Checking Arc-Consistency of randomly generated graphs using four Arc-Consistency algorithms (AC-1, AC-2, AC-3, AC-4)

Mirajul Mohin, Roll: FH-28

February 27, 2019

Problem Description

I have to select randomly generated graphs with random edges and select constraints for each edge from a constraint list, then run the Arc-Consistency algorithms (AC-1,AC-2, AC-3, AC-4) for increasing number of nodes and then measure the performances of each algorithm and plot a graph showing the comparison of four Arc-Consistency algorithms.

Experiment Setup

All the AC algorithms are run in graphs that are randomly generated with the following properties:
- Number of nodes/variables are in range from 10 to 300 and increased by 10.
- Domain of each variable are in size 5 to 35 and each domain can have integers value ranging from 1 to 100.
- Constraints that at are used are as follows:

1. X > Y
2. X < Y
3. X != Y
4. X = Y^2

- If each value of domain of $node_1$<15 and each value of domain of $node_2$>15 then $node_1$ = $node_2^2$ is used.
- If each value of domain of $node_1$ < 80 and each value of domain of $node_2$ > 10 then $node_1$!= $node_2$ is used.

Performance Graph

This section will include two graphs for four AC algorithms :

- Execution time v/s number of nodes.
- Domain reduction v/s number of nodes.

# Of Node	AC-1	AC-2	AC-3	AC-4
10	0.0042617321	0.0033900738	0.0100092888	0.0132653713
20	0.0102217197	0.0079462528	0.0249300003	0.0349652767
30	0.0166249275	0.0104160309	0.0497317314	0.0412845612
40	0.0170381069	0.0206143856	0.0323498249	0.0485782623
50	0.0383327007	0.0212697983	0.0556256771	0.0662755966
60	0.0272202492	0.0116801262	0.0762724876	0.0920517445
70	0.0629811287	0.0348703861	0.1058433056	0.1113991737
80	0.0249903202	0.0243706703	0.0414726734	0.1095626354
90	0.039103508	0.0593919754	0.0504603386	0.147995472
100	0.0517010689	0.0524275303	0.1027541161	0.1549050808
110	0.0638771057	0.060934782	0.1585431099	0.1474618912
120	0.0750837326	0.0355117321	0.1911063194	0.2123129368
130	0.0939488411	0.0601854324	0.1780354977	0.1684241295
140	0.0432281494	0.0303883553	0.0556745529	0.1829538345
150	0.06523633	0.068202734	0.1171014309	0.2356557846
160	0.0762181282	0.0595431328	0.2139544487	0.2439303398
170	0.1230254173	0.1076412201	0.2802698612	0.2771680355
180	0.0675816536	0.0738112926	0.2287325859	0.2725143433
190	0.0708918571	0.0781106949	0.1963319778	0.2826082706
200	0.1130554676	0.0808088779	0.2954003811	0.3060064316
210	0.0688388348	0.0664157867	0.1404042244	0.3185791969
220	0.0882306099	0.0825419426	0.136582613	0.3451268673
230	0.0856804848	0.1487529278	0.1996562481	0.3681504726
240	0.0934123993	0.1062574387	0.2286353111	0.3589828014
250	0.1359250546	0.0933246613	0.3449332714	0.4275202751
260	0.1359128952	0.1841871738	0.3920555115	0.436727047
270	0.0905373096	0.1114931107	0.1703228951	0.3922309875
280	0.0968031883	0.1078774929	0.1644122601	0.4330723286
290	0.128210783	0.1782431602	0.2673316002	0.4767613411
300	0.1260268688	0.0963172913	0.3451032639	0.4292337894

Figure 1: Execution time of four AC algorithms for increasing number of node.

The above figure shows the execution time of all arc consistency algorithms for increasing number of nodes (10, 20,..,300).

Figure 2 shows the comparison of four execution times of four AC algorithms. Here we see, almost linear increase of AC-1, AC-2 and AC-4. But AC-3 doesn't hold that property.

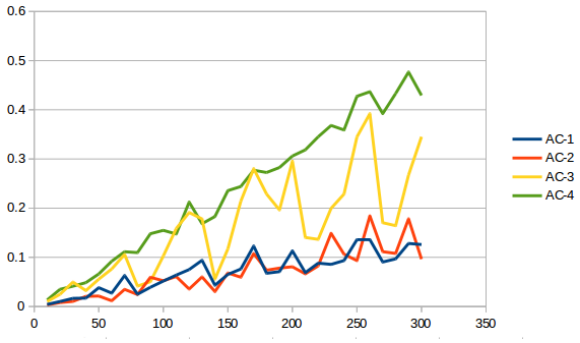

Figure 2: Comparison graph of execution time of ac algorithms for data of Figure 1.

Figure 3 shows the comparison graph of domain reduction of four AC algorithm v/s number of nodes. AC-1, AC-2 and AC-4 reduces the same number of domain where AC-3 produces result without reducing that large number of domain.

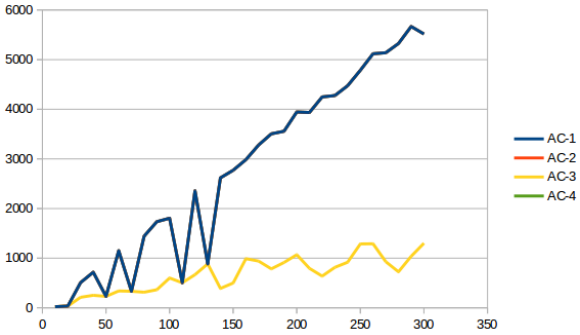

Figure 3: Comparison graph of domain reduction of ac algorithms.

Discussion on Fig-2,Fig-3:

From Figure 2, we see that AC-3's graph is changing its slope frequently unlike the other three. This is because AC-3 returns false as long as it finds any domain is zero. But the other three don't return immediately after finding any domain to be zero. Hence, we can guess AC-3 will take less time than the others. But my implementation of AC-3 and AC-4 (iteration through the constraint list) is not so well comparing with the others, thus these two are taking a little longer time comparing with the other two.

Now from Figure 3, we see AC-1, AC-2 and AC-4 are reducing exactly same number of domains, where AC-3 returns true/false immediately after finding any domain is zero. So, AC-3 reduces less number of domain comparing with other three.

ANOVA Test Result

treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Tukey HSD inferfence
A vs B	0.1129	0.8999947	insignificant
A vs C	5.3593	0.0013598	** p<0.01
A vs D	9.8553	0.0010053	** p<0.01
B vs C	5.4722	0.0010260	** p<0.01
B vs D	9.9681	0.0010053	** p<0.01
C vs D	4.4959	0.0100958	* p<0.05

Figure 4: Tukey HSD result.

Here A,B,C and D respectively denote AC-1, AC-2, AC-3 and AC-4.

Conclusion

There is two python file (generateGraph_028.py and Solution_028.py). generateGraph_028.py generates random graph having node number 10, 20, 30,..., 300 and edge number = (node number) × 1.5, assigns domain and constraint to each variable and arc, following the problem description and write it in Test.txt file. Solution_028.py reads Test.py and runs all AC algorithms and produces file_exeTime.txt and file_domain.txt. file_exeTime.txt contains all execution time of four algorithm and file_domain.txt contains number of domain reduction for four algorithms.