Department of Electrical & Computer Engineering

Methods and Tools for Software Engineering
ECE-650
PROJECT REPORT

Prof. Alireza Sharifi

NAME: Hamza Ali Qureshi
STUDENT ID: 20755087

NAME: Saud Aslam
STUDENT ID:20751814

SUBMISSION DATE: 9-DECEMBER-2018

Introduction:

Problem: Vertex Cover

In a graph with V vertices and (u,v) edges, a vertex cover is a subset of vertices for which from all the edges (u,v), either ‘u’ or ‘v’ in the vertex cover.

Vertex cover is an NP problem. It finds out those vertices in the list of all vertices that covers all the edges i.e: in every edge (u,v) in the graph, either u or v is in the vertex cover list. Specifically in this problem we have to find out the minimum sized vertex cover or in other words the least number of vertices that are incident to all the edges in the graph.

For this we have used three different algorithms to find out the minimum sized vertex covers. Following are the algorithms used:

1. CNF-SAT-VC solver
2. Approx VC-1
3. Approx VC-2

CNF-SAT-VC solver:

It has two steps i.e: Reduction to CNF and then SAT solver. CNF is the conjunction of disjunction of literal, which means that clauses are AND with each other and literals in those clauses are OR with each other. So first the vertex cover problem is reduced to CNF problem and then it is passed to SAT solver which checks for its satisfiability for particular assignments. Because of the two step algorithm CNF is slower as compared to other two algorithms.

Approx VC-1:

This algorithm starts by picking up the highest degree vertex i.e: the vertex with the highest number of incident edges and moves it to the an arbitrary set of vertices. Then it starts removing the edges that are incident to that vertex and this process is repeated until all the edges are removed and we are left with vertex cover set.

Approx VC-2:

This algorithm adds all the edges into a an arbitrary set and picks out an edge (u,v) randomly and starts removing those edges from the set which are incident to edge (u,v). This process is repeated until all the edges are covered and a vertex cover set is obtained.

We have run above three algorithms and compared their results on the bases of their running time and approximate ratio in order to find their efficiencies.

Running time:

We have used ‘pthread_ getcpuclockid’ to calculate the thread clock id and then ‘clock_gettime’ to get the clock time.

Among four threads used in the program, three of them use the above three algorithms respectively and each of them calculate the running time for each algorithm at the end.

Approximate Ratio:

Using CNF-SAT-VC results as the optimal vertex cover set, we calculate approximate ratio for each algorithm as shown in the excel file

$$Approximate\ Ratio = \frac{Size\ of\ Vertex\ cover\ of\ Approx\ VC1\ or\ Approx\ VC2\ or\ CNF-SAT-VC}{Size\ of\ Vertex\ cover\ of\ CNF-SAT-VC}$$

Data:

Running Time											Approximate Ratio								
Vertices	S. no	CNF-MINISAT solver	Average	Standard de	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev	CNF-MINISAT solver	Average	Standard deviatio	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev
5	1	0.000400714	0.0011394	0.00046	1.38E-05	5.85E-05	1.63695E-05	1.00E-05	4.65E-05	1.43427E-05	1.00	1.00	0.00	1.00	1.00	0.00	1.33	1.40	0.34
	2	0.000806131			7.62E-05			4.42E-05			1.00			1.00			1.00		
	3	0.00161374			6.17E-05			4.65E-05			1.00			1.00			1.33		
	4	0.00084305			6.08E-05			5.86E-05			1.00			1.00			2.00		
	5	0.00159814			6.38E-05			4.67E-05			1.00			1.00			1.33		
	6	0.00153318			6.21E-05			4.64E-05			1.00			1.00			1.33		
	7	0.000758953			6.07E-05			5.76E-05			1.00			1.00			1.00		
	8	0.00141226			6.22E-05			4.64E-05			1.00			1.00			1.33		
	9	0.000806495			6.08E-05			6.21E-05			1.00			1.00			2.00		
	10	0.00162124			6.31E-05			4.64E-05			1.00			1.00			1.33		
Vertices	S. no	CNF-MINISAT solver	Average	Standard de	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev	CNF-MINISAT solver	Average	Standard deviatio	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev
10	1	0.00918556	0.0087922	0.00145	2.97E-05	3.32E-05	1.15979E-05	1.07E-05	2.09E-05	2.42753E-05	1.00	1.00	0.00	1.00	1.02	0.06	1.20	1.48	0.27
	2	0.00976539			4.48E-05			2.40E-05			1.00			1.00			1.20		
	3	0.00912608			3.11E-05			1.04E-05			1.00			1.00			1.60		
	4	0.0089648			4.31E-05			8.82E-05			1.00			1.00			1.20		
	5	0.00946301			2.97E-05			1.07E-05			1.00			1.00			1.60		
	6	0.00922286			2.95E-05			1.05E-05			1.00			1.00			1.60		
	7	0.0100347			4.86E-05			1.05E-05			1.00			1.20			1.20		
	8	0.00511051			1.55E-05			1.01E-05			1.00			1.00			2.00		
	9	0.00757829			1.65E-05			1.06E-05			1.00			1.00			1.60		
	10	0.00947104			4.36E-05			2.36E-05			1.00			1.00			1.60		
Vertices	S. no	CNF-MINISAT solver	Average	Standard de	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev	CNF-MINISAT solver	Average	Standard deviatio	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev
15	1	0.81789	4.61E+00	5.92365	4.70E-05	4.81E-05	1.36719E-05	1.12E-05	2.13E-05	1.76401E-05	1.00	1.00	0.00	1.00	1.01	0.05	1.71	1.62	0.18
	2	0.768302			4.61E-05			1.14E-05			1.00			1.00			2.00		
	3	0.0726933			4.59E-05			1.15E-05			1.00			1.00			1.67		
	4	1.18417			8.42E-05			5.72E-05			1.00			1.00			1.43		
	5	11.44			4.73E-05			1.19E-05			1.00			1.00			1.50		
	6	2.04874			4.76E-05			1.15E-05			1.00			1.00			1.71		
	7	1.01768			4.70E-05			1.14E-05			1.00			1.14			1.71		
	8	0.950467			4.70E-05			1.11E-05			1.00			1.00			1.43		
	9	12.7763			3.41E-05			4.97E-05			1.00			1.00			1.50		
	10	15.0095			3.51E-05			2.61E-05			1.00			1.00			1.50		
Vertices	S. no	CNF-MINISAT solver	Average	Standard de	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev	CNF-MINISAT solver	Average	Standard deviatio	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev
17	1	17.0452	1.60E+01	8.45986	3.71E-05	4.13E-05	1.74333E-05	2.55E-05	2.16E-05	1.23279E-05	1.00	1.00	0.00	1.00	1.04	0.06	1.75	1.54	0.10
	2	9.10952			3.71E-05			1.22E-05			1.00			1.13			1.50		
	3	34.9406			2.39E-05			2.50E-05			1.00			1.13			1.50		
	4	17.1767			3.72E-05			1.23E-05			1.00			1.00			1.50		
	5	16.1174			5.06E-05			2.99E-05			1.00			1.00			1.50		
	6	19.5626			3.55E-05			1.19E-05			1.00			1.00			1.50		
	7	15.7623			3.48E-05			2.68E-05			1.00			1.00			1.71		
	8	1.5709			8.75E-05			4.96E-05			1.00			1.00			1.43		
	9	12.2038			3.47E-05			1.18E-05			1.00			1.00			1.50		
	10	16.2373			3.51E-05			1.16E-05			1.00			1.13			1.50		
Vertices	S. no	CNF-MINISAT solver	Average	Standard dev	Approx-1-VC solver	Average	Standard dev	APPROX-VC-2 solver	Average	Standard dev									
20	1	1462.6	1.46E+03		4.42E-05	4.42E-05		1.40E-05											

Special Case: 20 Vertice:

V 20

E{<14,12>,<16,3>,<14,1>,<9,13>,<3,13>,<4,11>,<1,11>,<17,2>,<10,8>,<0,14>,<17,12>,<0,19>,<0,10>,<0,3>,<3,5>,<18,0>,<7,0>,<7,14>,<1,0>,<7,13>,<8,1>,<7,10>,<0,5>,<14,10>,<3,9>,<15,10>,<0,6>,<15,7>,<15,2>,<1,9>}

CNF-SAT-VC: 0,1,2,3,4,7,10,12,13

Time: MINISAT solver: 1462.6

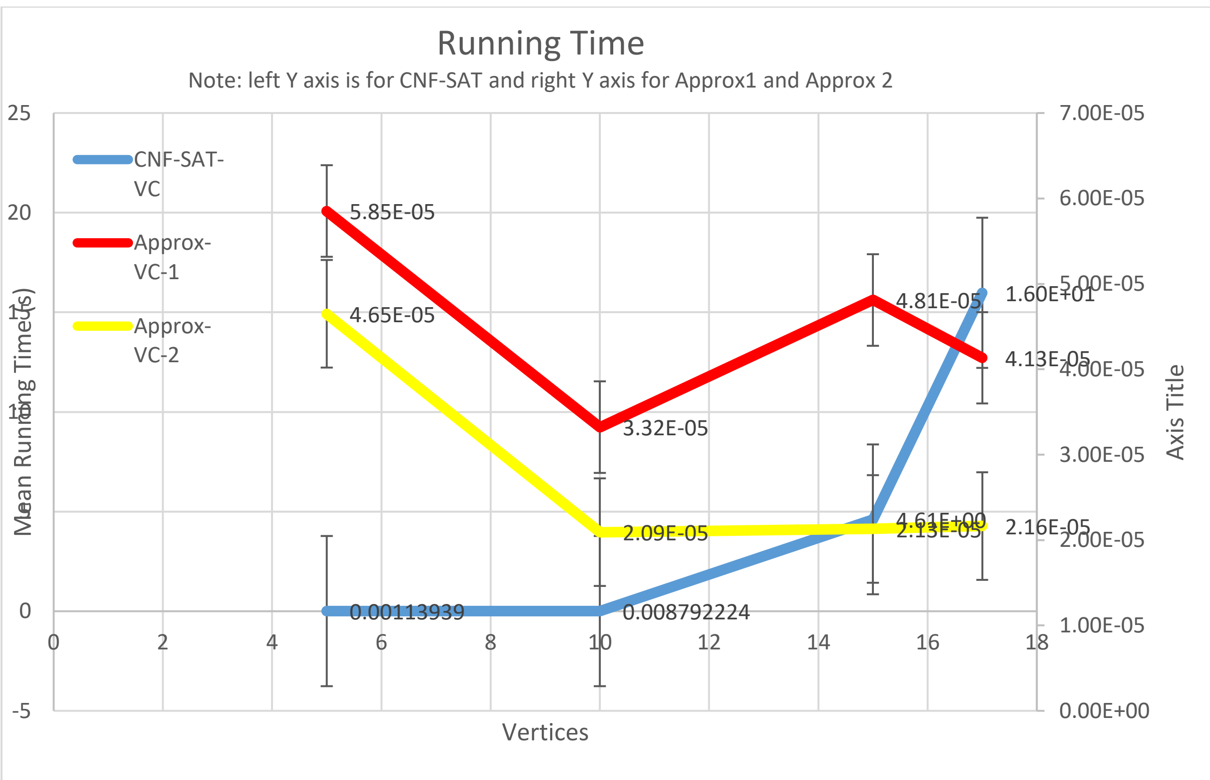
APPROX-VC-1: 0,1,2,3,4,7,9,10,12

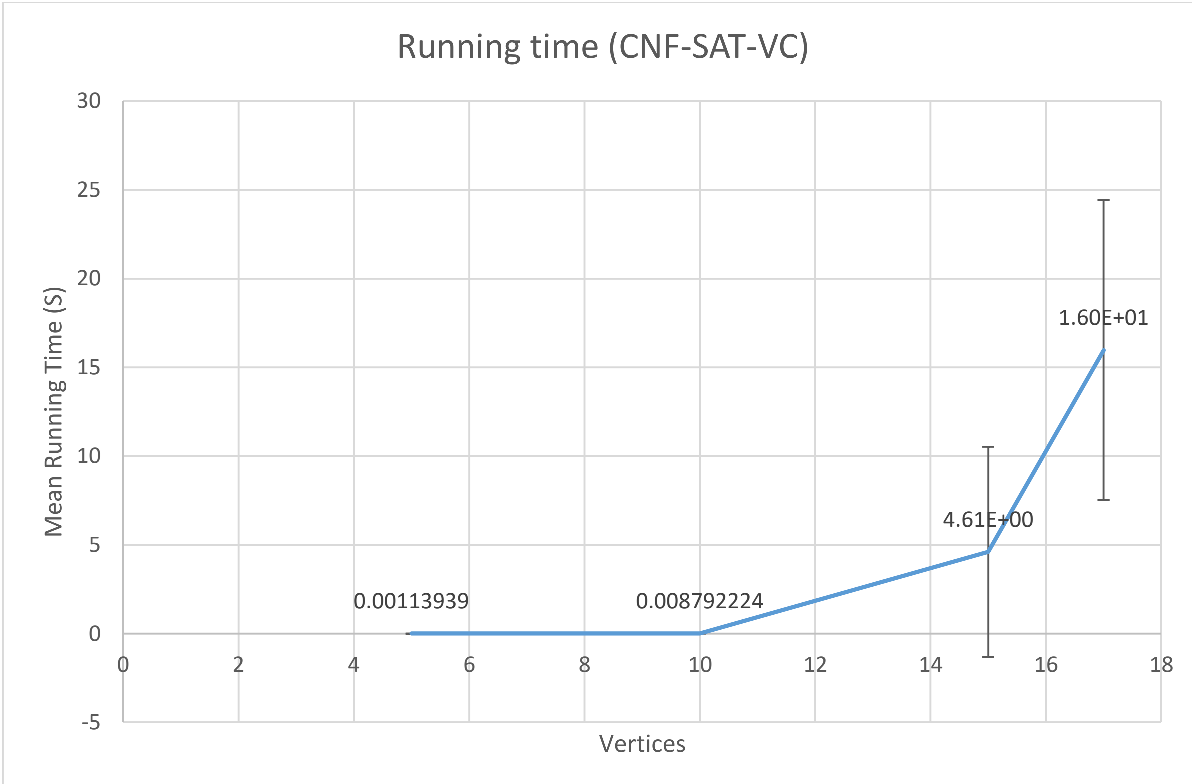
Time: Approx-1-VC solver: 4.4218e-05

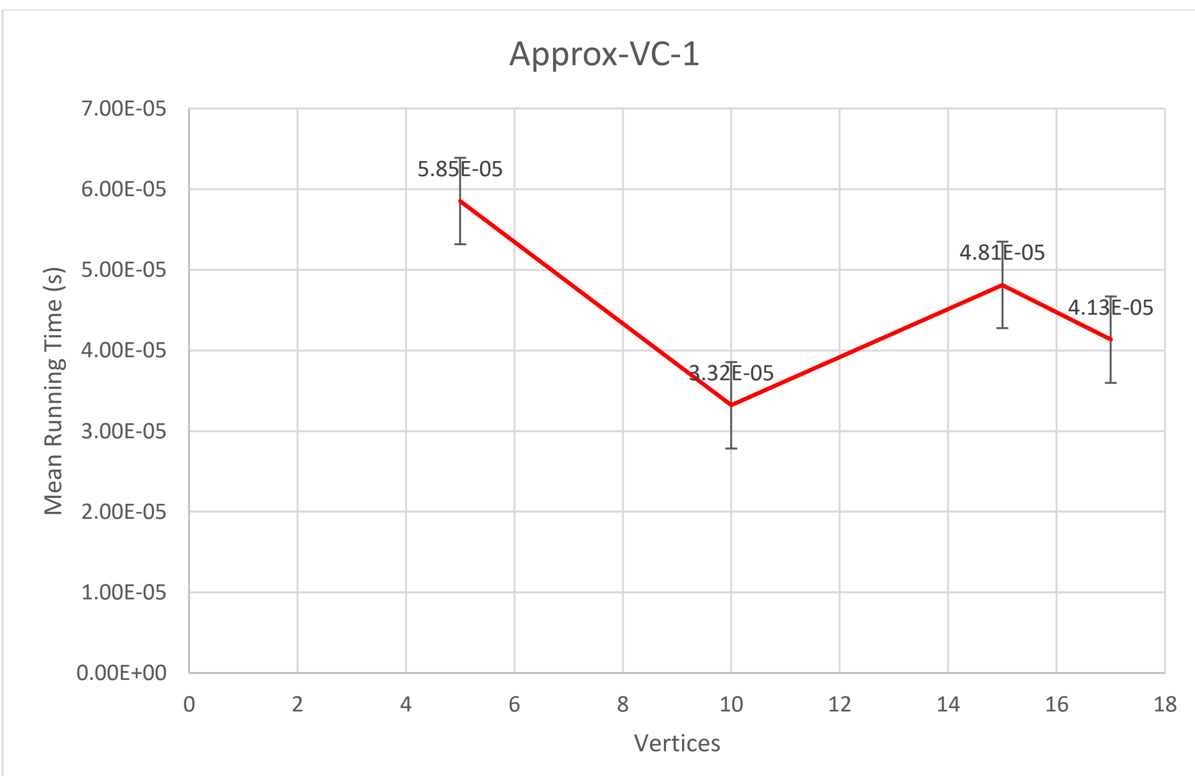
APPROX-VC-2: 0,1,2,3,7,8,10,11,13,14,16,17

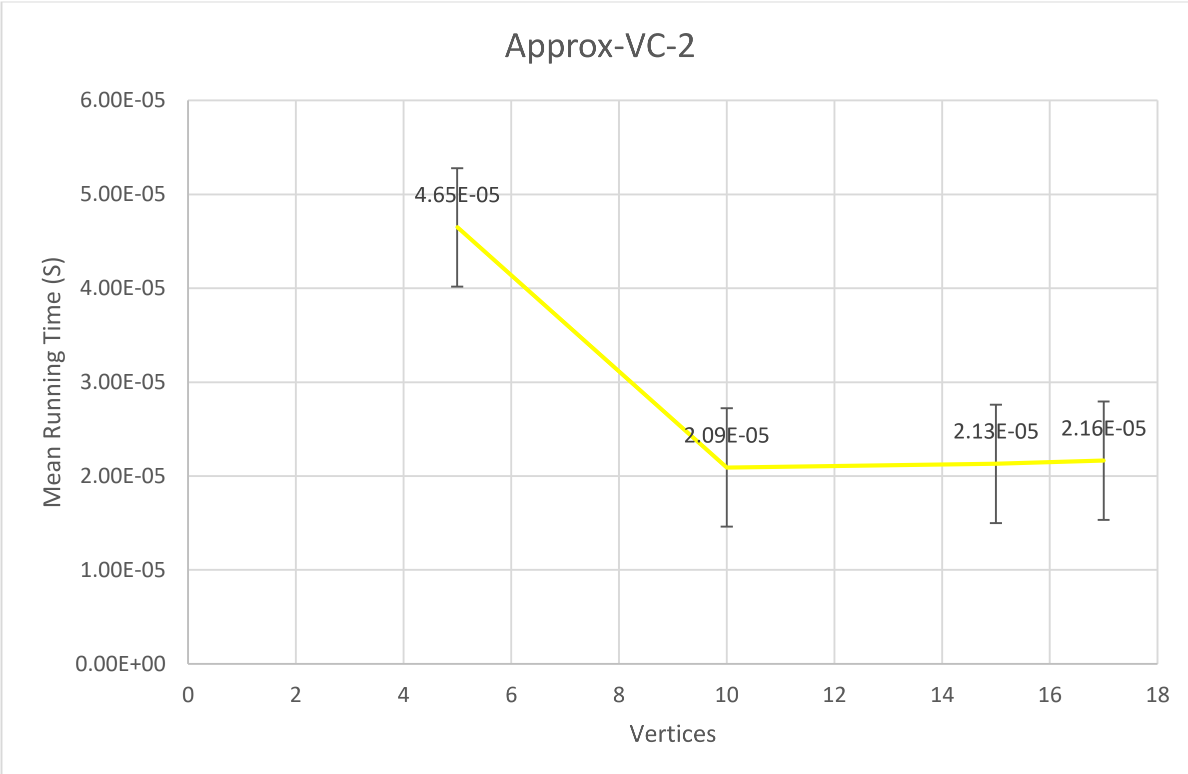
Time: APPROX-VC-2 solver: 1.4024e-05

Graphs:

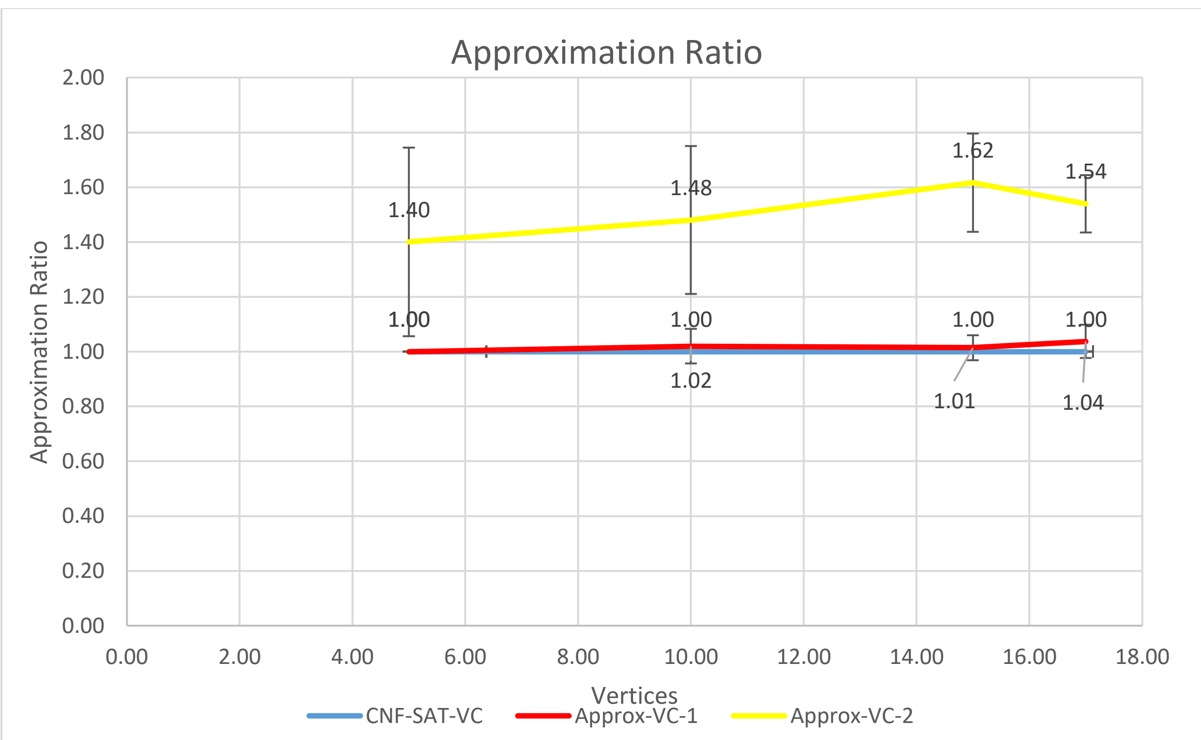



Running time (CNF-SAT-VC)
Mean Running Time (S)
Vertices
0.00113939
0.008792224
4.61E+00
1.60E+01
30
25
20
15
10
5
0
-5
0
2
4
6
8
10
12
14
16
18


Approx-VC-1
Mean Running Time (s)
Vertices
5.85E-05
3.32E-05
4.81E-05
4.13E-05
7.00E-05
6.00E-05
5.00E-05
4.00E-05
3.00E-05
2.00E-05
1.00E-05
0.00E+00
0
2
4
6
8
10
12
14
16
18



Approx-VC-2
6.00E-05
5.00E-05
4.00E-05
3.00E-05
2.00E-05
1.00E-05
0.00E+00
Mean Running Time (S)
4.65E-05
2.09E-05
2.13E-05
2.16E-05
0
2
4
6
8
10
12
14
16
18
Vertices



Approximation Ratio
2.00
1.80
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
Approximation Ratio
1.40
1.48
1.62
1.54
1.00
1.00
1.00
1.00
1.02
1.01
1.04
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
18.00
Vertices
CNF-SAT-VC
Approx-VC-1
Approx-VC-2

Analysis:

CNF-SAT: The running time of this algorithm in its worst time is:

$$\boldsymbol{\theta}(2^{|V|})$$

|V|: Total number of vertex in the graph

The *boolean satisfiability problem* (SAT) involves finding a satisfying truth assignment for a set of clauses *C* over the boolean variables $V=\{v_1,v_2,...,v_n\}$ so that each clause in *C* contains at least one true literal. Since *V* contains *n* variables and each of these variables can only have 2 different values (i.e., true or false), the total number of possibilities to be tested is 2^n

MiniSat on the other hand works on Davis–Logemann–Loveland algorithm (DPLL) making the running time for this this algorithm less than 2^n .

Increasing the number of vertexes (increasing number of literals) do result in increasing the running time for CNF-SAT algorithm. Furthermore, increasing the number of *k* increases the number of clauses in the CNF formula. As the value of *k* approaches to the true instance which satisfy the CNF formula, we can expect the sat-solver to run for a long time before giving out the SAT assignment for the formula. For example for the case when the vertexes are 20, the sat-solver is taking a very long(1420 unit time) to solve. This could be because choice of *branching literal*, which is the literal considered in the backtracking step. There exists some instances for which running time can shoot up on the heuristic choice of branching literals

APPROX-VC-1: The running time of this algorithm in its worst time is:

$$\boldsymbol{\theta}(|V|^2+|E|)$$

|V|: Total number of vertex in the graph

|E|: Total number of edges in the graph

This algorithm scan all the nodes of the graph and the finds the node with the maximum degree. The process is repeated for the nodes until no node is left with any edge (i.e the graph has empty edges set).

Also it visits and remove all edges incident of the node being picked. In the worst case, it will scan |E| edges.

The approximate ratio is considered to be close to 1 because every time we visit a node we seek for the node which have highest degree among all the nodes.

The running time of this algorithm is higher than Approx-VC-2, because of the need to visit all nodes at least once.

Although we expect the running time of this algorithm to increase with increase in number of nodes, but we see that when the input to the graph is 5 vertexes, there is a spike in the mean running time. This is because at lower values of |V|, the time for the entire one invocation of the code is not spread out (average out) completely. That’s the reason we often ignore running time of algorithms for lower values of N.

APPROX-VC-2: The running time of this algorithm in its worst time is:

$$\boldsymbol{\theta}(|V|+|E|)$$

|V|: Total number of vertex in the graph

|E|: Total number of edges in the graph

The algorithm picks an arbitrarily edge and put both of the node in the vertex cover list. In the worst case, suppose that there exists a family of graph:

$$V = \{u_0, u_1, \ldots., u_{2n-1}\}, where\ n\ is\ a\ positive\ integer, E = \{\langle u_{2i}, u_{2i+1}\rangle \mid i \in [0, n-1]\}$$

If we run this algorithm on this family of graph, we will get the vertex cover which would be twice the minimum vertex cover. The reason is that the, in each iteration this algorithm picks two nodes and in the worst case, it will keep on picking two nodes making the vertex cover list twice that of optimal solution. That's the reason why the approximation ratio value of this algorithm is usually higher and close to 2.0.

Although we expect the running time of this algorithm to increase with increase in number of nodes, but we see that when the input to the graph is 5 vertexes, there is a spike in the mean running time. This is because at lower values of |V|, the time for the entire one invocation of the code is not spread out (average out) completely. That's the reason we often ignore running time of algorithms for lower values of N.

The running time, however, is better than Approx-VC-1 because this algorithm removes an extra node in each iteration and mark both nodes as visited which means that it will never visit that node again. So the number of nodes are at most reduce by 1 in each iteration.

Finally, note that comparing algorithms by their complexity classes is useful only for large N (in this case, V). For example it is difficult to state, whether an **O(V)** algorithm or an **O(V^2)** algorithm is faster for small N. To gauge the running time in it true sense, we might need larger values of Vertexes e.g 1000.