Efficiency Analysis for Three Vertex Cover Algorithms

Eric Yu
z333yu@uwaterloo.ca

Yichun Ma
y66ma@uwaterloo.ca

December 5, 2023

Abstract

In this report, we analyze three algorithms for calculating the Vertex Cover problem. We call them CNF-SAT, APPROX-1, and APPROX-2, among which the first one calculates one of the minimum vertex covers, and the other two calculate approximate vertex covers. We analyze the time efficiency of these three algorithms and the approximation ratio of the two approximation algorithms. Also, we explore the CNF-SAT and improve the encoding to make it more efficient.

1 Runninng Time Analysis

As we have conducted experiments with three algorithms: **CNF-SAT**, **APPROX-1**, **APPROX-2**, we will show the time experimental results in Figure 1 and Figure 2.

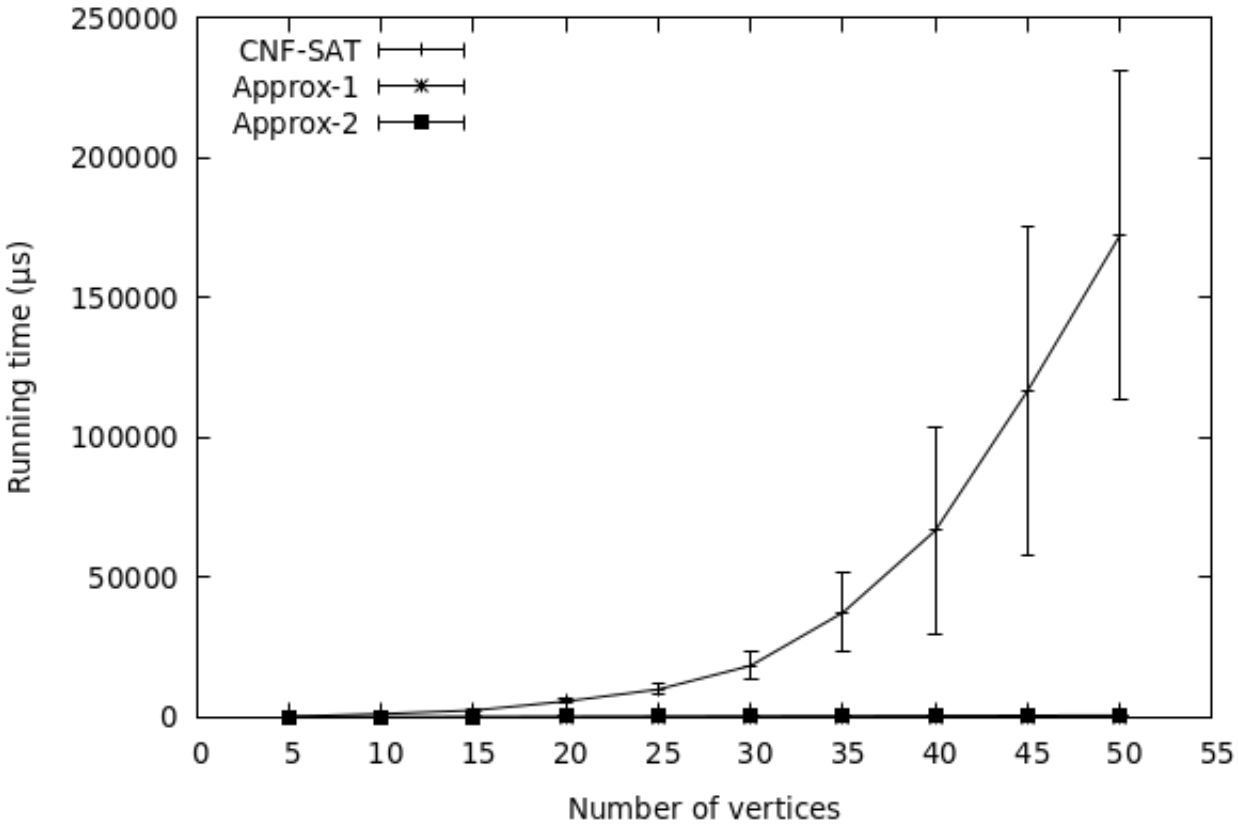

Figure 1: Running time of the three algorithms

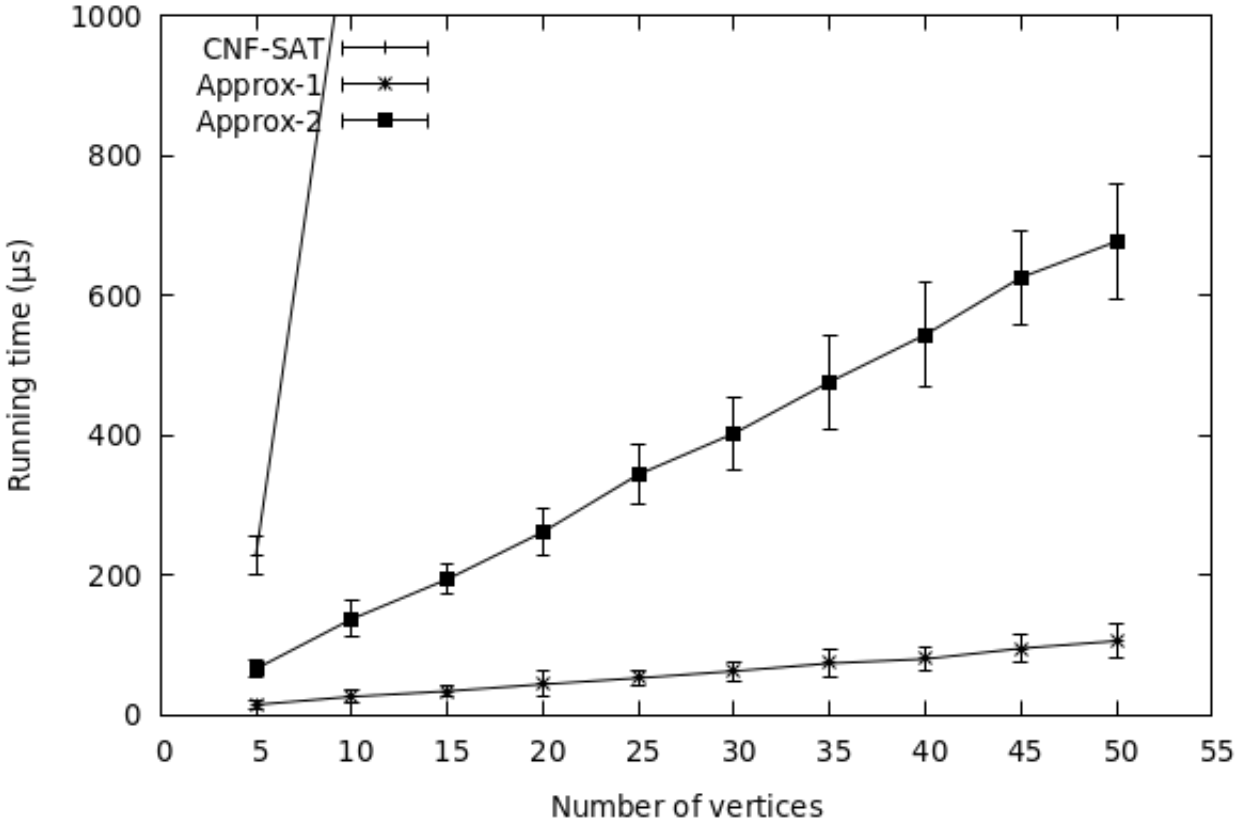

Figure 2: Running time of the three algorithms, fitting into APPROX-1 and APPROX-2

In the time analysis experiment, we utilize the μs as our metric due to the fast speed of C++, as it is shown in link. It is easy to observe that the time needed for CNF-SAT will grow exponentially with the increase of the number of nodes in the graph, since the time complexity is $O(2^{|V|})$, which is proved in link. In contrast, as for the method of both APPROX-1 and APPROX-2, the time will only grow linearly with the increase of the number of nodes in the graph, since their complexity if $O(|V|)$. Moreover, the APPROX-1 will be slightly faster than APPROX-2, since the iterations for APPROX-1 will be faster, which is contributed by less randomness compared with APPROX-2.

As for the standard deviation, APPROX-1 and APPROX-2 have low standard deviation. For APPROX-2, it increases a little bit with the number of vertices, meaning the running time of them does not change much for different graphs. However, for CNF-SAT, the standard deviation increases a lot when the number of vertices increases, meaning when running it for a large graph, the running time varies a lot for different graphs. It appears that the complexity of different graphs varies a lot when the graph size is large, causing the SAT solver to take different times to solve different instances of graphs.

2 Approximation Ratio Analysis

Same as running time, the way we run the experiment is to first pick the number of vertex from 5 to 50, with 5 as a step. For each number, we generate 10 different graphs randomly and run each of them 10 times. The result is shown in Figure 3.

The approximation ratio is calculated by dividing the result vertex cover size by the minimum vertex cover size. Overall, APPROX-1 is more efficient than APPROX-2, which

is kind of desired, because APPROX-2 adds both vertices of an edge into the result vertex cover, which only one of them is needed to cover this edge.

The average approximation ratio of APPROX-1 is nearly 1, while for APPROX-2, the average approximation ratio is around 1.6. The average approximation ratio remains similar for different numbers of vertices, which means these two algorithms do not change the efficiency much when the number of vertices changes.

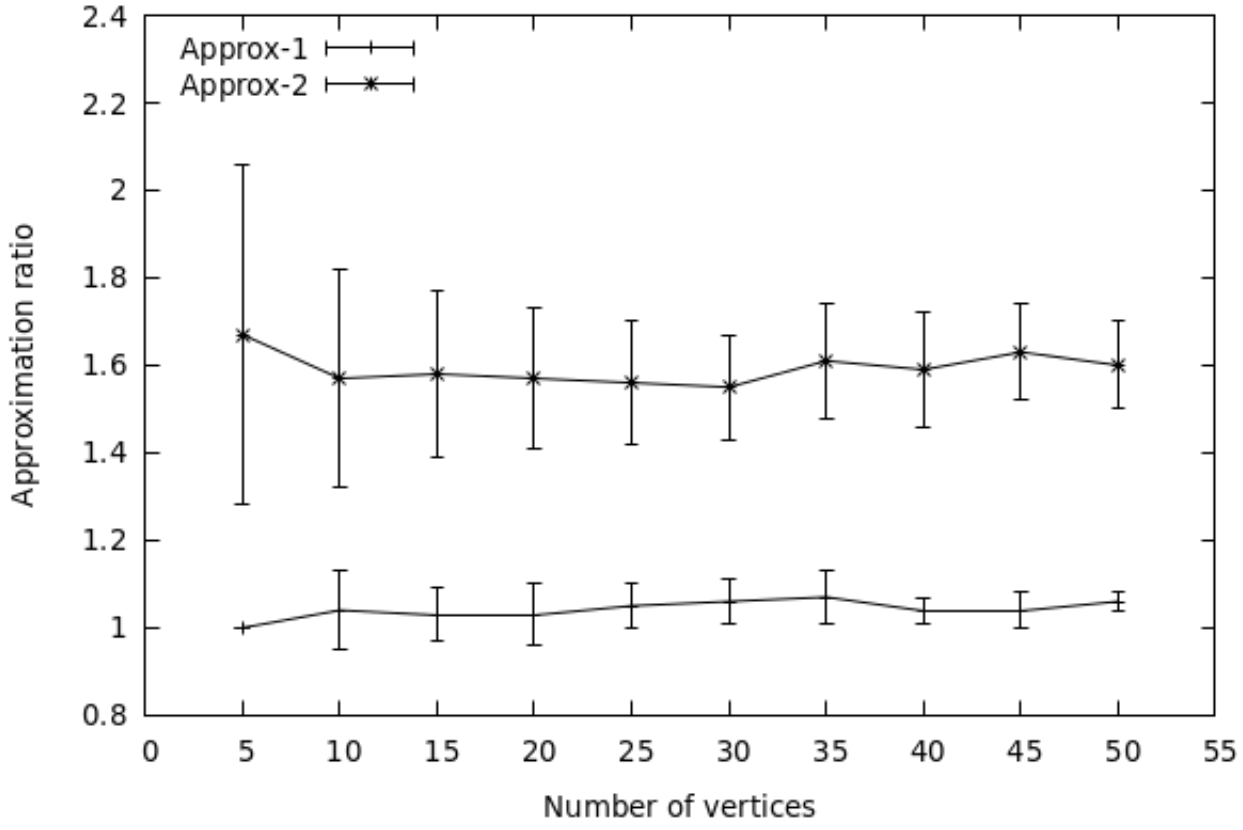

Figure 3: Approximation ratio of APPROX-1 and APPROX-2

For standard deviation, APPROX-1 appears to be non-related to the number of vertices. It is also not very high, meaning it is also not related to the graph being generated. For APPROX-2, its standard deviation is larger when the number of vertices is low, and it becomes smaller and stabler when number of vertices becomes higher. This means when dealing with a larger graph, the efficiency does not change much across different graphs for APPROX-2, but when the graph is small, the graph may affect its efficiency for APPROX-2.

In general, considering the very low running time of APPROX-1 and APPROX-2 compared with the CNF-SAT, if a scenario does not require a very precise solution for vertex cover, we may consider choosing APPROX-1, which can produce a nearly optimal solution in a very short period of running time.

3 Encoding Improvement for CNF-SAT

To improve the running time of CNF-SAT, the encoding is a very crucial part. Different encodings may result in different sizes of the number of clauses, the size of a certain clause, and the number of variables that need to be assigned a true or false value. With observing

the sub-problem of vertex cover is to find at most k or exact k vertices among n vertices, we choose to use the Efficient CNF Encoding of Cardinality Constraints [1] introduced by Olivier Bailleux and Yacine Boufkhad, which is an efficient way of solving the problem of finding exact k from n elements.

The encoding of the original vertex problem for finding a vertex cover of size k from a graph G(V, E) vertices is now composed with below two sets of constraints: (1) find k distinct vertices out of $|V|$ to be assigned true. (2) for each edge in E, make sure at least one vertex of this edge is assigned true.

The second set of constraints remains the same as the original solution, and the first one can be encoded with the new encoding mentioned above. To be more specific, the Cardinality Constraints encoding performs like an adder, which adds up the number of vertices among all the $|V|$ vertices and represents its value with k. The main point is to use unary representation, where the number is represented as consecutive '1' followed by consecutive '0'. And by binary splitting the number of vertices and recursively using this adder for the smaller-sized problem, it could encode it with $O(|V|log(|V|)$ variables and $O(|V|^2)$ clauses.

There are two points we figured out during the experiment of this improvement of encoding.

First, we need to reduce the number of variables, as well as the number and size of clauses. The reason is obvious, that the SAT solver needs to assign the value of these variables and verify these clauses. The number of variables and the clauses' count and size affect this directly.

Second, we need to remove symmetrical solutions when considering the encoding. Let's take the original encoding as an example. The original encoding creates $k * |V|$ variables, representing that if a certain vertex is chosen at position k of the vertex cover. This includes many symmetrical solutions, that for a valid choice, we can simply create another valid choice by re-ordering the result, making some of the vertices to be assigned to a different position. When we run the SAT solver for a true instance of the problem, this helps because we just need to find one of the vertex covers. However, during the whole algorithm, we may need to eliminate many false instances of the problem. When we have many symmetrical solutions, the SAT solver needs to verify them all before it can return the result that it is unjustifiable. The above new encoding removes these symmetrical solutions by forcing the '1' to be at the front of '0', which means for the encoding of picking 5 vertices out of 8. it will force the value of k to be 11111000, eliminating other possible order of the 1 and 0. This increases the number of clauses, however, it could bring up the efficiency a lot. We did another run to compare this improved encoding with the original encoding in A4, which is shown in Figure 4.

We observe a big improvement in running time with the improved encoding. For the original A4 version, it takes an average of about 8 seconds for running graph with 15 vertices, while the improved encoding only takes an average of about 4 seconds for 75 vertices and an average of about 13 seconds for 80 vertices.

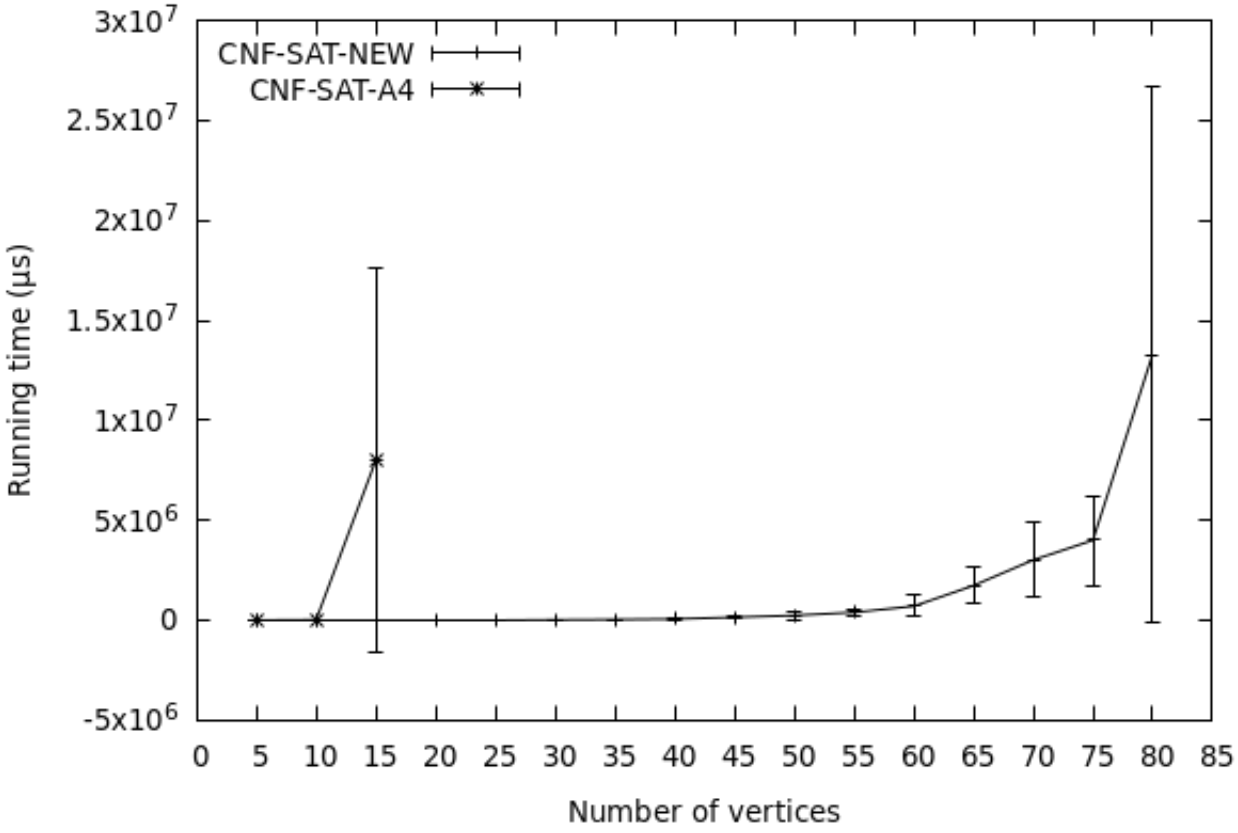

Figure 4: Running time of the improved encoding and original A4

4 References

[1] Bailleux, Olivier, and Yacine Boufkhad. "Efficient CNF encoding of boolean cardinality constraints." International conference on principles and practice of constraint programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.