Performance Analysis of SAT Solver and Approximation Algorithms for Vertex Cover Problem

Rayan Nagi Makki Hassan
20924251
rayannag@uwaterloo.ca

Anil Kumar Boga
20900307
aboga@uwaterloo.ca

ECE 650: Methods and Tools for Software Engineering
Department of Electrical and Computer Engineering

1. Introduction

For a given graph (V, E) - Where, V is the total number of nodes and E denotes the edges which are formed between the nodes - We define Vertex cover of a graph as the set of vertices in a graph such that each edge is incident to at least one vertex of the set. This problem of finding a minimum vertex cover is a classical optimization problem in computer science and is a typical example of an NP-hard optimization problem that has an approximation algorithm. Its decision version, the vertex cover problem, was one of Karp's 21 NP-complete problems and is therefore a classical NP-complete problem in computational complexity theory. In this project the minimum vertex cover problem has been solved using three algorithms implemented in C++: a 3-CNF-SAT Solver by polynomial reduction of the problem, and two approximation algorithms.

The MiniSAT Solver used in the 3-CNF-SAT algorithm always provides an efficient and correct minimum sized vertex cover. Nevertheless, the approximation algorithms may not always generate the minimum sized vertex cover. The following sections present the Implementation steps for the algorithms as well as performance evaluation for the computed results in terms of running time and approximation ratio.

2. Implementation

This section explains the implementation steps for the three algorithms and the specification of the development environment.

2.1 Related Methods

MiniSAT: is A SAT solver which can determine the possibility 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: 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. 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.

2.2 Experiment Setup

The analysis program is developed in C++ language and based on multi-threading. A total of four threads are used, namely I/O thread (for input), and three more threads (one for each of the vertex cover algorithms). The I/O thread is responsible for distributing input to the other threads. The running time is computed at the end of each thread, *pthread_getcpuclockid ()* function was used for getting the CPU time of each of the algorithm function.

The Experiments was run on eceubuntu server which have Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz and 64GB ram and a 64-bit Ubuntu Operating System. The performance and the output of the experimentmay vary with a different computer configuration.

2.3 Algorithms

In this experiment we have implemented three algorithms to compare their performance:

1. CNF-SAT Solver: Relies on the conversion of polynomial-time reduction form vertex cover to CNF-SAT (Conjunctive normal form) problem.
2. Approximation Algorithm 1: Picks a vertex of highest degree (most incident edges). Adds it to the vertex cover and throws away all edge's incident on that vertex. Repeat till no edges remain. We will call this algorithm APPROX-1.
3. Approximation Algorithm 2: Picks a random edge $< u, v >$ and add both u and v to the vertex cover. Throws away all edges attached to u and v. Repeat till no edges remain. We will call this algorithm APPROX-2.

The SAT solver needs to look for all possible combination to find a solution. However, the two approximation algorithms do not need to do that. The time complexity of APPROX-1 can be represented as $O(V^2)$. As we need to check for each value of V to check for highest degree it takes V^2 comparisons. Thus, we utilized an adjacency matrix-using vector of vectors and keep checking the adjacency matrix for highest degree. So every time We iterate over the entire matrix(column) to find the highest degree and then loop over the particular row value of v to find its neighbors and delete the incident edges. So in worst case its V^2 comparisons. Since the adjacent matrix uses a space of V^2 the space efficiency is also $O(V^2)$.

In the case of APPROX-2, we utilize an array and keep picking an edge $< u, v >$ randomly then delete all the incident edges on the $< u, v >$ we picked. Here, we make V comparisons in the worst case and iterate over the Edges(E). So running time can be written as $O(V * E)$ which again is polynomial to size of the input. The space complexity is also the same as APPROX-1.

3. Performance Evaluation

The analysis of efficiency for all the three algorithms were done on the basis of two factors (1) Running Time (which is the CPU clock time for each algorithm). (2) Approximation Ratio (which is the ratio of the size of computed vertex cover to the size of optimal/minimum vertex cover). Here, CNF-SAT output is considered as an optimal approach than the other outputs.

3.1. Running Time

In order to explain the running time, two graphs have been generated. Figure 1 demonstrates the logscale of running time for the SAT-solver, and Figure 2 shows the running time for both approximation algorithms.

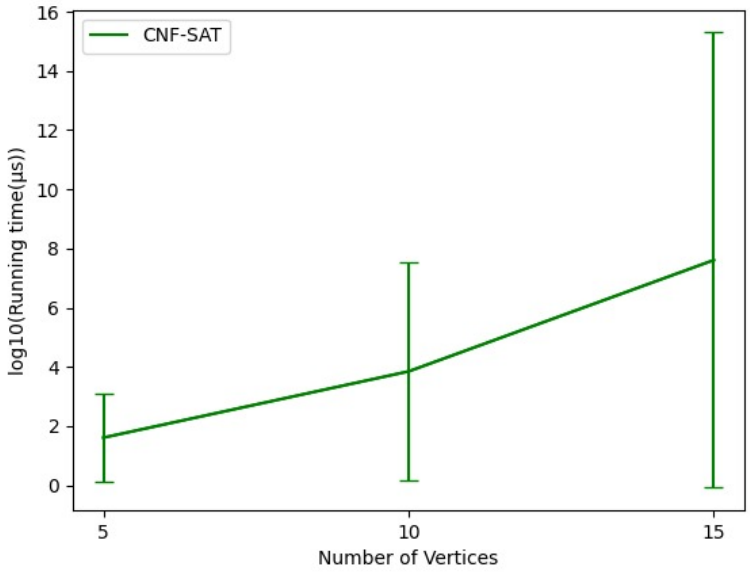

Figure 1. Running Time of SAT solver vs Number of Vertices with Log Scale

In Figure 1, we have plotted the log scale of average running time for CNF-SAT algorithm with vertices [5, 10, 15]. Each graph of size V from [5, 10, 15] was run for ten times and then for each size of V the mean and standard deviation were calculated. It is to be noted that the local CPU timed out[more than 5 minutes] in certain cases where more than vertex 15 were provided. The running time of CNF-SAT algorithm was exponential in nature and so difficult to plot in linear scale. Hence, log scale was utilized for the plot.

The CNF-SAT algorithm takes very less running time when number of vertices are less, specifically less than 10 but it shows exponential rise in the running time when number of vertices becomes greater than 10. This is because polynomial time reduction used in sat solver is related to the number of vertices, for any graph the number of clauses passed to the sat solver is given by:

$$k + n\binom{k}{2} + k\binom{n}{2} + |E|$$

Here, $\boldsymbol{n}$ is the number of vertices for a given graph. So as per the above equation the number of clauses increases exponentially with the increase in the number of vertices and

hence, we have an exponential increase in the running time of SAT algorithm after 10 vertices. We can also see the Error bars, which represent the standard deviation of the data is increased greatly for vertices above 10. This means that the running time calculation for them varied greatly for different graphs of the same vertices.

In Figure 2, we have plotted the average running time for APPROX-1 and APPROX-2 algorithms with vertices [5, 10, …, 50]. Each graph of size V from [5, 10, …, 50] was run for ten times and then for each size of V the mean and standard deviation were calculated

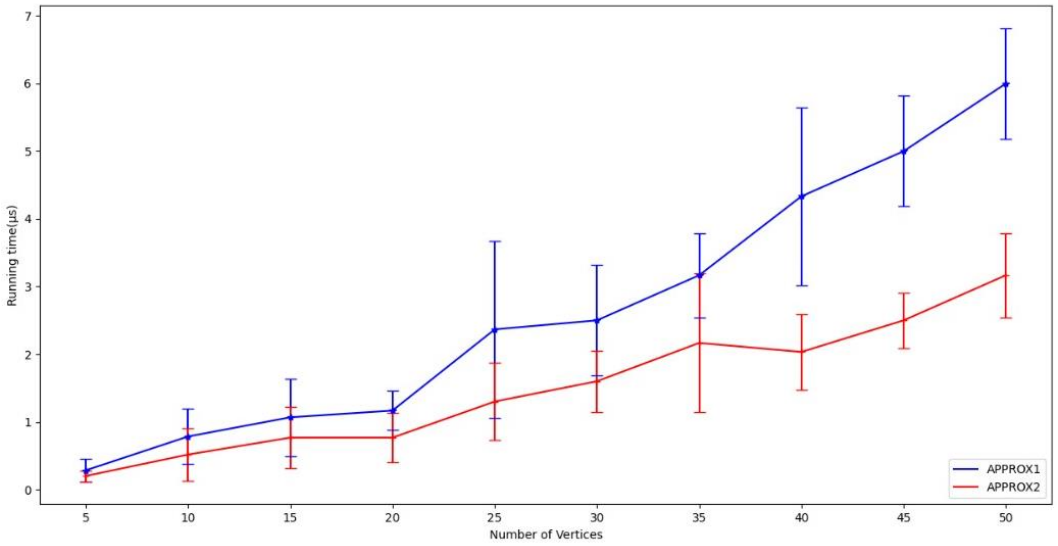

Figure 2. Running Time of Approximation Algorithms vs Number of Vertices

We find from Figure 2 that both approximation algorithms have a running time which increases with the size of the vertices. However we do not see an exponential increase as compared to the SAT solver running time. The Running time of both algorithms is still polynomial with the size of the Input(Vertices). We see that APPROX-1 takes slightly more time than APPROX-2

Upon analyzing all the graphs for running time the CNF-SAT-Solver takes by far the maximum amount of time among all approaches, while APPROX-2 have the smallest time, slightly less than APPROX-1

3.2. Approximation Ratio

In order to compare the correctness of these algorithms we take into consideration the approximation ratio. The CNF-SAT algorithm always provides an optimal solution as it checks for all possible solutions then returns the output. Thus, we take the vertex cover by CNF-SAT as the optimal solution and compare the size of this vertex cover with the vertex cover produced by APPROX-1 and APPROX-2 algorithms. Hence, we can denote:

$$\text{Approximation Ratio } = \frac{Size\ of\ Vertex\ Cover\ (Approximation\ Algorithm)}{Size\ of\ Vertex\ Cover\ (CNF-SAT)}$$

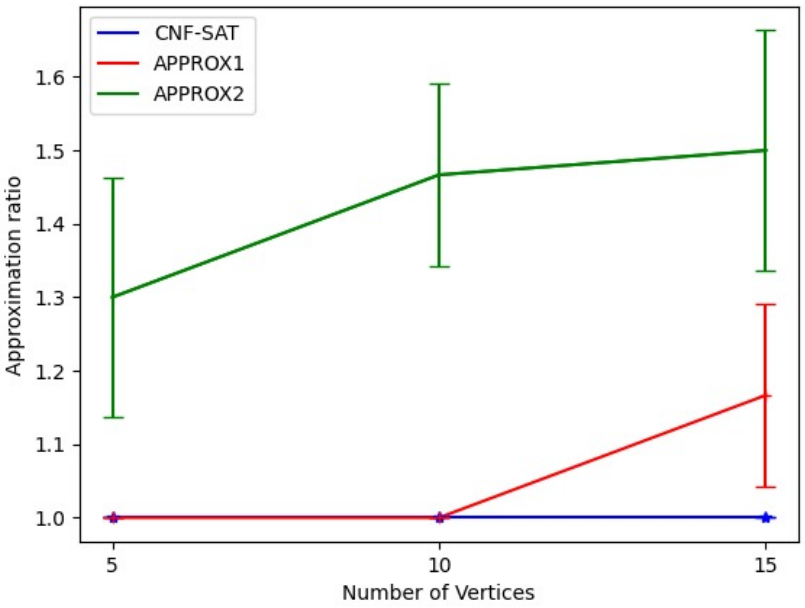

Figure 4. Approximation Ratio for All Algorithms vs Number of Vertices

We see from Figure 3 that the approximation ratio for APPROX-1 seems to have close to 1 approximation ratio which is the best we can achieve. This implies that the APPROX-1 givesalmost the optimal solution in polynomial time the same given by CNF-SAT in exponential time. The APPROX-2 is taking less time but performing worst in terms of approximation ratio. This is because its deleting edges, without considering any other issue, thus it has more chances to calculate bigger vertexcover than APPROX-1. As a result, we conclude that APPROX-1 has better approximation ratio than APPROX-2.

4. Conclusion

The running time of CNF-SAT solver is exponential with the size of the input. Though it always generates the optimal solution, it is not efficient in terms of running time for large number of vertices. When compared with the two approximation algorithms, it seen that one of the algorithms APPROX-1 seems to produce almost the same optimal solution while running at a polynomial time. In such cases APPROX-1 has an advantage over CNF-SAT. On the other hand,APPROX-2 runs in polynomial time and also faster than APPROX-2. Nevertheless, itdoes not always provide the desired optimal vertex cover solution. In conclusion, we selectAPPROX-1 to solve the vertex cover problem as it is better in terms of running time and has a near optimal approximation ratio as well.However, if optimal solution is the main desire regardless of the running cost, then CNF-SAT algorithm is to be selected.