UNIVERSITY OF WATERLOO, DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Report of Efficiency Analysis of Vertex Cover Algorithms

Zhuo Chen

December 3, 2015

1 INTRODUCTION

In this report, I will analysis three different algorithms that solving the vertex cover problems. These three algorithms are CNF-SAT, APR1, and APR2. SAT is an algorithm which using a polynomial-time reduction form vertex cover to CNF-SAT problem. APR1 just takes a simple strategy that picks a highest degree vertex in a graph and delete all related edges and continue this operation until no edges left. Moreover, APR2 will just pick edges by order and delete these edges until no edge left. The algorithms will evaluate at two ways: 1) running time and 2) approximation ratio, in which the computed vertex cover by SAT is used as optimal vertex cover (benchmark).

2 ANALYSIS

2.1 EXPERIMENT BENCHMARK DETIALS

This section analysis the efficiency of three different algorithms (SAT, APR1, APR2) that solving Vertex Cover problem. Efficiency will be evaluated by two aspects: 1) running time, and 2) Approximation Ratio, which is the ratio of the size of computed vertex cover to the size of minimum-sized vertex cover. In particular, the vertex cover computed by SAT algorithm was used as benchmark to compute approximation ratios of other two algorithms, since SAT algorithm is always guarantee an minimum-sized vertex cover for any input graph. In the following subsections, the experiment conducted analysis will be first introduced, following by discussing running time analysis of this three algorithms and approximation ratio analysis of last two algorithms (APR1, APR2).

The analysis program is based on the multi-threads program assigned in assignment 5. The main thread is responsible for I/O and repeatedly calling a wrapper function for each input graph to create three more sub-threads for executing these three algorithms separately. The running time of each algorithm is considered as the running time of the wrapper function. pthread-getcpuclockid() function is used for getting the CPU time of each wrapper function instance. The analysis program is running on ECE linux server 1. This server machine is an 8-core 4.0GHz AMD CPU with 32G of RAM, and it's OS is CentOS 5.11.

2.2 Running Time Analysis

2.2.1 SAT running time analysis

According to the experiment result, generally the SAT algorithm has the largest running time among all three algorithms for all input graphs. Particularly, SAT will meet an exponentially increment of running time at vertices number 18, and when vertices number over 18, generally SAT algorithm will cost hours to get an result. Thus the analysis of SAT running time is based on the sale of vertices number 2 to 18 rather than the requirement 5 to 50. Figure 2.1 shows the running time of SAT algorithm of graphs with vertices 2 to 18 in increments at 2.

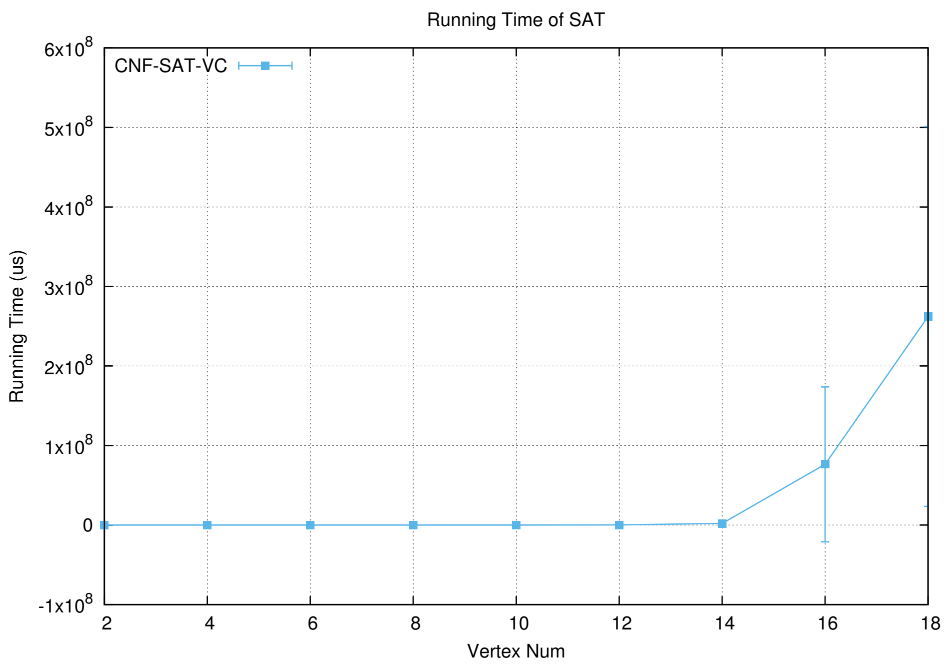

Figure 2.1: Running Time of SAT with vertices [2, 18] in increments of 2

As we can see in Figure 2.1, SAT algorithm has a very fast running time with less vertices(2 to 14), but it's running time will dramatically increase when graphs become more and more complex and connected (greater than 14). Particularly, there is an exponential growth of running time between vertex number 16 and 18. This is because the polynomial-time reduction used in my SAT algorithm is highly related on vertices number of a given graph. For any graph, the clause number of my SAT algorithm will be $k + n\binom{k}{2} + k\binom{n}{2} + |E|$, in which k is the number of vertex cover tried in each loop inside SAT algorithm, and n is the vertices number of a given graph. Thus, the clause number will exponentially increase by the increment of vertices number of a given graph. Consequently, the running time of SAT algorithm has an exponentially increment especially on vertices number between 16 and 18 according to the experiment result.

As for the standard deviation of SAT algorithm's running time, I re-plot the graph with log-scale on vertical Axes in case the huge running time on complexed graph(more than 16 vertices in a graph) compresses the observation of the standard deviation of running time on graphs with small vertices numbers. Figure 2.2 shows the re-plot result with log-scale on vertical Axes.

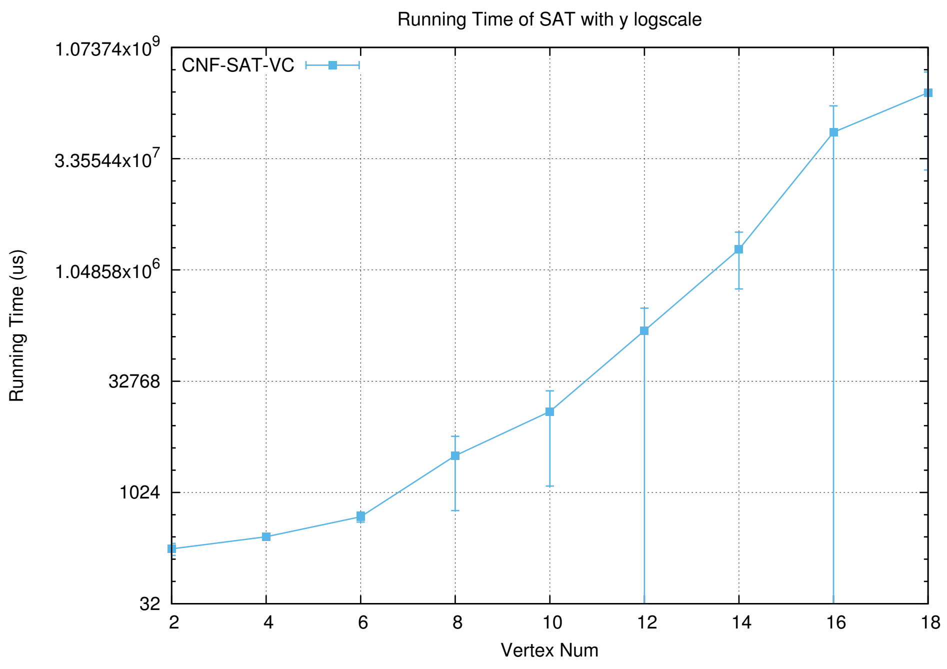

Figure 2.2: Re-plot of Running Time of SAT with log-scale on y Axes

According to Figure 2.2, we can see the standard deviation of SAT running time become bigger when vertices of a given graph increase. This is because the size of the minimum-sized vertex cover of graphs even with same numbers of edges and same vertices numbers can be very different. Moreover, this phenomenon is amplified when the vertices number of graph. As the result, the standard deviation of SAT running time become higher and higher by the size of graph growth larger and larger.

2.2.2 RUNNING TIME ANALYSIS OF ALL THREE ALGORITHMS

When all three algorithms compared together, generally the running time of SAT algorithm is highest and will over-whelm other two algorithms running time at large size of graphs. Figure2.3 shows the running time of all three algorithms at one figure with log-scale on vertical Axes.

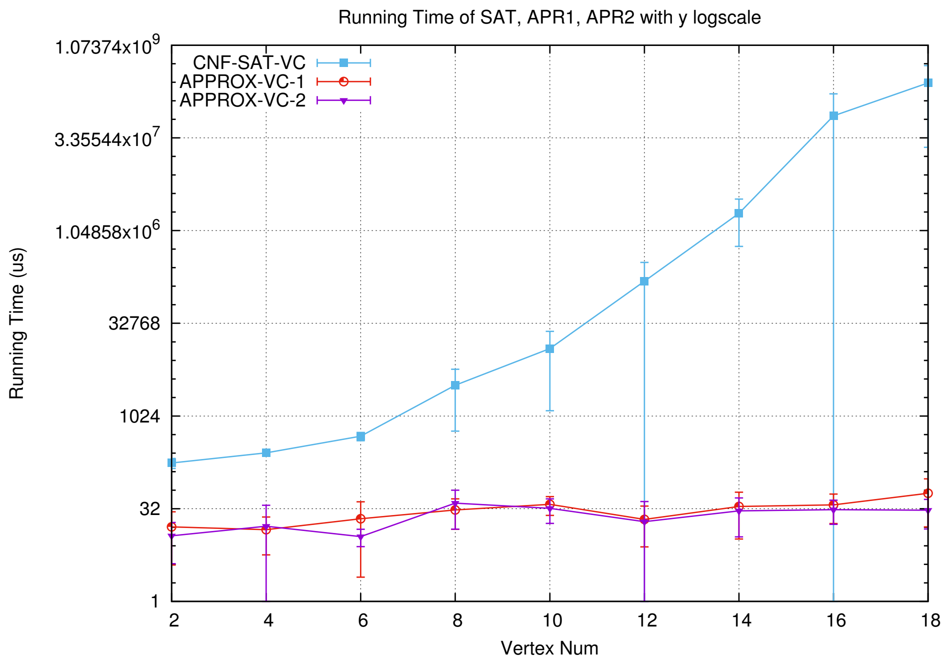

Figure 2.3: Running Time of SAT, APR1, APR2 with log-scale on y Axes

As shown in Figure 2.3, generally the SAT running time is highest and has an exponential increment of running time. In contrast, neither other two algorithms has exponential increment of running time. This is a root difference between SAT algorithm and algorithm APR1, APR2. As for SAT algorithm, it has to traverse the whole solution space and tried all possible assignments of the given CNF in order to satisfied every clause to find the minimum-sized vertex cover. For the other two algorithms, they are both $O(V^2)$ time complexity and thus never will have an dramatically exponential increment of running time as SAT did. However, in Figure 2.3 the detailed difference of APR1 and APR2 are hidden on this figure since the SAT running time is over-whelm running times of these two algorithms.

2.2.3 RUNNING TIME COMPARATION BETWEEN APR1 AND APR2

In order to observer the difference between the running time of APR1 and APR2, I running these two algorithms separately on the scale of vertex number 1 to 50 at increment of 1. Figure 2.4 shows the running time of APR1 and APR2 on this scale.

As we can see in Figure 2.4, algorithm APR1 generally has higher running time than APR2,

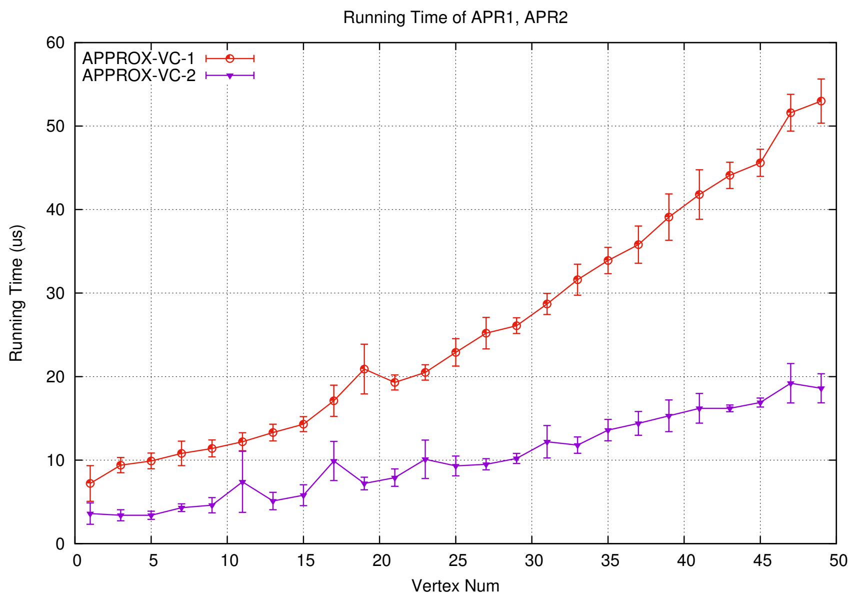

Figure 2.4: Running Time of APR1, APR2

and the growth speed of APR1 is higher than APR2. This is because in each loop APR1 has to find the vertex of highest degree in a graph, and then delete edges relates to this vertex. This operation need to traverse a given graph, which is a (V^2) time-consuming operation. In contrast, my implementation of APR2 just pick an edge by oder in each loop, which is a constant time-consuming operation. In other operations involved in these two algorithms, which mostly are edge deletions, these operations share the same implementation and thus basically consuming same time. Thus, the running time difference between of APR1 and APR2 is mainly caused by the difference between operation of "find a highest degree vertex" in APR1 and the operation of "pick edge by order" in APR2.

2.3 Approximation Ratio Analysis

In approximation ratio analysis, I use $\frac{sizeOfComputedVertexCover}{sizeOfOptimalVertexCover}$ to compute the approximation ratio of APR1 and APR2. Thus it is better to have a lower ratio among algorithms. The result is shown in Figure 2.5 as below.

According to Figure 2.5, APR1 generally could find the optimal vertex cover at small graphs(vertices number between 2 to 10) and also could keep a well performance when it comes to large graphs(vertices number more than 10). In contract, APR2 generally is performer worser than APR1 but it will become better when the size of graph become bigger. It is also can be noticed that there is a spike at the beginning of APR2. This might because at first the graph has very small vertices number and thus the APR2 will pick very much vertices as the vertex cover.

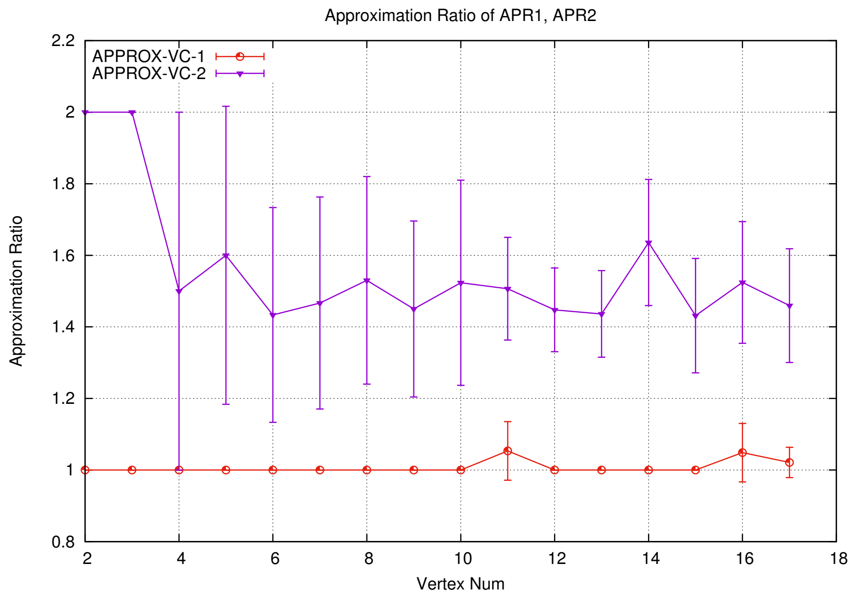

Figure 2.5: Running Time of APR1, APR2

3 CONCLUSION

In conclusion, the SAT algorithm could always find the minimum-sized (optimal) vertex cover, but it is very time-consuming. In contrast, the other two algorithms is much more efficient both in time and space but could not always guarantee an optimal solution. It seems the APR1 has the best performance in terms of running time and size of computed vertex cover because generally it cost much lower time and space compare to SAT but has lower approximation ratio compare to APR2.