Title: Project Report for Methods and Tools for Software Engineering

Course ID: ECE 650

Student ID: c5mao, c279li

Student Group: Chunyu Mao, Chunxiao Li

Introduction

This course project is to evaluate the performance of three different algorithms that designed for minimum vertex cover problem. The evaluation is demonstrated with C++ implementation based on assignment 4. By running various graphs which are generated by /home/agurfink/ece650/graphGen/graphGen on ecelinux, the efficiency is characterized in running time and approximation ratio with respect to different number of vertices range from 5 to 20. This report is structured in two parts. Brief introduction to three minimum vertex cover algorithms will be given first. The second part is the performance analysis demonstrated with figures.

Minimum Vertex Cover Algorithms

CNF-SAT-VC: This is an optimal (minimized sized) vertex cover algorithm that reduce the vertex cover problem to CNF-SAT problem in polynomial time. It takes the input graph from the vertex cover problem and produces a formula F with the property that the graph has a vertex cover of size k if and only if F is satisfiable. Then an SAT solver is employed to verify the formula satisfiability.

APPROX-VC-1: This is an approximate vertex cover algorithm. It greedily picks a vertex of highest degree and removes all the edges incident on that vertex. Repeat till no edges remain. The time complexity is characterized as O(|V|+|E|). This greedy algorithm will result in a vertex cover with size at most log(|V|) times as the optimal solution.

APPROX-VC-2: This is an approximate vertex cover algorithm. It randomly picks an edge and add both vertices of the edge to vertex cover and remove all the edges incident on those vertices. Repeat till no edges remain. The time complexity is characterized as O(|V|+|E|). This algorithm will result a vertex cover with size at most twice as the optimal solution. Since at least one vertex of an edge should be in vertex cover.

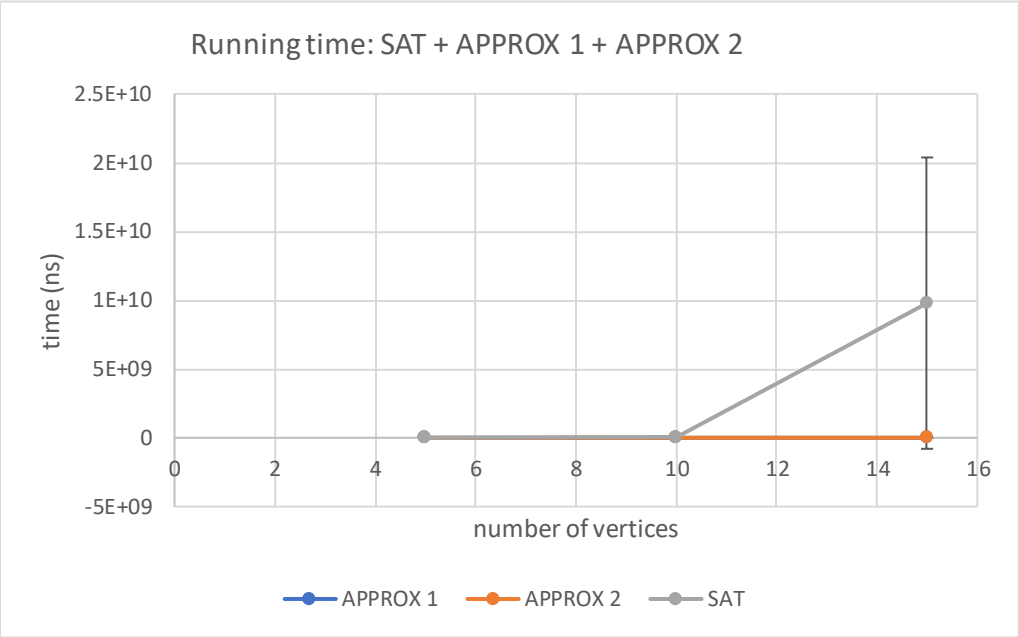
Experiment setup

We implemented the three algorithms described above in C++, we will not describe our implementation in details for this report, but we will attach our source code along with this report. We performed our experiments on eceubuntu machines, and the machine we used is Ubuntu 18.04 LTS i5-8400 on eceKVM.

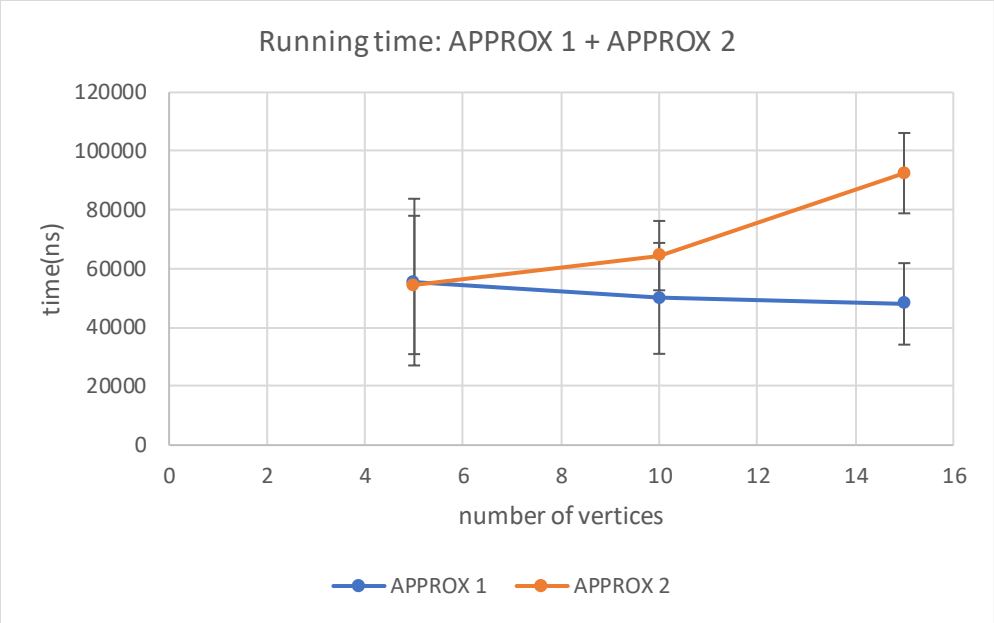
Analysis (Running time)

The performance evaluation is divided into two parts, one is running time, and the other is approximation ratio.

Figure below is the plot of running time of each of the three algorithms described above versus number of vertices from 5 to 15.



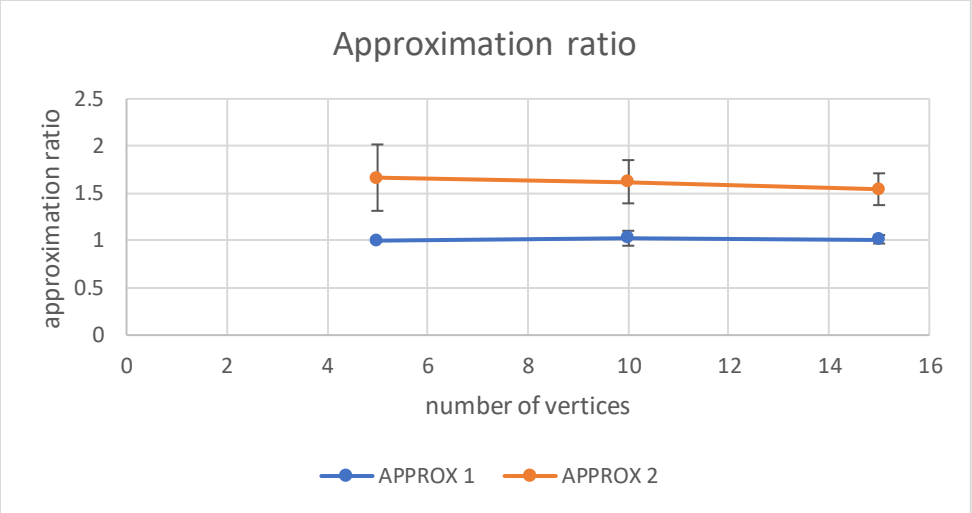
From the figure we can see that the running time for our SAT approach becomes much larger as we increase the number of vertices from 10 to 15. Due to its hardness to solve, we were not able to solve the minimum vertex cover problem for a graph with 20 vertices. However, by observing the figure above and considering the fact MiniSAT times out for 20 vertices, we would like to conjecture that, the grey line will grow exponentially as number of vertices increase. On the other hand, in the figure above, the blue line and orange line seems to be colliding, and it is to compare the two approximation algorithms just from the figure. Below is a plot without the grey line, and the axis are scaled accordingly. (We didn't discuss much on the standard error, since our y axis is in nanoseconds, and the solving time for our SAT approach takes around 20 seconds, and each run differs in scale of seconds, which means the standard deviation is going to be large in the unit of nanoseconds, and it may not be meaningful to analyze the standard error in this case per se.)



From the figure above, clearly, our first approximation algorithm performs better than the second from the set of benchmarks we tested on. However, note that the running times for each algorithm for solving graphs with up to 15 vertices is well under a second, so the efficiency difference between the two algorithms we are observing may not necessarily reflect the difference in the complexity of the algorithms themselves, for example, physical conditions of hardware may affect running time, also, the differences we observe may due to implementation as well. But after all, if we consider the first plot, it is obvious that the two approximation algorithm are "efficient" for graphs with up to 15 vertices, and it seems like they will scale well asymptotically, in fact, this observation is consistent with our theoretically analysis that both approximation algorithms are polynomial algorithms.

Analysis (Approximation ratio)

Due to our reduction from the minimum vertex cover problem to SAT, we are guaranteed to find the minimum vertex cover the input graph, we will not consider approximation ratio for outputs from our SAT algorithm, instead, we use it as the standard, and compute the approximation ratio of for each of the approximation algorithms. The approximation ratio is computed as follow: for each input graph, we count the number of vertices returned by our approximation algorithms and use that number to divide the number of vertices returned by our SAT algorithm. Consider the figure below:

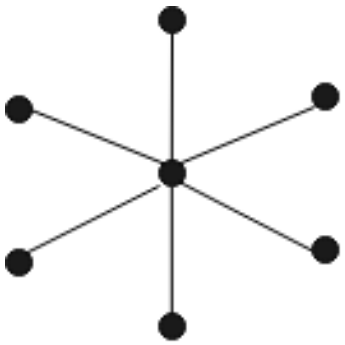


Due to the way approximation ratios are computed, clearly, our first approximation algorithm outperforms the second as the size of vertex cover returned by it is closer to the minimum vertex cover, and it is consistent across our benchmarks.

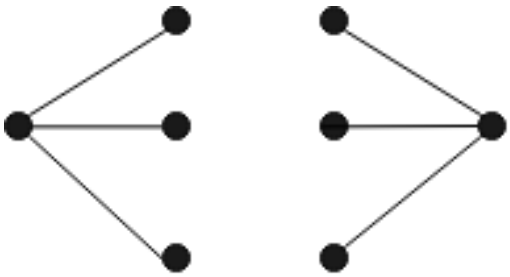
Beyond the Benchmark

Regarding the random generated graphs with number of vertices equal to 20, because of the computational complexity of SAT solver, CNF-SAT-VC can not give a solution in fixed time (less than the limited time on eceubuntu) compared to the APPROX-VC-1 and APPROX-VC-2. Out of curiosity, we investigated into whether certain properties of input graphs may imply a faster SAT solving time. We show the existence of some graphs with 20 vertices, that our SAT based approach was able find a solution quickly. Consider the following two classes of graphs:

1.



2.



We choose not to give a formal mathematical definition for these classes of graphs, as we generated these graphs based solely on intuition, and we do not know if these classes of graphs indeed capture what we want. On a high level, we conjecture that graphs that have many low degree vertices are easy for our SAT based approach. And it seems that the set of experiments we performed confirms with our intuition, in fact, we encoded graphs with 20 vertices from

both classes, and our SAT based approach was able to find the minimum vertex cover under a second. To be more precise, we encoded the graphs with the following:

G1:
V 20
E{<0,1>,<0,2><0,3>,<0,4>,<0,5>,<0,6>,<0,7>,<0,8>,<0,9>,<19,10>,<19,11>,<19,12>,
<19,13>,<19,14>,<19,15>,<19,16>,<19,17>,<19,18>}
G2:
V 20
E {<0,1>,<0,2>,<0,3>,<0,4>,<0,5>,<0,6>,<0,7>,<0,8>,<0,9>,<0,10>,<0,11>,
<0,12>,<0,13>,<0,14>,<0,15>,<0,16>,<0,17>,<0,18>,<0,19>}

And we summarize the outputs for our algorithms for G1 and G2 in the table below:

G1	run time (ns)	vertex cover size
SAT	661391	1
APPROX 1	97210	1
APPROX 2	125970	2

G2	run time (ns)	vertex cover size
SAT	1444011	2
APPROX 1	83080	2
APPROX 2	70870	4

Even though our SAT approach takes longer time in both cases, but we still consider it as efficient with respect to these classes of inputs since it still runs under a second. And on the other hand, the approximation ratio for APPROX 1 and APPROX 2 seems to be consistent with our experiments for scaling studies.

Conclusion

In summary, CNF-SAT-VC is guaranteed to give the optimal solution, however, due to the NP-complete essence, the computation time is extremely slow as the number of vertices increase. In contrast, the two approximation algorithms are not always devising optimal solution. However, the time complexity of those two algorithms is roughly in the same level which is far faster than CNF-SAT-VC. From the simulation of project implementation, APPROX-VC-1 is faster than APPROX-VC-2 within the test scenario. And, APPROX-VC-1 is also more accurate than APPROX-VC-2.