ECE650
Methods and Tools for Software Engineering

Final Project Report

Xiaoyi Zheng Student ID: 20827166
Rongzhi Gu Student ID: 20855042

1 Introduction

In this report, we mainly compare the efficiency of three methods which aim to solve the problem of VERTEX-COVER. The three methods are CNF-SAT-VC, APPROX-VC-1, and APPROX-VC-2. There are two facts affect the efficiency: running time and approximate ratio and we analyse them in section 3.

1.1 Methods

The principle of CNF-SAT-VC is to construct a propositional logic formula which is in Conjunctive Normal Form and then used the minisat to solve it. The satisfying assignment from the minisat is the solver of the VERTEX-COVER. The other two methods are much more easier than CNF-SAT-VC, one of them firstly picks a vertex which has the largest number of degrees, then choose it as one of the results and throw away all edges which on that vertex. Repeat this process until there is no edge. The other one picks an edge arbitrarily and then adds two vertices which are connected by this edge to the Vertex Cover. After that, throw away all edges attached to these vertices. Repeat it until no edges remain.

1.2 Input

The inputs come from the graphGen which could generate graphs with the same number of edges for a particular number of vertices. The graph could be different each time. To compare the running time and approximation ratio among the three methods, we initially generate graphs for the number of V from 5 to 50, increased by 5. For each value of V, we generate at least 10 graphs, and each graph we run 10 times to compute the mean value and calculate the error bars.

2 Improvement of encoding

We find that the factors affecting the speed of the minisat to solve the CNF mainly depend on the number of literals and clauses. The more literals, the longer time it will take, and the more clauses, the shorter time it will take. The reason of the former is obvious and the latter is mainly due to the additional constraints. Therefore, as long as we maximize the number of restricted conditions, the algorithm can be improved.

2.1 Set useless literals to false

The reduction uses $n \times k$ atomic propositions(n is the number of vertexes,k is the number of VERTEX-COVER). We consider it as a matrix. For instance, if there are five vertices and the number of VERTEX-COVER is three, we have a 5×3 matrix. We know that a VERTEX-COVER is a list of k vertexes,which means the vertexes in the list are in order. So there will be k! solutions for each computation if k exists. However, we don't expect that. In hence, we could set the literals which located above the diagonal of the matrix false initially to reduce the number of solutions (Figure 1).

$$\forall m \in [0, k-2], \forall n \in [m+1, k-1], a\ clause(\neg x_{m,n})$$

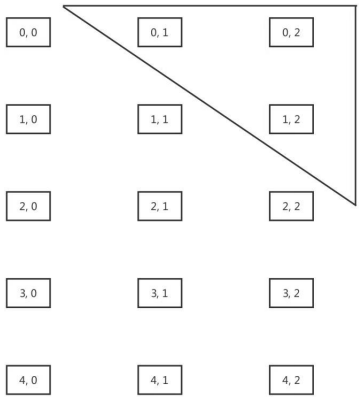

Figure 1: set literals located in triangle areas to false

In this instance, if V 0 , V 3 and V 4 is the solution of the problem, V 0 could only be in the first position. Compare with the original algorithm, the number of solutions reduce from 6 to 2. This method could improve the algorithm to some extent and reduce the running time. However, this method could only be useful when most covering vertices are locate before the k-th row. If the covering vertex is V 2, V 3 and V 4, this method will be useless. Therefore, this method could only compute graphs which contain up to 20 vertices.

2.2 Fix the location of all solutions

As mentioned above, the number of solutions will affect the speed of the algorithm. A vertex cover of size k could have k! solutions. Inspired by this, if we only have one solution, it could improve the algorithm and shorten the running time. To achieve that. we have to add clauses to make sure that the solution can only be in one order. For example, if the solution is V 0,V 3 and V 4, we only require one solution in which $x_{0,0},\ x_{3,1}\ and\ x_{4,2}$ are true and other permutations have to be false. That is: a vertex with a smaller index number must be located before one with a larger index number in vertex cover list. Therefore for any two elements in the matrix, we could not allow $x_{y,p}\ and\ x_{z,q},\ where\ y, z \in [0, n-1]\ with\ y < z,\ p, q \in [0, k-1]\ with\ p > q$ to be true at the same time (figure 2).

$$\forall y, z \in [0, n-1]\ and\ y < z, \forall p, q \in [0, k-1]\ and\ p > q,\ a\ clause(\neg x_{y,p} \vee \neg x_{z,q})$$

In this condition, we fix the order of the solution. The number of solutions decrease from k! to 1, which greatly improve the efficiency of the algorithm.

3 Analysis

In this section, we mainly analyze how efficient each approach is. The efficient is characterized by two factors, running time and approximation ratio. We create three threads to allow each approach could run independently.

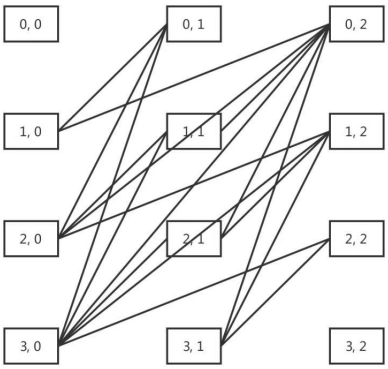

Figure 2: Any two literals connected by line can not be true simultaneously

3.1 Running time

The first factor is running time. As mentioned before, the three methods run in three threads independently and concurrently. Each execution time is measured by the clock_gettime function and the cpuclockid of each thread is obtained by phthread_getcpuclockid from the phthread library. Because the magnitude difference of time between CNF-SAT-VC and the other two methods is too large, in order to facilitate observation, we plot them in two figures.

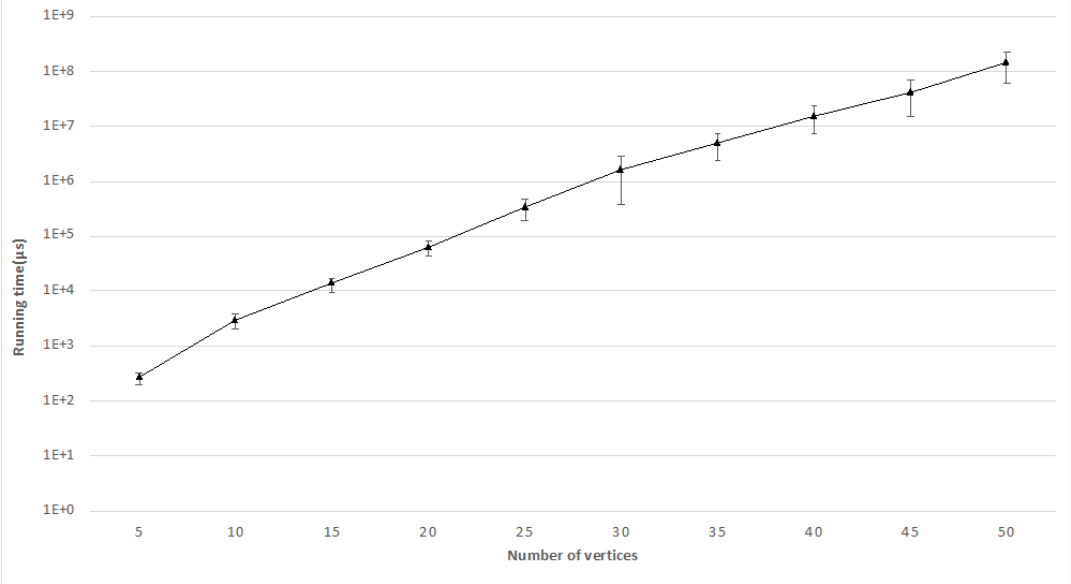

Figure 3: The running time of CNF-SAT-VC

As it can be seen in the figure 3, the running time of CNF-SAT-VC grows exponentially with the increment of vertices given that the y-axis uses a logarithmic scale. The reason behind that could be related to the increment of literals. As the vertices added by n, the number of

covering vertices k will increase accordingly, which result in the number of literals will increase by $n \times k$. We can also find that the standard deviations vary greatly with the increment of vertices. This is resulted by k. For graphs with the same vertices, even their numbers of edges are identical, their k may be different. As mentioned before, the value of k greatly affect the running time. For the graphs that contain even the same number of vertices, when the cover size of one graph is k, it will take much shorter time to compute than one another that has a k+1 cover size. So that could explain why the standard deviation of the running time corresponding to some certain numbers of vertices are relatively large.

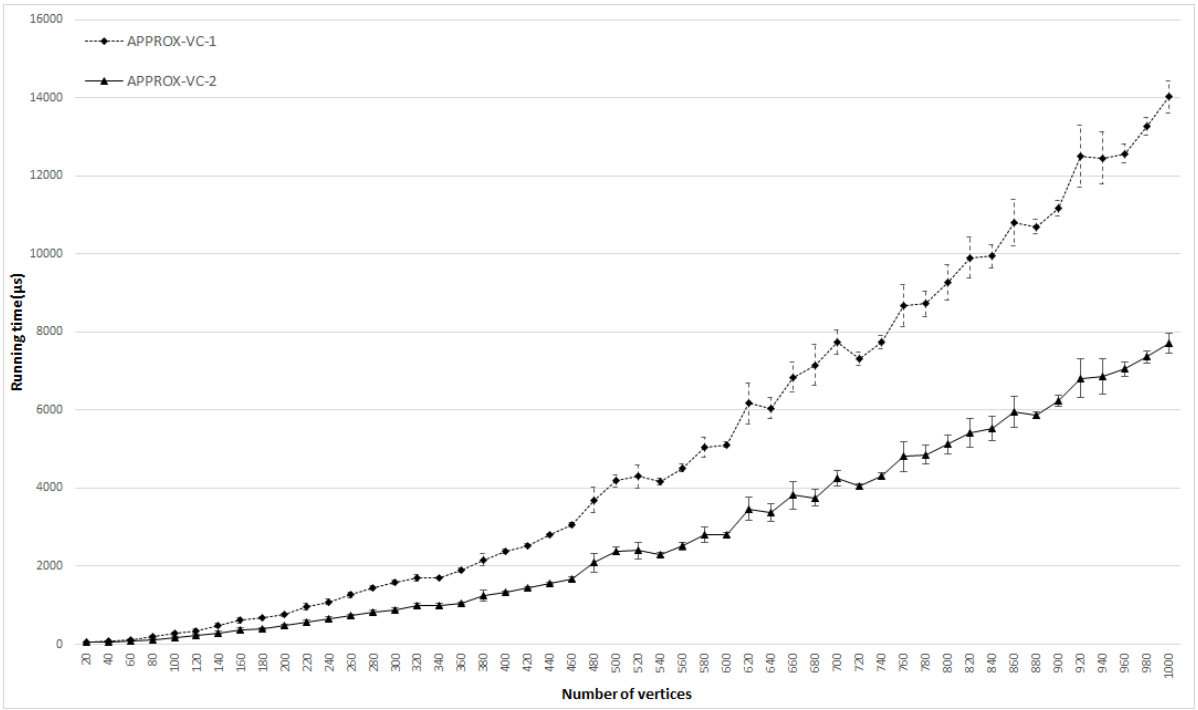

Figure 4: The running time of APPROX-VC-1 and APPROX-VC-2. Some error bars are not visible because the standard deviation is small.

The running time of other two methods is shown as figure 4. Compare with the CNF-SAT-VC, both of them take a much shorter time to solve the problem. To find the tendency of them, we increase the vertexes to 1000 and each graph increased by 20 vertexes, when the number of vertices comes to 360, they take about 2000μs and 1000μs respectively which still less than when CNF-SAT-VC computes 10 vertexes. We can find that both of them grow quadratically, this trend is more obvious when the number of vertices is thousands or more. Their time complexity is $O(|E|^2)$. The reason why APPROX-VC-1 takes more time than APPROX-VC-2 is that APPROX-VC-1 has to compute the degrees of each vertex and choose one with the highest degree using an $O(|E|)$ time complexity algorithm every time before throwing away edges, whereas APPROX-VC-2 picks an arbitrary edge instead.

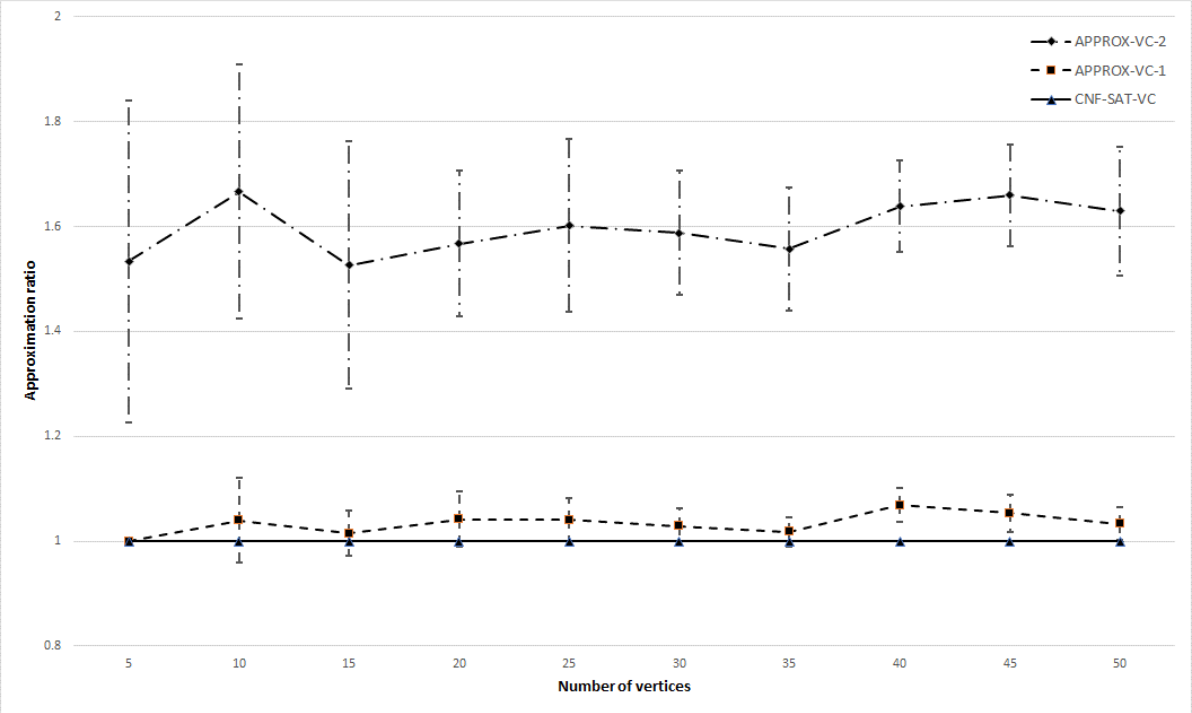

Figure 5: The approximation ratio of three methods

3.2 Approximation ratio

The other important factors is approximation ratio. We assume that the CNF-SAT-VC could give the optimal solution so that the approximation ratio is characterized as the ratio of the size of the computed vertex cover to the size of vertex cover which computed by CNF-SAT-VC. The result is shown as figure 5. As we can see that the approximation ratio of APPORX-VC-1 is highly close to the CNF-SAT-VC and the error bar of each number of vertices is also relatively small. The reason behind it could be that APPROX-VC-1 uses an approximate greedy algorithm. It picks a vertex which has the highest degree and add it to the vertex cover, which guarantee that the vertices in the cover list are the most efficient to some extent. In contrast, the approximation ratio of APPROX-VC-2 is much worse and unstable. When the number of vertex is 10, it has a spike which arrives at about 1.63. It mainly relays on its algorithm. It choose an edge randomly and don't consider its effectiveness. If it chooses an edge whose vertices have relatively small degrees, then it has to pick more edges to complete the algorithm, which results in more vertices in the vertex cover computed.

3.3 Comparison before and after improvement

The running time of CNF-SAT-VC before and after improvement is shown as figure 6. As we can see the approach before improvement could only compute the graphs which contain up to 15 vertices. After 15 vertices, it will run out of time (5 minutes). In addition, the running times vary a lot before and after improvement. The reason is that before improvement, there are many permutations of solutions, which means that most literals are uncertain. That

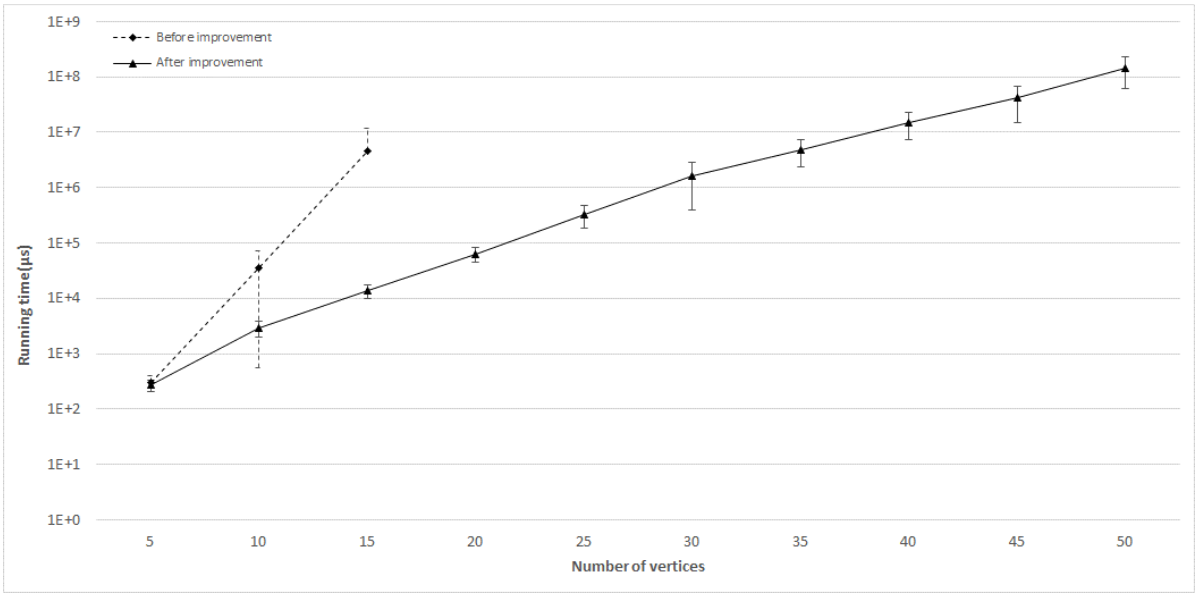

Figure 6: The running time of CNF-SAT-VC before and after improvement

significantly increase the time of solution. After improvement, a certain number of clauses have been added which means the number of multiple solutions has been reduced, which could explain that the improved algorithm has smaller standard deviations and takes much less time than before the improvement.

4 Conclusion

In this report, we mainly improve the encoding of CNF-SAT-VC and analysis the efficient of three different approaches. The CNF-SAT-VC is the most optimized among the three methods. When the number of vertices below 30, it will take about one second to give the solution, which means we could not feel the difference of time when three approaches run at the same time. However, when the number of vertices is more than 30, we can see that this solution computes much slower than the other two approaches. In addition, the size of vertex cover of APPROX-VC-1 is similar to that of CNF-SAT-VC but APPROX-VC-1 takes much shorter time.
To sum up, in practical applications, if the scale of a graph is not too large, the CNF-SAT-VC is the best method, otherwise CNF-SAT-VC-1 can be taken into consideration if some error or additional overhead is allowed to some extent. Besides that, if a optimal vertex cover solution is not necessarily needed and running time is considered more important, APPROX-VC-2 could be chosen as a compromise.