ECE 650: Project

Due on Dec 9^{th}, 2018 at 11:30pm

Professor Alireza Sharifi

Lian Sun & Mengchen Shi

Department of Electrical and Computer Engineering
University of Waterloo

Contents

Introduction

This project presents a study of solving the vertex cover problem using three algorithms to achieve polynomial-time reduction of the vertex cover problem. `CNF-SAT-VC` is to translate the vertex cover into CNF clauses then solved with `MiniSat`, a binary search algorithm was also used to improve the speed of finding the minimum number of vertices`k`. The second algorithm is `APPROX-VC-1`, which picks the vertex of highest degree (most incident edges); add it to the vertex cover and throw away all edges incident on that vertex; repeat till no edges remain. The third algorithm, `APPROX-VC-2`, is to pick an edge $\langle u, v \rangle$, and add both u and v to the vertex cover. Throw away all edges attached to u and v; repeat till no edges remain. All three algorithms are running concurrently multithreaded on a machine with an Intel ® Core™ i5-4258U Processor to benchmark the algorithms.

Running time analysis

To compute the ruining time of each algorithm, the function `pthread_getcpuclockid()` was used to get the CPU time of threads, then formatted into CSV(Comma Separated Values) for plotting. The original plan was to start the program with vertices 5, then step to 20 with a 5 vertices increment in each step. However, we found that it requires hours for the `CNF-SAT-VC` to get a result with 20 vertices, and usually terminated by the shell in the middle of a run. So we use 17 vertices in the last run instead of 20. Figure 1 and 2 show a cross comparison between the three algorithms, with box and whisker plots to check the variation in samples, scatter plot to visualize the distribution of each run, and average group result with +- one standard deviation(68% *confidence interval*) to estimate an overall trend of the running time. Also, note that the y-axis ticks are scaled with log_{10} to accommodate the extreme outlets in the run with 15, 17 vertices.

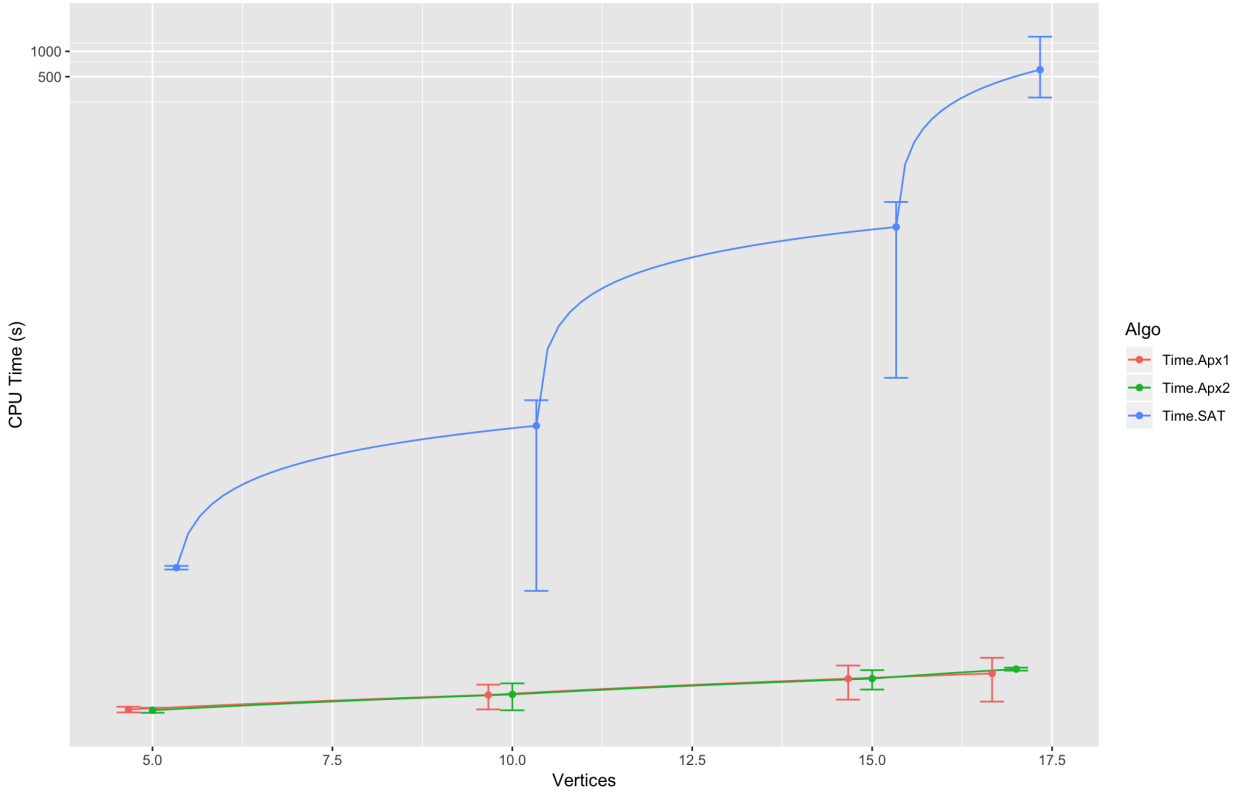

Figure 1: Average CPU time with standard deviation comparison between three Algorithms

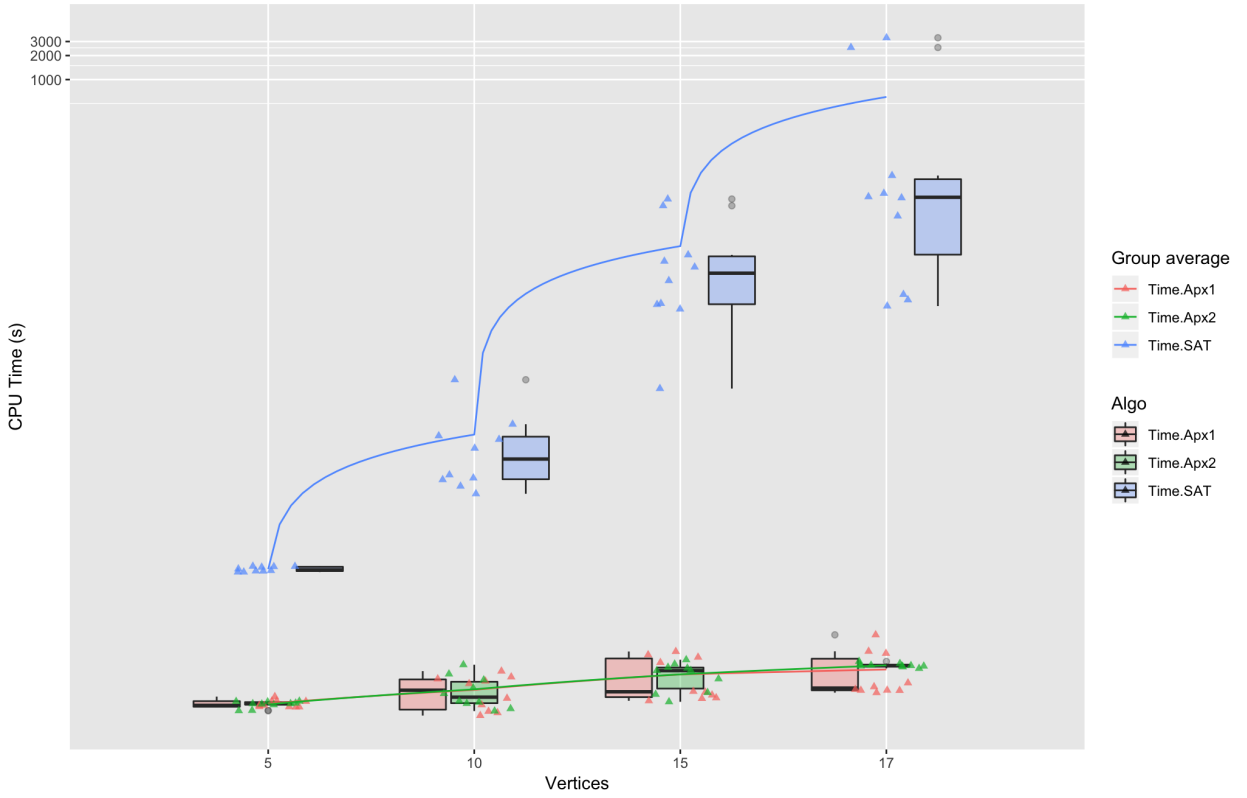

Figure 2: CPU time comparison between three Algorithms

CNF-SAT-VC

The running time of `CNF-SAT-VC` is depicted in light blue in Figure 1 and 2. The running time shows an exponentially increasing trend with the increase of $|V|$. The binary search was used to efficiently find the smallest vertex cover k, in each iteration of the search, the middle value of k within the range of available Ks was used to generate and add the clauses to the `Minisat` solver. For every k, the number of clauses can be calculated as:

$$k + n\binom{k}{2} + k\binom{n}{2} + |E| \tag{1}$$

From the above formula, we can see that as $n = |V|$ grows, the number of clauses that the Minisat needs to solve is growing at a rate much faster than the growth rate of $|V|$, and we can conclude that the running of `CNF-SAT-VC` is exponential and can be greatly impacted by the number of vertices.

It is also worth noting that the running time variation of the `CNF-SAT-VC` algorithm is also greatly increased as the number of vertices increases. As can be seen in the Figure 1 and 2. The distribution of the running time is tight when $|V| = 5, 10$. However as $|V|$ increased to 15 and 17. The distribution of the running time spreads out a lot. Some with extreme values are as high as $3000s$ in the run with 17 vertices, while some are as low as few seconds. This could be explained as when$|V|$ gets larger, the graphs differ a lot, as some graphs could have a small vertex cover, well others may have a big one. With more vertices, the number of clauses increases dramatically, plus there will be more trials in the binary search $O(log(n))$ to find the minimum vertex cover.

APPROX-VC-1

The average running time of `APPROX-VC-1` is the lowest among the three, as seen in Figure 1 and 2 depicted in red. The tendency of running time growth appears to be linear, which is expected as the running time complexity is $O(V^3)$. The performance of this algorithm is relatively stable, as there is no obvious trend of increase in running time variation with the increase of the number of vertices.

APPROX-VC-2

The average running time of `APPROX-VC-2` is also small, depicted in green in Figure 1, and 2. The performance is very close to `APPROX-VC-1`, with more stable performance, since the distribution is even tighter as compared with `APPROX-VC-1`. The running time grows faster than `APPROX-VC-2`. The difference may come from the algorithm design, the `APPROX-VC-2` involves more randomness in selecting a starting edge, as compared with `APPROX-VC-1` which starts with the vertex of the highest order, which is naturally more likely to be in a vertex cover.

Approximation ratio analysis

In the previous section, the speed of the algorithms is compared. This section evaluates the accuracy of the two approximation algorithms(`APPROX-VC-1`, `APPROX-VC-2`) using `CNF-SAT-VC` as a benchmark. The ratio distribution and group average are shown in Figure 3 and 4.

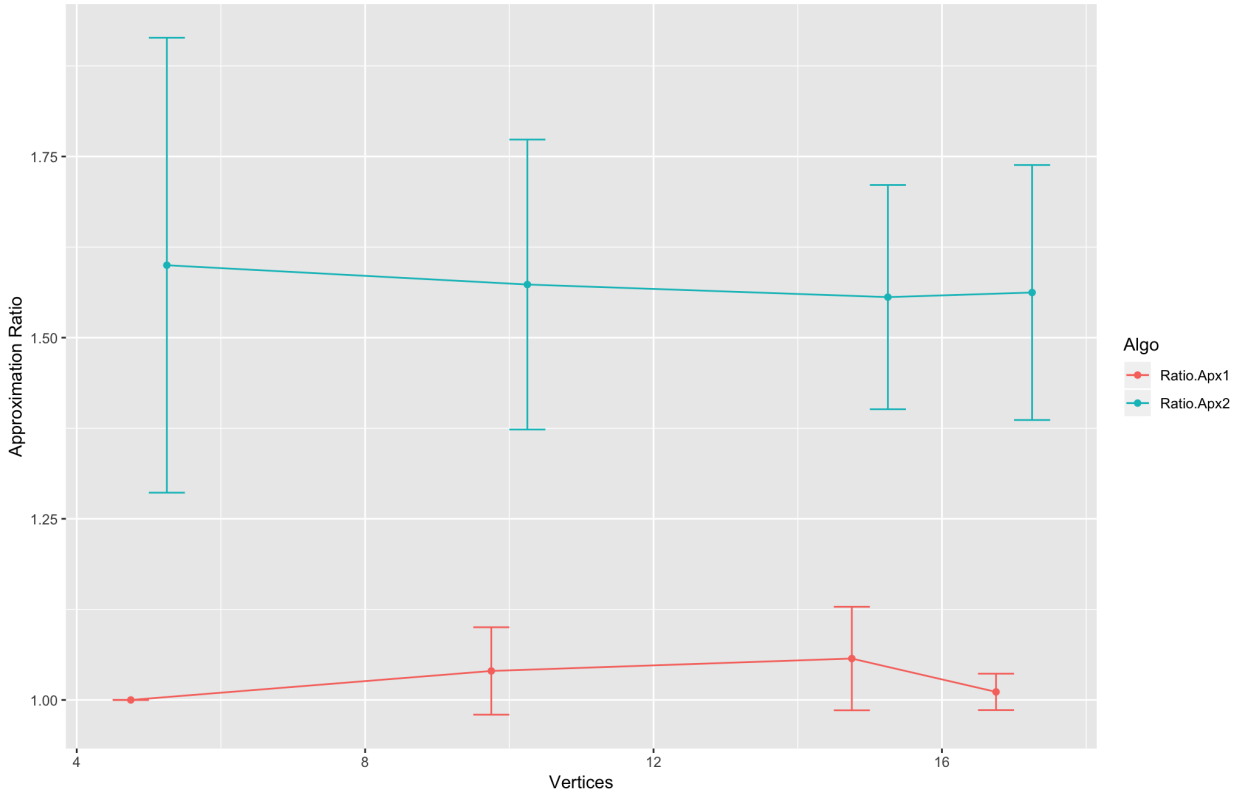

Figure 3: Average approximation ratio with standard deviation comparison of APPROX-VC-1 and APPROX-VC-2

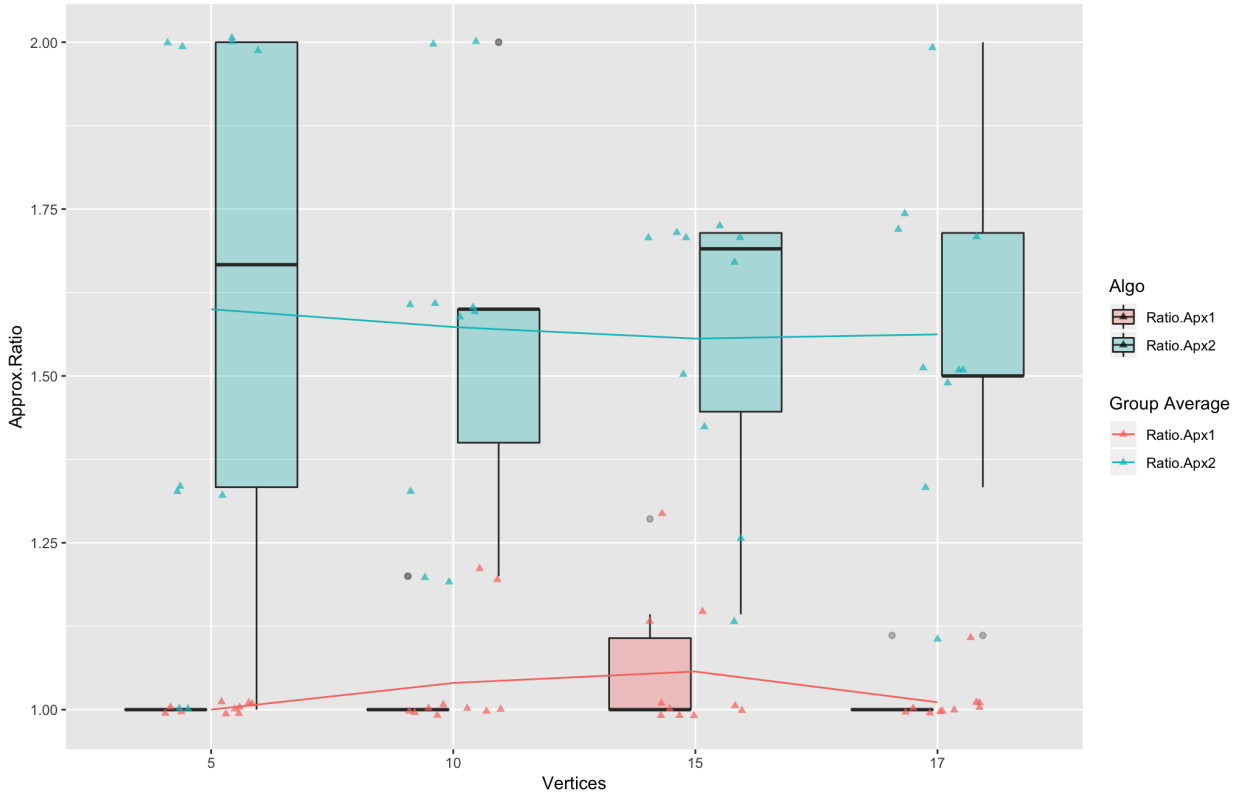

Figure 4: Approximation ratio comparison of APPROX-VC-1 and APPROX-VC-2

APPROX-VC-1

The approximation ratio of `APPROX-VC-1` is depicted in red of Figure 3, and 4. We can see that the approximation ratio is very close to 1, the slight increase with 10 vertices might be explained by outliers. The close to 1 approximation ration means that this algorithm can output a minimum-sized vertex cover for most of the times. The accuracy of its approximation is very close to `CNF-SAT-VC`.

APPROX-VC-2

The approximation ratio of `APPROX-VC-2` is depicted in teal of Figure 3, and 4. The result shows the approximation is slightly greater than 1.5. For this algorithm, we can prove that the best case approximation ratio is 1, and the worst case approximation is 2. The slight higher approximation ratio could be explained as `APPROX-VC-2` computes vertex cover in pairs of vertices of any selected edges. From the box plot, we can see that the variation of the distribution is much greater than that of `APPROX-VC-1`, which is expected as `APPROX-VC-2` choosing the edges randomly.

Conclusion

Considering the tradeoff between running time and accuracy. The `APPROX-VC-1` outperforms `APPROX-VC-2` in terms of running time and approximation ratio with approximation ratio very close to `CNF-SAT-VC`. In contrast, `CNF-SAT-VC` guarantees to find the minimum-sized vertex cover, but it requires a much longer time(exponential) to get the result, sometimes more than 1000 times longer than `APPROX-VC-1` and `APPROX-VC-2`, and we could only expect it to take much longer time to find the vertex cover with vertices larger than 17. Thus for applications when minimum errors are acceptable, we should go with `APPROX-VC-1`. For situations when accuracy is on the first priority or when the number of vertices is relatively small(i.e. $|V| < 17$), we should consider using `CNF-SAT-VC`, as it guarantees the minimum-sized vertex cover.