Comparison on Algorithms for Vertex Cover Problem

Tiankai Jiang Chuxi Zhang

November 30, 2019

Abstract

This paper presents the reduction of Vertex Cover Problem to CNF Satisfiability Problem using a naive encoding method and sequential counter encoding, then it presents the comparison on time efficiency and approximate ratio between the exact solution of Vertex Cover using *miniSAT* and two of the approximate solutions.

1 Introduction

The Vertex Cover Problem: given a graph $G = (V, E)$, a positive integer $K \leq |V|$, $\exists V' \subseteq V$, $\left|V'\right| \leq K, \forall \{u, v\} \in E : \{u, v\} \cap V' \neq \emptyset$?

The CNF-SAT Problem: given a boolean formula $f(x_1, \ldots, x_n)$ in conjunctive normal form, is f satisfiable?

Cook–Levin theorem[1] states that Boolean Satisfiability Problem(**SAT**) is NP-complete. By reducing the SAT problem to CNF-SAT problem, we can prove the latter is also NP-complete. And by reducing CNF-SAT to Clique problem, which can be further reduced to Vertex Cover Problem, we know the Vertex Cover Problem is NP-complete as well. Hence, Vertex Cover Problem can be solved using a SAT solver by reducing it to CNF-SAT Problem.

2 The Reduction

Given a graph $G = (V, E)$, let $n = |V|$, let $x_i\{0, 1\}, i \in n$ denote if the ith vertex is in the vertex cover, for a vertex cover of k vertices, two conditions must be met[2] for a valid reduction:

- For an arbitrary $e\langle u, v\rangle \in E$, at least one of $\{x_u, x_v\}$.

- $\sum_{i=1}^{n} x_i \leq k$

The first condition ensures that at least one vertex of an edge is selected therefore that edge is covered. And the second condition, which is called boolean cardinality constraint, ensures that there are at most k vertices selected.

It is straightforward to express the first condition in conjunctive normal form:

$$\bigwedge_{\langle u,v\rangle \in E} x_u \vee x_v$$

But it is not that easy for the second one. In fact, the encoding of cardinality constraint has a great impact on the performance of the SAT solver, which we will discuss in the next section.

3 Encoding of Boolean Cardinality Constraints

We will present the sequential counter encoding[3, 4] and its advantage over the encoding given in ece650.a4.

3.1 The Sequential Counter Encoding

Given $x_1 \ldots x_n$, the encoding introduces an extra of $(n-1) * k$ variables, denoted $s_{i,j}$, where $i \in [1, n-1], j \in [1, k]$. The k variables in $s_{i,*}$ represent the number of variables in $x_1 \ldots x_n$ that are true in base 1.

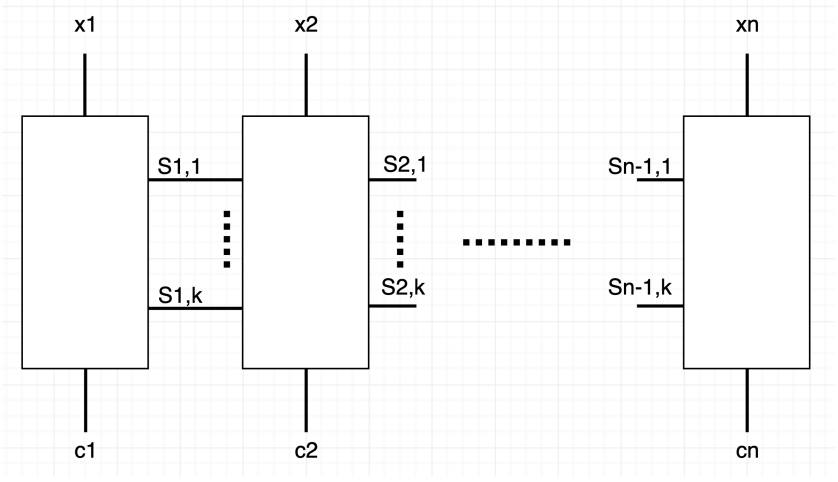
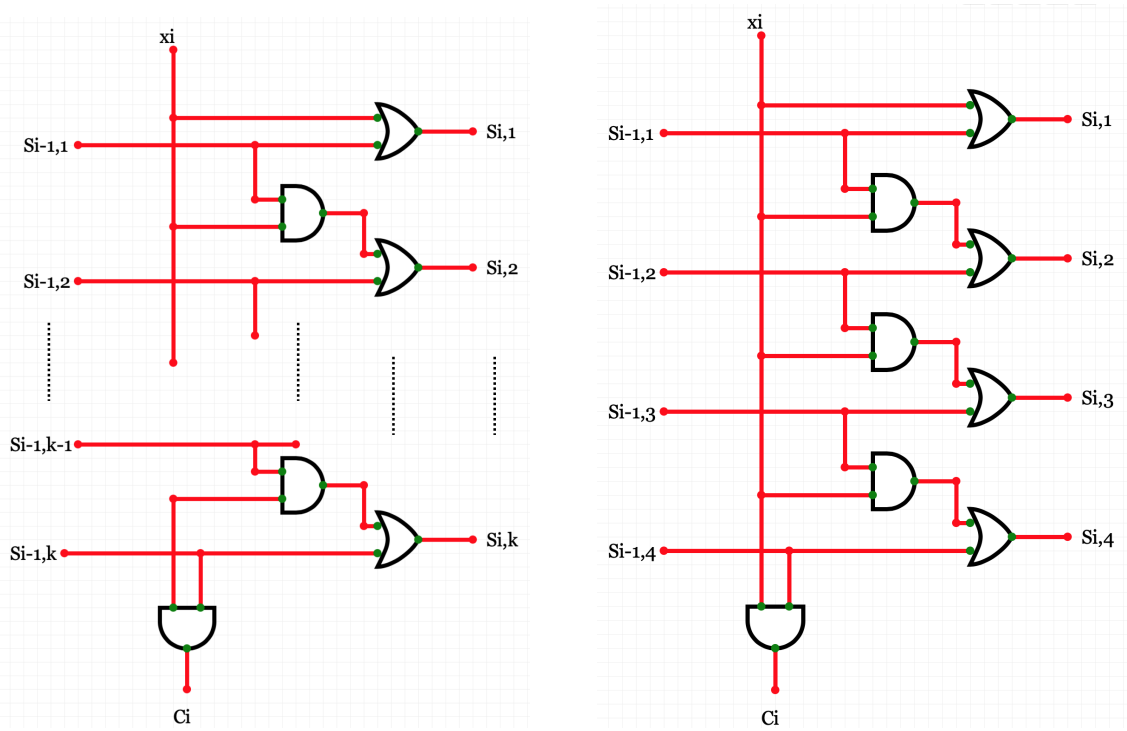


Figure 1: The Sequential Counter Circuit



(a) Sub-circuit of the sequential counter

(b) Demo counting using a 4 bits input

Figure 2: Sub-circuits

Figure 1 shows the sequential counter circuit. Initially, the counter is set to 0. And the circuit will count the number of 1 in $x_1 \dots x_n$. As long as the overflow bit c_i is not 1, $\sum_{i=1}^{n} x_i \leq k$ holds.

Figure 2(a) shows the implementation of the sub-circuit in Figure 1. The sub-circuit takes a number in base 1 from $[s_{i-1,1}, s_{i-1,k}]$, add $x_i(0 \; or \; 1)$ and output the results as a number in base 1 to $[s_{i,1}, s_{i,k}]$. Figure 2(b) shows a demo when $k = 4$. Suppose the input is 2(1100 in unary), and x_i is 1, after the addition, the output will be 3(1110 in unary). If the input is 4(1111) instead, after adding $x_i = 1$, the overflow bit c_i is 1 and we will know that the result exceeds 4.

The encoding are as follows:

Since x_i connects directly to an OR gate in each sub-circuit, $s_{i,1}$ must be 1 if x_i is 1.

$$\bigwedge_{i=1}^{n-1} \neg x_i \vee s_{i,1}$$

Since there is no input in the first sub-circuit, the output of it is at most 1, which means $s_{1,2}$ to

$s_{1,k}$ are guaranteed to be 0.

$$\bigwedge_{i=2}^{k} \neg s_{1,i}$$

Since $s_{i-1,j}$ connects directly to an OR gate in each sub-circuit, $s_{i,j}$ must be 1 if $s_{i-1,j}$ is 1.

$$\bigwedge_{i=2}^{n-1} \bigwedge_{j=1}^{k} \neg s_{i-1,j} \vee s_{i,j}$$

Since x_i and $s_{i-1,j-1}$ are connected to a AND gate and then the AND gate connects to an OR gate, $s_{i,j}$ must be 1 if x_i and $s_{i-1,j-1}$ are both 1.

$$\bigwedge_{i=2}^{n-1} \bigwedge_{j=2}^{k} \neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}$$

There should not be any overflow in all n sub-circuits.(i starts from 2 because there is no input for the first sub-circuit).

$$\bigwedge_{i=2}^{n-1} \neg x_i \vee \neg s_{i-1,k}$$

3.2 Encoding Efficiency

The encoding method to represent cardinality constraint in ece650.a4 uses in total $n * k$ atomic propositions and $k + n\binom{k}{2} + k\binom{n}{2}$ clauses. The sequential counter encoding also uses $(n-1) * k + n$ atomic propositions, and in total $(n+1)+(k-1)+k(n-2)+(k-1)(n-2)+(n-2) = 2nk+n-3k-2$ clauses. In the worst case, when $n = k$, the naive encoding uses n^3-n^2+n clauses while the sequential counter uses only $2n^2 - 2n - 2$ clauses, which is significantly better than the previous one. When $k = \frac{n}{2}$, the former uses $\frac{3n^3-4n^2+4n}{8}$ clauses while the latter uses $\frac{2n^2-n-4}{2}$ clauses. The sequential counter is still much better.

4 Approximation Algorithms

For a NP-complete problem, an exponential running time algorithm for an exact soluton is expected. To reduce the time complexity, approximate solutions are considered. We apply greedy algorithm and random algorithm on Vertex Cover Problem.

4.1 The Greedy Algorithm

Choose a vertex with highest degree each time and remove all its edges until no edge remains. The time complexity is $O(|V|+|E|)$. Assume set C contains the vertices the algorithm chooses, m is the number of vertices in the optimal solution and H_m the Harmonic series.

$$|C| = \sum_{i=2}^{m} |C_i| = \sum_{i=2}^{m} \left\lfloor \frac{m}{i} \right\rfloor \geq \sum_{i=2}^{m} (\frac{m}{i} - 1) \geq m \sum_{i=1}^{m} \frac{1}{i} - 2m = m(H_m - 2)$$

$\frac{|C|}{|C^*|} = \frac{m(H_m-2)}{m} = H_m - 2 = \Omega(\log(n))$ And we get its upper bound by set cover problem.[5]. Hence, this is a $\theta(\log(n))$-approximation algorithm.

4.2 The Random Algorithm

Randomly choose an edge each time and remove both of its vertices' edges until no edge remains. The time complexity is $O(|V|+|E|)$(adjacency list representation). Assume that set A includes all the edges chosen. Every time we can put two vertices into vertex cover set C. Hence, $|C| = 2|A|$. Assume C^* is the optimal solution, which must cover A. For no edges in A share a common vertex and C^* includes at least one vertex of each edge in $A, |A| \leq |C^*|, |C| = 2|A| \leq 2|C^*|, \frac{|C|}{|C^*|} \leq 2$. Hence, this is a 2-approximation algorithm[5].

5 Performance Evaluation

5.1 Experiment Setup

Randomly generate 10 graphs for each $|V| \in [5, 50]$ in increment of 5 and test the CNF-SAT(two encodings) and two approximate algorithms with each graph for 10 times. Calculate mean and standard deviation of running time for each $|V|$ for all four algorithms(CNF-SAT with two encoding methods) and calculate approximate ratio for the two approximate algorithms. Randomly generate 10 graphs for each $|V| \in [50, 2000]$ in increment of 25 and test two approximate algorithms. Calculate the average run time and vertices selected.

5.2 Experiment Result

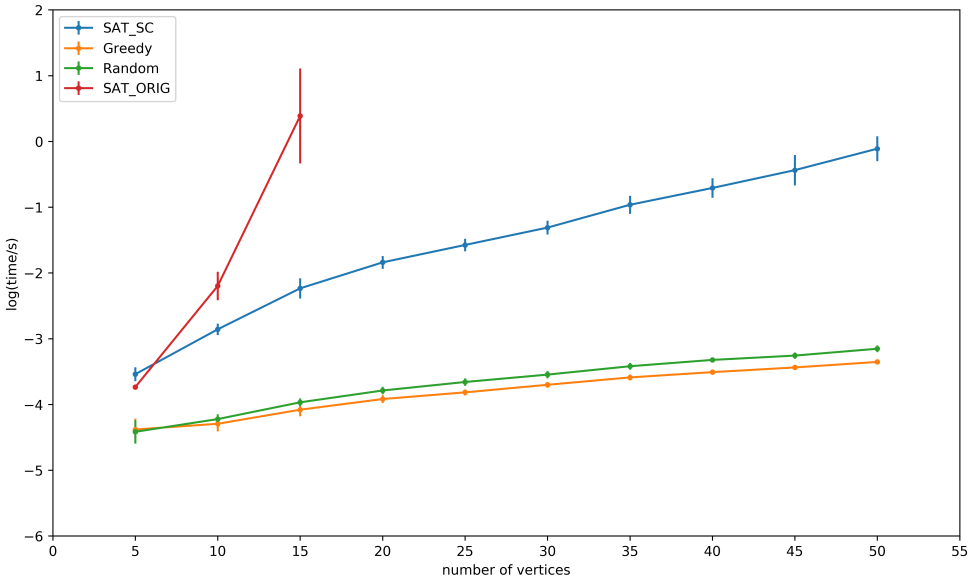


Figure 3: Log Run Time of the Four Algorithms

Figure 3 shows the log running time of the algorithms vs the graph size. In the plot, SatOrig refers to the exact algorithm with encoding in ece650.a4. The SatSC refers to the exact algorithm with sequential counter encoding.

The slope of SatOrig and SatSC is relatively large, indicating *miniSAT* uses algorithm with exponential running time. There is a huge performance gap between SatOrig and SatSC due to different encodings of the same problem. The former times out when $n \geq 20$ while the latter could handle an input of at least 50 vertices. Another difference between these two encodings is their standard deviation. SatOrig has a much larger standard deviation, which means for different graphs with the same number of vertices, the running time varies a lot, which also indicates this kind of encoding will generate much more clauses as k grows: for a fixed vertex number n, it will take less time if a graph is dense. But if the graph is sparse, the original encoding will make *miniSAT* run for a much longer time. This is not a big problem for the sequential counter encoding.

The slope of the two approximate algorithms is relatively small, indicating polynomial time algorithms. And both of them are faster than the exact algorithm in tens of orders.

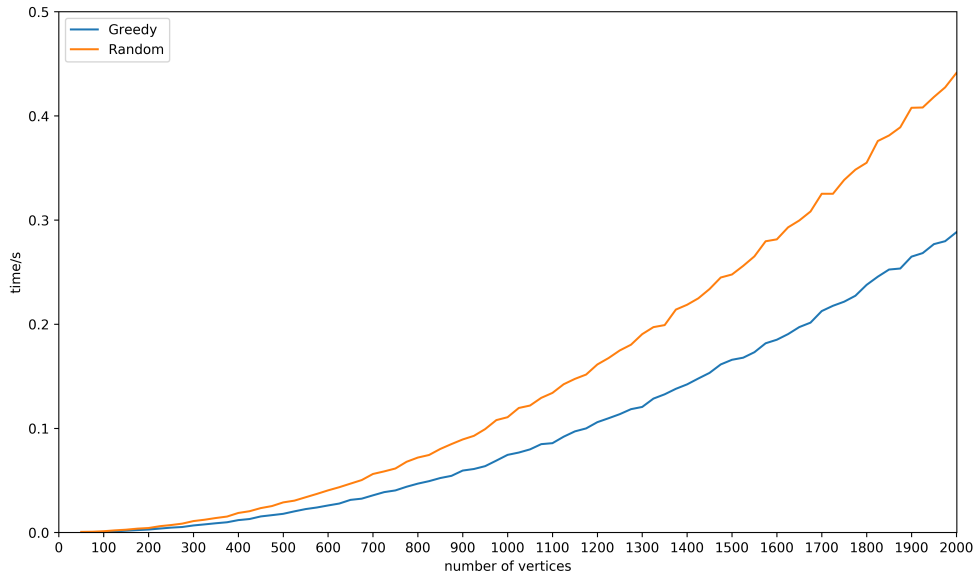


Figure 4: Run Time of the Two Approximate Algorithms

Figure 4 shows the run time of them when n becomes larger. The greedy algorithm is still faster than the random, which may due to the adjacency list implementation of the graph. We cannot pick an edge directly, instead, we can only pick a vertex randomly and then pick another vertex in its adjacency list, which may slow down the algorithm.

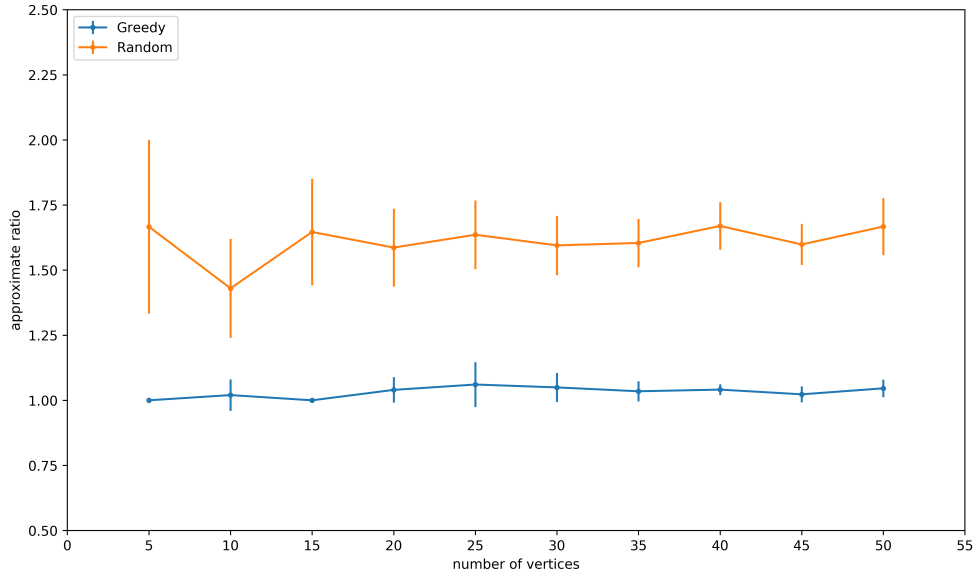


Figure 5: Approximate Ratio of Greedy and Random Algorithm

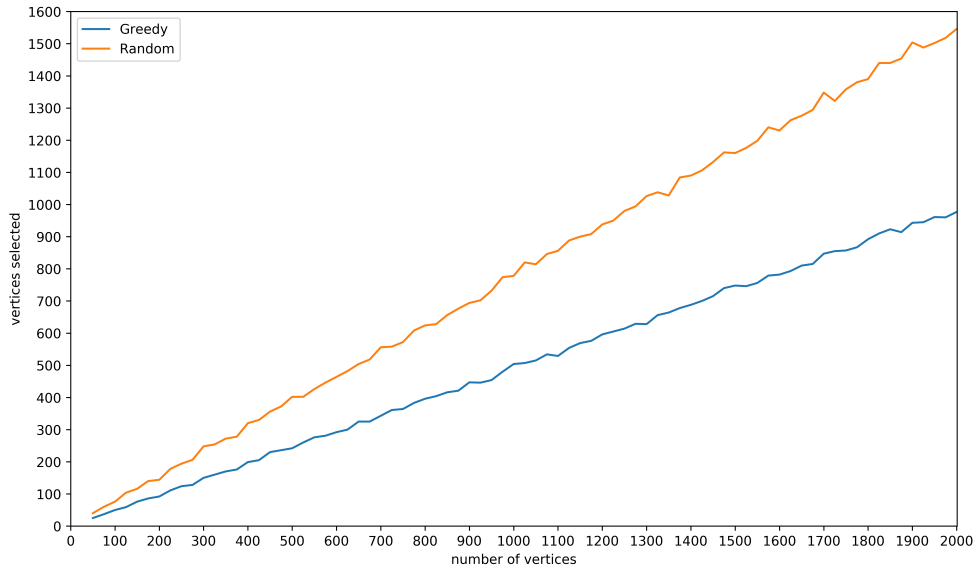


Figure 6: Approximate Ratio of Greedy and Random Algorithm ($n \leq 2000$)

Although in theory, as shown in Chapter 4, the random algorithm has a better appromimate ratio (2) than the greedy algorithm ($\log(n)$), the greedy algorithm outperforms the random one in practice, with an approximate ratio slightly greater than 1, while the random algorithm is about 1.5. As currently the $n = 50$ is not large enough, we cannot conclude which algorithm is indeed better than the other one. When n becomes larger, although we cannot calculate the approximate ratio as the exact algorithm takes too long, we can still compare the number of vertices both algorithm selected, as shown in figure 6. We can say at least when the input graph is small($n < 2000$), the greedy algorihtm is a better choice for an approximate solution.

6 Conclusion

In this paper, we explored two different encodings of reduction from Vertex Cover Problem to CNF-SAT problem. We reached a conclusion that the sequential counter encoding is better than the original encoding in terms of the clauses they produce, which will affect the performance of the CNF-SAT solver. We also compared them with two approximate algorithms, and we found for a small graph($n < 2000$), the greedy algorithm is a better choice for approximation, with a lower approxiamte ratio than the random algorithm.

References

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, (New York, NY, USA), pp. 151–158, ACM, 1971.

[2] H. Stamm-wilbrandt, "Programming in propositional logic or reductions: Back to the roots (satisfiability)," 1993.

[3] P. M. Bittner, T. Thüm, and I. Schaefer, "Sat encodings of the at-most-k constraint," in *Software Engineering and Formal Methods* (P. C. Ölveczky and G. Salaün, eds.), (Cham), pp. 127–144, Springer International Publishing, 2019.

[4] C. Sinz, "Towards an optimal cnf encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming - CP 2005* (P. van Beek, ed.), (Berlin, Heidelberg), pp. 827–831, Springer Berlin Heidelberg, 2005.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.