ECE650 - Methods and Tools for Software Engineering

University of Waterloo

Department of Electrical and Computer Engineering

Project: Report

Authors:
Runfeng Qian (ID: 20835372)
Xiaoliang Zhang (ID: 21051256)

Date: December 8, 2023

1 Introduction

In the field of software engineering, addressing complex problems with efficient algorithms is a core challenge. One of the such main complex problems is the minimum vertex cover (MVC) problem, which is a NP-complete problem. This kind of problems is to generate a valid vertex cover with the minimum size of a given graph. In this report, we will implement three different approaches trying to solve the MVC, and give the analysis based on their results. We notice that, there is a balance between the quality and the running time of the solutions among these approaches.

The rest of the report consist of three sections. Section 2 will describe how we implement the three approaches and how we carried our experiments. Our results and analysis will be given in section 3. Finally, we will give a brief conclusion.

2 Methodology

2.1 Minimum vertex cover algorithms

The three algorithms implemented to solve MVC are: reduction to CNF-SAT (we call it CNF-SAT-VC), APPROX-VC-1 and APPROX-VC-2, whereas CNF-SAT-VC give a precise solution to the MVC problem, and the latter two just provide an approximation for them.

We would like to briefly introduce these three approaches, more details can be found in documents given in ece650, especially in assignment4.pdf, a4-encoding.pdf and project.pdf.

CNF-SAT-VC: in this algorithm we logically create a formula in CNF form to represent the problem of MVC. Then by using minisat, a famous high-effiency SAT solver, we solve the derived CNF problem.

APPROX-VC-1: as said in project.pdf, in this algorithm we pick a vertex of highest degree. Add it to the vertex cover and throw away all edges incident on that vertex. Repeat till no edges remain.

APPROX-VC-2: as said in project.pdf, in this algorithm we 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.

2.2 Multithreading

In this project, we make our program run in a multi-thread way. There are four threads in total in our program. One is the main thread as the I/O for processing input data and output results, the other three are worker threads, each one of them in charge of one approach mentioned above.

The main thread runs a infinite loop, reading input lines until the end of file (EOF) is reached. It parses the input and creates 3 worker threads mentioned before. Then the main thread monitors the running process of the 3 worker threads, the worker thread for CNF-SAT-VC will be terminated if it takes very long time before getting results, besides, it will wait for the completion of the three worker threads and output the results they have got.

Multi-thread allows for parallel computation by different MVC algorithms simultaneously, effectively reducing the overall computation time and making full use of accessible resources.

2.3 Running time measurement

In order to measure the running time in high accuracy, we employed a high-resolution timing tool called CLOCK_MONOTONIC. The time will be measured in nanoseconds and converted to microseconds, so that it would be in a reasonable range to be plotted in figures.

As mentioned above, we implement a timeout mechanism for the CNF-SAT-VC. It is required because it costs a very long time running this exact approach when the graph size is extremely large, in the worst case exponential to the size of the graph due to the NP hard nature of the problem. So in order to prevent infinite waiting, we will simply terminate the thread.

2.4 Experiment procedure

For our experiment, all the code is run on ECEUBUNTU with the Clang compiler.

We generate the graphs by graphGen for number of vertices in (5,10,15,...,50), and 10 graphs are generated for each value of number of vertices. Moreover, for each of the above generated graphs we run the code 10 times to record their output MVC and their running time for further processing and plotting. In all, for each value of number of vertices, we will have the data of 100 runs.

2.5 Data processing

We define the approximation ratio as the value of the size of the estimated minimum vertex cover by the 2 approximation methods over the size of the real minimum vertex cover, with the output of the CNF-SAT-VC being the guaranteed real solution.

We group the data based on their number of vertices. And we calculate the approximation ratio for every group. Similarly, we will calculate the mean and standard deviation for the running time for all groups. Finally we use the calculated mean and standard deviation to plot the figures.

3 Results and analysis

3.1 Results

The running time and approximation ratio are plotted in the following figures. Figure 1 and figure 2 depicts the running time of the algorithms vs. number of vertices, and figure 3 details the approximation ration vs. number of vertices. Our analysis with all this information is given in section 3.2. We also notice that there exist a trade-off between the running time and the approximation ratio, which is the quality of the solution.

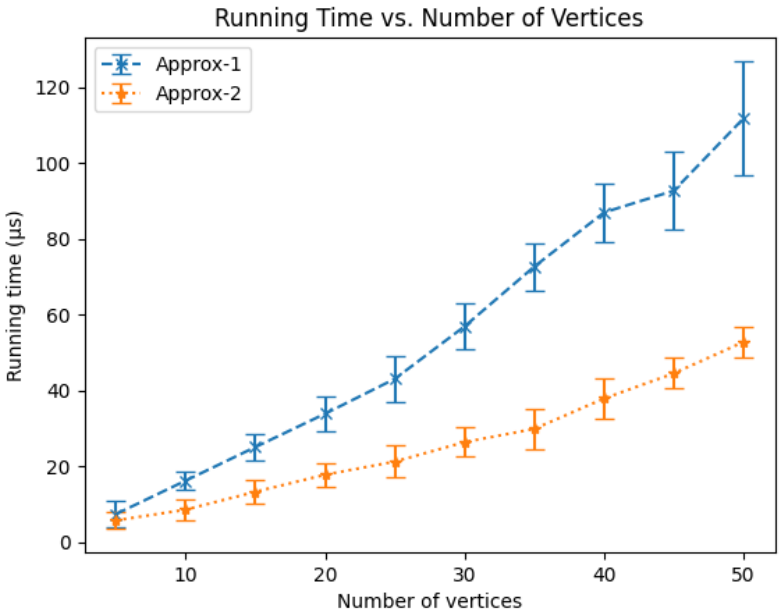

Figure 1: Running time for APPROX-VC-1 and APPROX-VC-2 vs. number of vertices

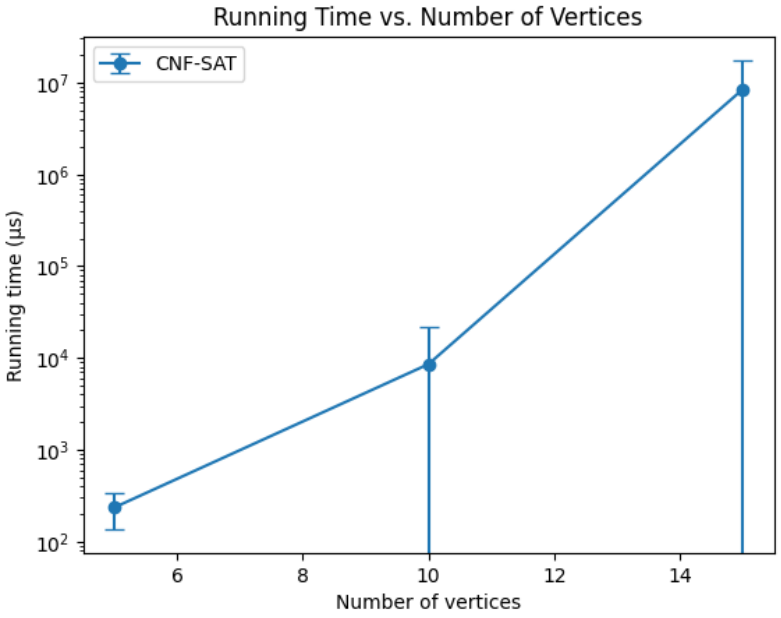

Figure 2: Running time for CNF-SAT-VC vs. number of vertices

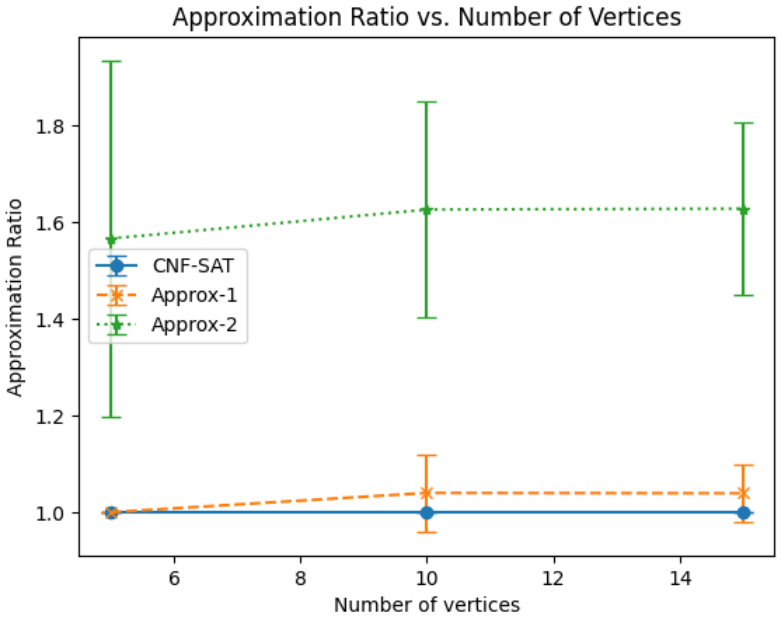

Figure 3: Approximation ratio for 3 approaches vs. number of vertices

3.2 Analysis

3.2.1 Running Time

Figure 1 illustrates the running times of the APPROX-VC-1(Approx-1) and APPROX-VC-2 (Approx-2) algorithms. We can see that both algorithms shows a trend of increasing running time. For Approx-1, its running time increases from 7.4 to 111.8 μs, whereas the running time for Approx-2 increases from 5.7 to 52.8 μs. So Approx-1 has both a larger running time and a steeper increasing trend in running time than Approx-2, suggesting that Approx-1 may be slightly less efficient as the graph size grows. Besides, Approx-2 also has lower standard deviation, ranging from about 2.1 to 5.5, compared with that ranging from 2.5 to 15.1 for Approx-1, for its running time, which means that Approx-2 is more stable and predictable with the increase of the running time. And for both methods, the standard deviation is increasing with the number of vertices, we think that is because the graph is becoming complicated and the search space grows.

In contrast, Figure 2 shows the running time for CNF-SAT. We use logarithmic coordinates because the running time can be extremely large when the number of vertices increases, which is because the NP-hard nature of the problem. With the number of vertices increase from 5 to 10, the running time increases from about 235.5 μs to about 8564.2 μs, and when it increase from 10 to 15, the running time reaches $8.4 \times 10^6 \mu s$. Then, when the number of vertices is larger than 16, the timeout mechanism is triggered, indicating the impracticality of this method for larger graphs. The standard deviation of its running time is also very large, ranging from 99.1 to 9.08×10^6, and grows sharply with the increase of number of vertices, we believe it is due to the exponential search space and complexity of the MVC problem.

3.2.2 Approximation Ratio

Figure 3 illustrates the comparison of approximation ratios among three algorithms. The CNF-SAT approach, an exact method, is set to have a constant approximation ratio of 1. It is evident from the figure that Approx-1 outperforms Approx-2; Approx-1's approximation ratio ranges from 1 to 1.04, whereas for Approx-2, its approximation ratio ranges from 1.566 to 1.628. We can see that Approx-1's ratio is just slightly more than 1 for every vertex number, and Approx-2's ratio is approximately 0.5 greater than that of Approx-1 at each vertex number. Besides, the standard deviation for two methods also differs. For Approx-1, its standard deviation is in the range of 0 to 0.08, with the change of number of vertices. On the other hand, Approx-2's standard deviation is much larger, ranging from 0.18 to 0.37. So we can summarize that the solution quality of Approx-1 is much better than Approx-2. We think this might be due to a combination of things. Firstly, the Approx-2 algorithm incorporates an edge (two vertices) in each iteration, unlike Approx-1, which adds just one vertex at a time. Secondly, the search heuristics of the algorithms differ: Approx-1 systematically seeks the vertex with the highest number of incident edges, whereas Approx-2 selects edges randomly. Moreover, we also notice that there is no obvious trends of standard deviation with the increase of number of vertices, this might be

caused by the insufficiency of samples. Since it becomes intractable to get the ratio when the number of vertices is larger of equal than 20.

Furthermore, we notice that there is a trade-off, as said before, between quality of solution (indicated by the approximation ratio) and running time among the three algorithms. CNF-SAT, as an exact algorithm, generate the solution with the highest quality, but with lowest running time. For the two approximation algorithms, Approx-1 has a larger running time, but a higher solution quality compared with Approx-2.

4 Conclusion

In the end the data indicates that the two approximation algorithms run much faster than the CNF-SAT function. However they will produce slightly larger vertex cover than the exact algorithm. So in the end if running time is not of concern we recommend using CNF-SAT method. But if running time is of great importance, we recommend using Approx-1 instead of Approx-2. This is because the vertex cover Approx-1 generates is much closer to the exact minimum vertex cover, while it only trades off slightly in running time and scalability.

5 References

assignment4.pdf
project.pdf
a4-encoding.pdf