Analyze Performance of SAT solver with Four different approaches to find Vertex Cover of a graph

Tandra Chakraborty

Department of Electrical and Computer Engineering
{t3chakra}@uwaterloo.ca

Abstract. In this report I have analyzed different approaches to find Vertex Cover of a graph. I use SAT solver and other approximation algorithms for this and then compare their running time and approximation ratio.

1 Introduction

In the mathematical discipline of graph theory, a vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. The problem of finding a minimum vertex cover is a classical optimization problem in computer science and is a typical example of an NP-hard optimization problem that has approximation algorithm(Wiki).Here in this report I run 32 bit SAT solver(SAT)to polynomial time reduction of Vertex Cover problem. And then I develop two approximation algorithms and two refined versions of those algorithms to find vertex cover. SAT solver gives me minimum vertex cover set and the approximation algorithms gives me vertex cover set which is not minimum always. For SAT solver, it checks all possible combinations of vertices to find minimum vertex cover, which is not applicable for approximation algorithms. In the following section I present my result with graph and explain how the running time and approximation ratio changed for different approaches.

2 Analysis

We use polynomial time reduction of vertex cover problem from(Assignment 4 class material). With the main thread I run 5 other threads; each for different approach to solve Vertex Cover Problem. In one thread we run SAT solver to the clauses generated by reduction. In other threads approximation algorithms are run. The implementation details is availabe in (Assignment 5 specification). In the following subsection I discuss my result from these approaches.

2.1 Experimental Setup

I run the program in my local computer. The processor is *AMD Phenom(tm) II X4 955 Processor 4* . It has 7.6Gb Memory, OS: 64 bit Ubuntu 12.04 Lts, Hard disk 270.8Gb. The performance might vary in oter computer with different configuration.

2.2 Running Time

I find SAT solver takes the maximum running time among all approaches. In order to show the result I use three graphs for running time. One for SAT solver Figure1(without logscale), Figure2(with logscale) for better understanding of time and other for all four approaches3. I calculate average running time of graph with vertices 5..15. I use Gnuplot. For getting running time I use Timing function of Pthread.

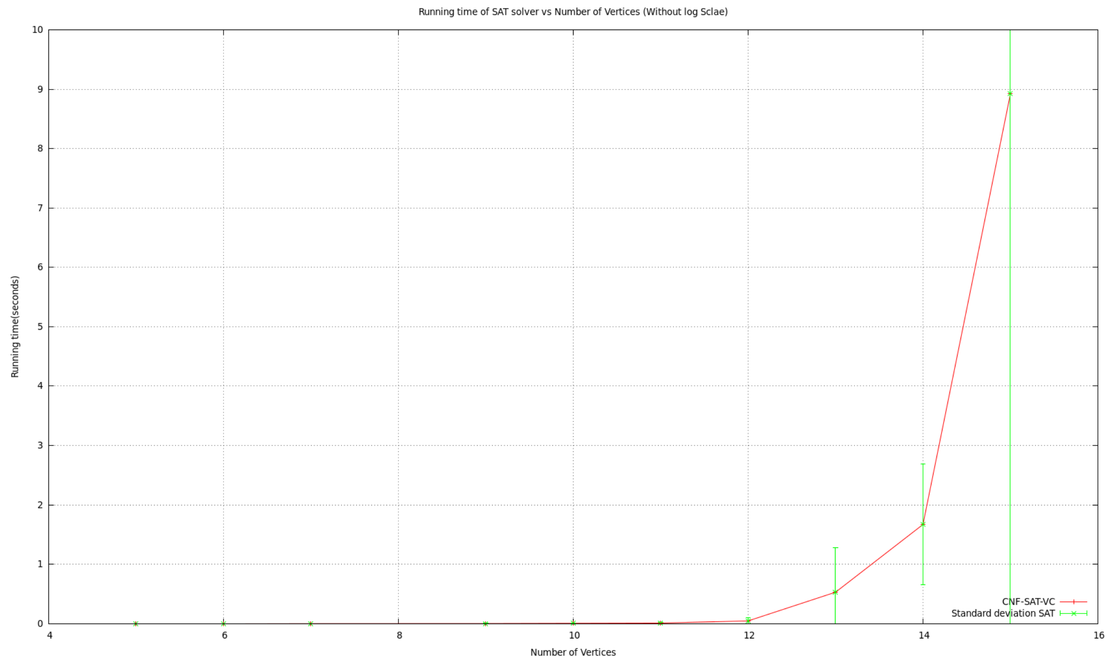

Fig. 1. Running time of SAT solver vs Number of Vertices without Log Scale

From the Figure1 it is clear that SAT solver performs very fast for less number of vertices(5..10). We are using graphGen which produce different graphs with different number of vertices. When number of vertices increase, it takes more time to find vertex cover. We can see an exponential growth between vertex 14 and 15(Figure1). I expect this growth will continue when we run SAT solver for graphs with larger number of vertices. This is because as number of vertices(v) increases and it will increase the number of clauses $k * v * (v - 1)/2$ where k is the size of vertex cover. And also with increased edge numbers it has to satisfy

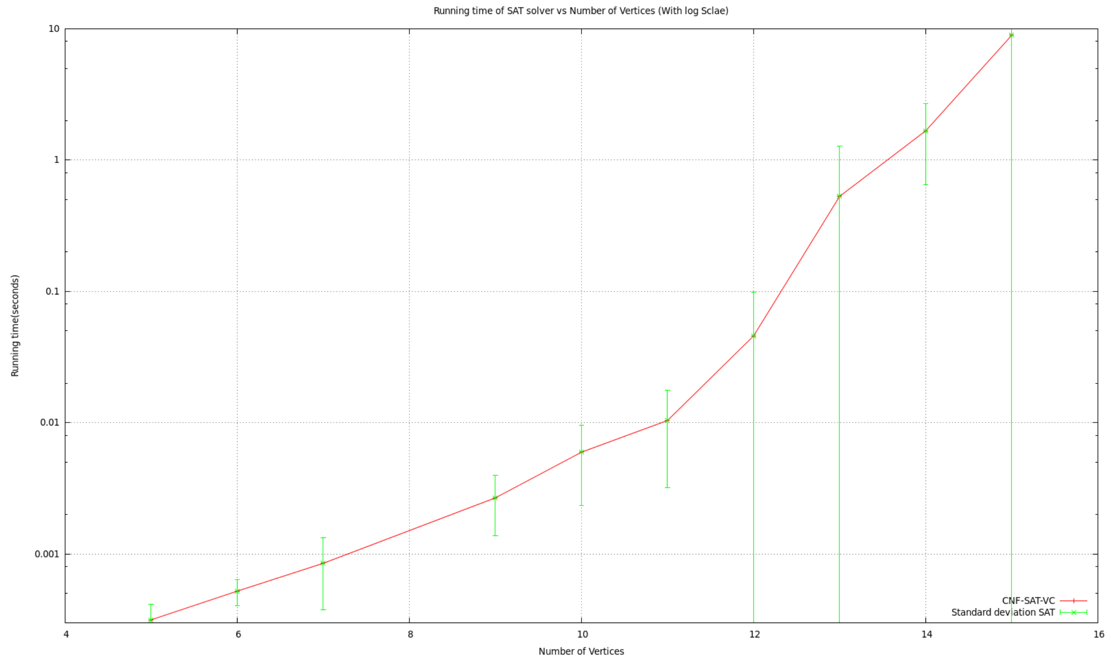

Fig. 2. Running time of SAT solver vs Number of Vertices with Log Scale

more clauses. In case of small number of vertices it takes less number of clauses. So it takes less time as it has to satisfy less number of combinations of variables. So the exponential growth of SAT solver is justified.
Another observation is for vertices 15 standard daviation which is shown by errorbars, is very high. For different graphs with same number of vertices, the running time is varied the most in this case. It is also because different graphs have different connection between edges. Based on that number of vertex cover varied. In case of 15 vertices, this variation is even more. So the running time to calculate them also varied from each other. So we get maximum standard davition in this case.

Running time for all approximation algorithms are increasing with number of vertices(v) in most cases. From the figure3 in most cases Refined Approx VC-1 is taking more time then Approx VC-1. Approx version-2 takes the lowest time in all cases, where Refined Approx VC-2 takes more time than it. None of these approach has exponential growth. This is a basic defference bethween these approaches and SAT solver. As I mentioned before, SAT solver needs to satisfy all possible combinations of variables to find minimum vertex cover. In the case of other approaches, we are not checking all possible combinations.
In Approx VC-1 we are taking vertex with most incident edges and in its refined version we are deleting the vertices which we find unnecessary in our vertex

cover set. The time complexity is $O(v^2)$. As I need to check the whole graph to find vertex with maximum incedent edges, it takes v^2 comparisons, then I am running this process until no edge left. To delete edge, I keep a temporary array of two dimentions. Let number of edges denoted by E. So complexity is $O(v^2)+O(v^2)+O(E)$. So the dominating factor is v^2, so complexity is $O(v^2)$. I use a temporary two dimentional array to keep my graph, and delete edges from that. So space complexity is also $O(v^2)$. In the Refined Approx VC-1 all process is repeated as Approx VC-1 and we get a vertex cover set. After that I keep that vertex cover in a temporary array, and delete unnecesary vertex from that array. It is an one dimentioal array, so space complexity is still $O(v^2)$(counting the space needed for Approx VC-1). It has the similar time complexity for calculating the temporary vertex cover set. Then I am running other loop to delete unnecessary vertex, and this loop is checking whether after deletion of that vertex the vertex cover is still valid or not. If we denote the size of temporary vertex cover as vt it is obvious that, vt can not be greater than v. So that comparison has complexity of $O(v * vt)$. Still it is not more than $O(v^2)$. So the time complexity is $O(v^2)$.

In case of Approx VC-2 I am taking the 1st edge and deleting the edges

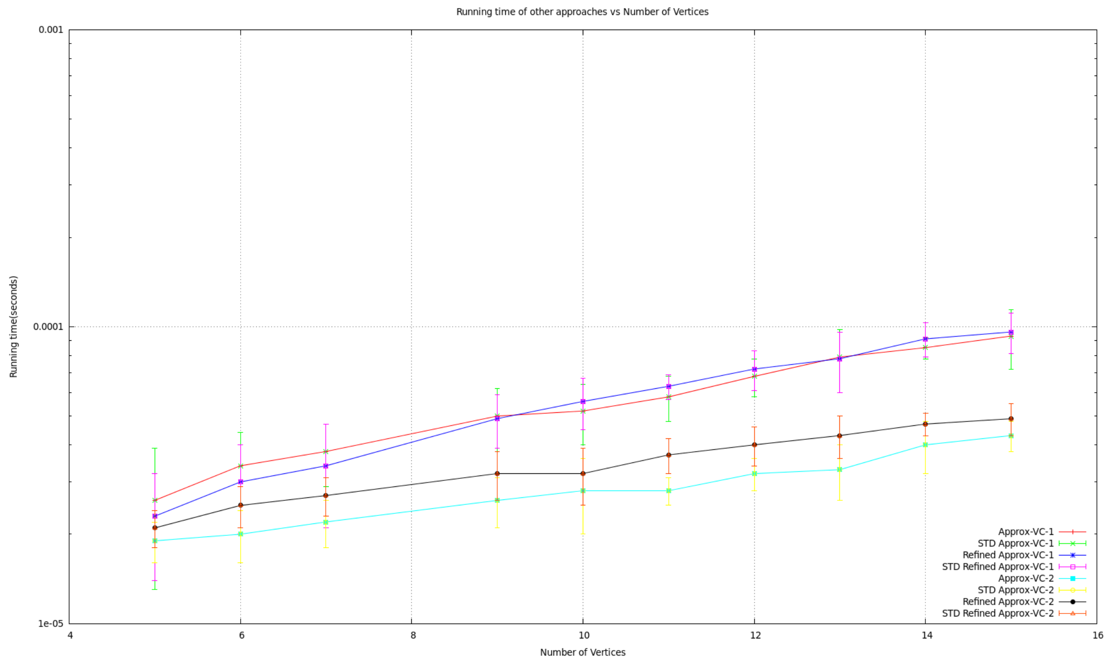

Fig. 3. Running time of other approaches vs Number of Vertices with Log Scale

incedent to the endpoints of that edge. I repeat this until no edge remains in the graph. So time complexity is $O(E)$. Here also I have a temporary array,for the same reason as Approxc VC-1. It takes $O(v^2)$. So the time complexity is $O(v^2) + O(E)$. We can notice that time complexity is still $O(v^2)$ but we don't have the v^2 comparisons this time to find vertex with maximum incedent edges. So this is reducing the running time. It is also visible in figure3. The space complexity is similar as Approx VC-1, $O(v^2)$. The Refined Approx VC-2 has similar space and time complexity as Refined Approx VC-1. It is working on the output of Approx VC-2, which is taking less time as I mentioned above. For the same reason Refined Approx VC-2 is taking less time than Approx VC-1 and Refined Approx VC-2 as can see in figure3.

2.3 Approximation Ratio

I calculate approximation ratio by the following formula:
$ApproximationRatio_i = \frac{sizeofvertexcoverbyCNF-SAT}{sizeofvertexcoverbyapproach_i}$.
The result is presented by figure4. The algorithm which results approximation

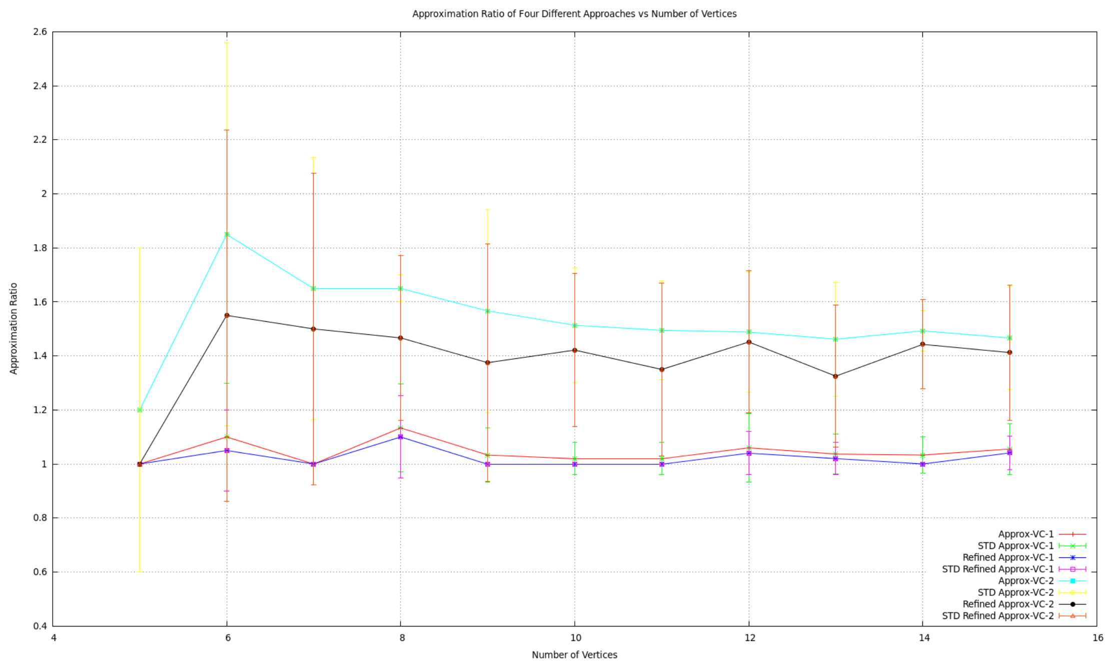

Fig. 4. Approximation Ratio of Different Approaches vs Number of Vertices

reatio close to 1 is a good approximation algorithm. We can see that Both refined versions are performing well than approx versions. Refined Approx VC-1 is performing the best. I find that the vertex cover calculated by Approx VC-1 is very close to the optimal solution in most of the observed cases here. And as refined version is refining the output of appox version, its giving even more close result to CNF SAT. The Approx VC-2 is taking less time, but performing worst in terms of approximation ratio. This is because its deleting edges, without considering any other issue. So it has more chances to calculate bigger vertex cover than Approx VC-1. Refined version still doing well here than approx version.

We can see a spike in approximation ratio curve of Approx VC-2. It is possible that in this case, the graphs were very connected, but as this approach does not check for maximum connection, it deletes edges one by one, so it outputs almost two times then the optimal solution.

3 Conclusion

We know CNF-SAT always gives us the optimal result, the minimum vertex cover. But its not efficient for graphs with large numbe of vertices. On the other hand, result of approximation algorithms are not guranteed optimal, but they are more efficient in terms of time and space complexity.