Amit Sarker, Roll: 99

Let $N = (V, D, C)$ be a constraint network. The constraint graph $G = (V, E)$ consists of V vertices which is the variables of our constraint network and E edges. There are some constraints $c \in C$ enforced on these edges. I have used *Arc consistency* algorithms for making the constraint network arc consistant.

1 *Design of the solution*

I have generated a random graph *(graph = nx.erdos_renyi_graph(N, P))* using *networkx* library, where *N = number of nodes*, *P = probability*, which means *each edge is included in the graph with probability P independent from every other edge*. From the graph, I have generated the variables, domains and constraints list. *Domains* are randomly selected with a fixed length domain size and I have used 10 *constraints* which are assigned randomly to every node as I described my earlier problem description assignment. I have used this graph for all the arc-consistency algorithms and the comparisons and findings are described in the following sections.

2 *Comparisons*

I have compared all these four algorithms (**AC-1**, **AC-2**, **AC-3**, and **AC-4**) based on ***total number of nodes vs run time*** and ***probability of edges vs run time***. For both cases, I have changed the domain size from *10 to 100*, increased by *30*. I have calculated the average run time for each domain size. I have taken an average of *200 iterations* for both cases to calculate the average run time.

- ***Total number of nodes vs Run time:***

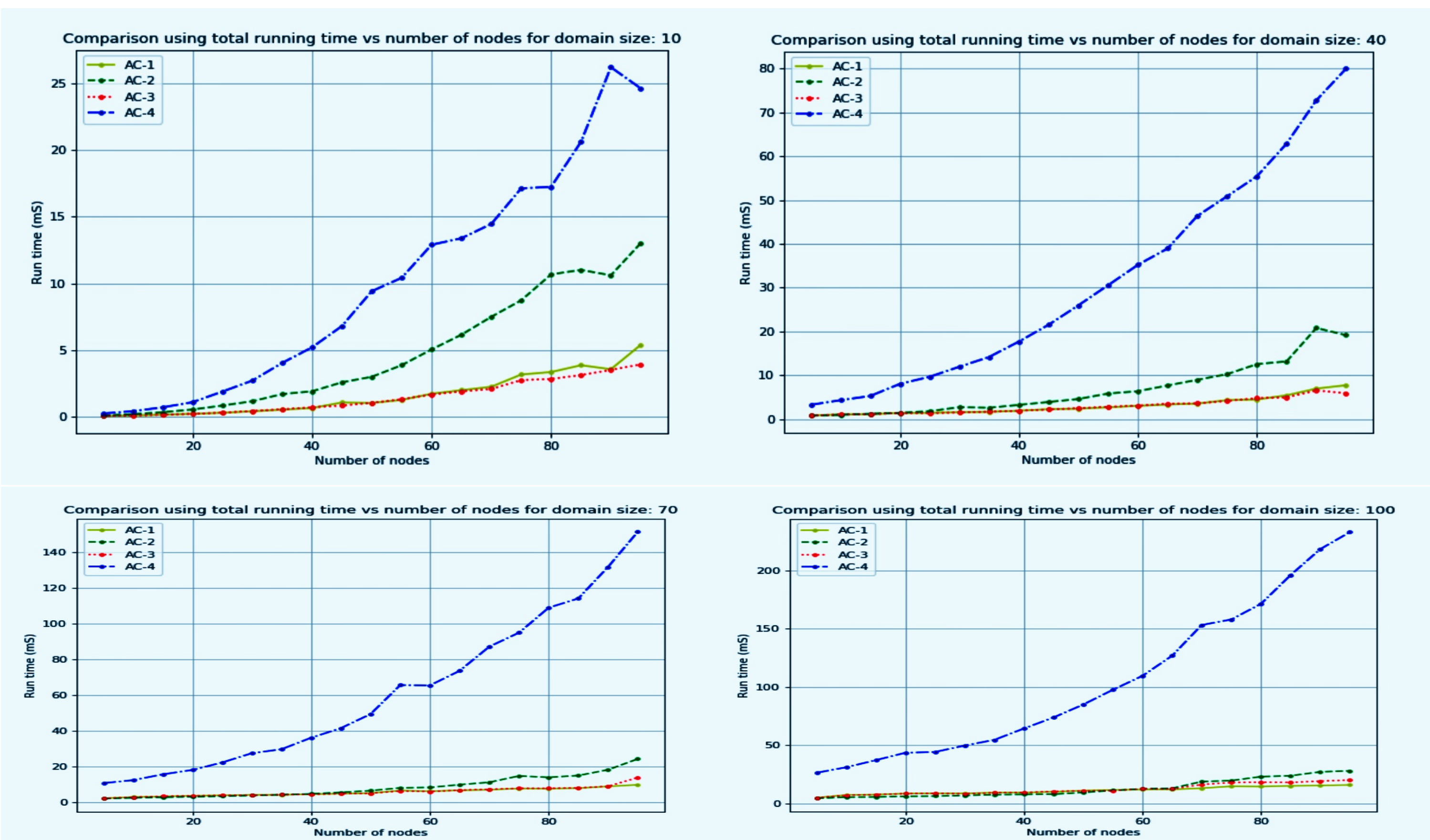

Figure 1: **Total number of nodes vs Run-time for domain size 10, 40, 70, 100**

- ***Probability of edges vs Run time:***

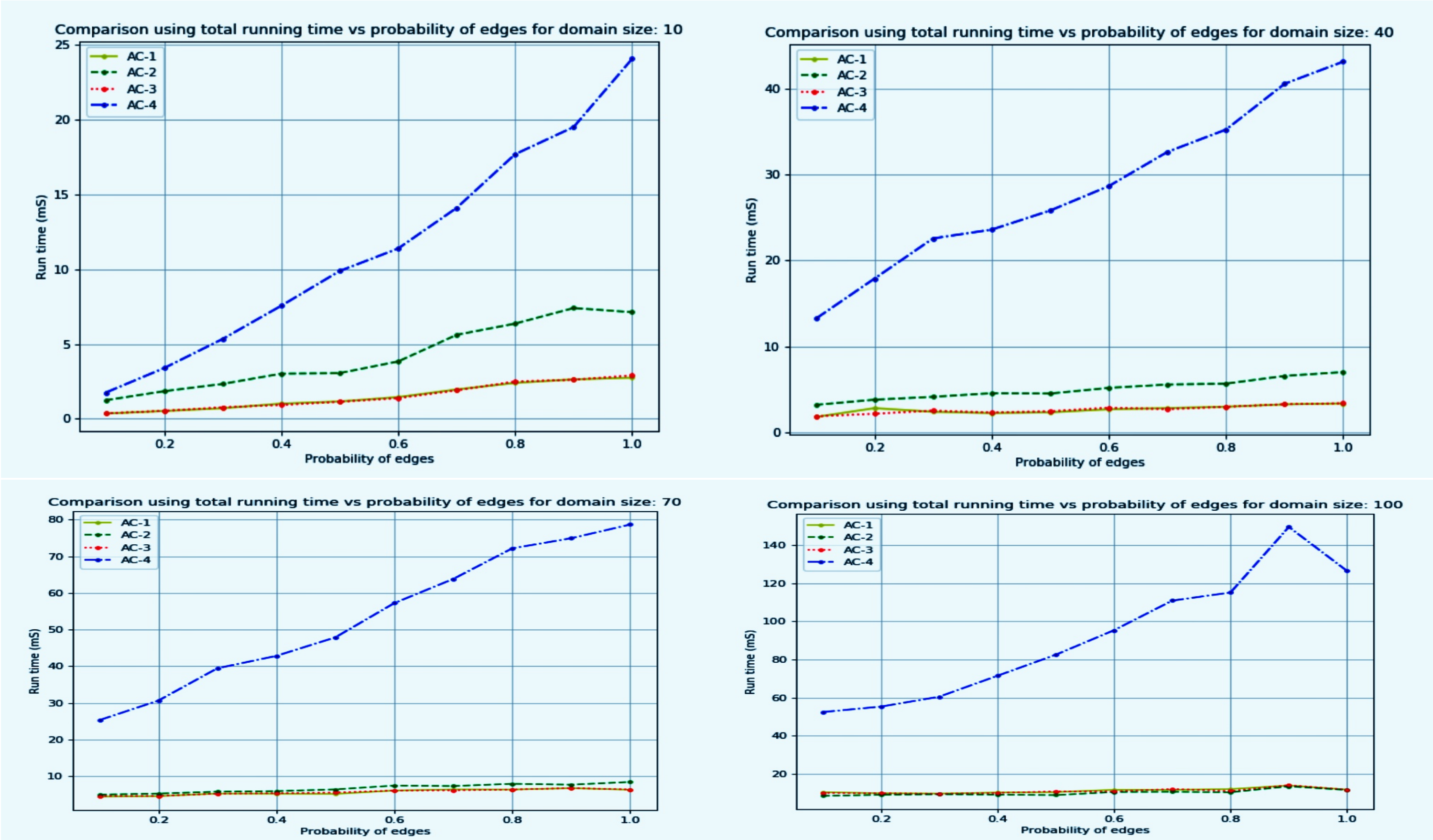

Figure 2: **Probability of edges vs Run-time for domain size 10, 40, 70, 100**

For comparing the *total number of nodes vs run time*, I have changed the value of the total number of nodes from *5 to 95*, increased by *10*. I have fixed the probability of edge to *0.5* for this case. Figure 1 shows the comparison for *total number of nodes vs run time*. For the second case, *probability of edges vs run time*, I have changed the value of the probability of edge from *0.1 to 1.0*, increased by *0.1*. I have fixed the number of total nodes to *50* for this case. Figure 2 shows the comparison for *probability of edges vs run time*.

3 *Findings*

From the relative comparisons, it is clear that the performance of these four algorithms depends highly on *domain size* as well as *constraints selected*, *number of nodes*, *edges*, and other factors. From Figure 1, we can say that, run time of *AC-1* and *AC-3* are close to one another. These two perform well in every case. This is because in the *AC-3*, if the domain of a node has changed then I have checked the domain of each nodes adjacent with that particular node. In *AC-1* algorithm when the domain of a node has changed I have checked every nodes for revise. But I have returned false if the domain of a single node becomes empty for all the four algorithms. For this reason, time complexity becomes $O(e.k^3)$ for both cases and the run time of *AC-1* and *AC-3* are close in every case. For larger domain size, run time of *AC-2* are also close to *AC-1* and *AC-3*. In *AC-2* algorithm, we have made the graph consistent in an iterative increment process. First of all, we have made the graph consisting of less or equal j $(j < total\_nodes)$ nodes consistent. Then we have tried to make the graph consisting of less or equal $j+1$, $j+2$, ..., ..., i $(i = total\_nodes)$ nodes consistent. So, if domain size increases, time to perform a revise increases, but number of nodes are same. For this reason, *AC-2* performs well in larger domain size. But *AC-4* performs worst for all the cases. Time complexity of finding all the supports $S[v_j, a_j]$ and counter(v_i, a_i, v_j) is $O(n.k)$ and $O(e.k)$ respectively, where n = *number of nodes*, e = *number of edges*, k =

maximun length of a domain. For these reasons, if the graph is already arc consistent or close to it, the best case time for *AC-1* and *AC-3* becomes $O(e.k)$. But for *AC-4* it is still $O(e.k^2)$. That is why, *AC-1* and *AC-3* are outperforming *AC-4* although *AC-4* has the best run time in the worst case scenario. In Figure 2, conditions are very similar to that of Figure 1. Running time of *AC-4* increases dramatically and *AC-2* performs well for larger domains.

4 *Statistical significance Test*

I have performed the one-way *ANOVA Tukey HSD Test* to determine whether there are any statistically significant differences between the mean running times of four arc consistency algorithms. Figure 3 shows that, all the possible pairs of arc consistency algorithm, the mean running times are significanty different except *AC-1* vs *AC-3*. This is quite expected, as we have seen from Figure 1 and Figure 2 that run time of *AC-1* and *AC-3* are very close to each other.

Tukey HSD results

treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Tukey HSD inferfence
A vs B	6.2528	0.0010053	** p<0.01
A vs C	0.2527	0.8999947	insignificant
A vs D	19.9457	0.0010053	** p<0.01
B vs C	6.5054	0.0010053	** p<0.01
B vs D	13.6930	0.0010053	** p<0.01
C vs D	20.1984	0.0010053	** p<0.01

Scheffé results

treatments pair	Scheffé T-statistic	Scheffé p-value	Scheffé inferfence
A vs B	4.4214	0.0002331	** p<0.01
A vs C	0.1787	0.9984966	insignificant
A vs D	14.1038	1.1102e-16	** p<0.01
B vs C	4.6000	0.0001099	** p<0.01
B vs D	9.6824	1.1102e-16	** p<0.01
C vs D	14.2824	1.1102e-16	** p<0.01

Figure 3: **One-way ANOVA with post-hoc Tukey HSD Test results**