find minimum vertex then, APPROXVC-2 is more likely to fail. In most cases where these two goal is needed to be satisfied, APPROX-VC-1 seems to be the optimal solution to choose.