Amit Sarker, Roll: 99

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 variable, domain and constraint 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 *5 to 85*, increased by *20*. 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. 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. For the second case, *probability of edges vs run time*, I have changed the value of the probability of edge from *0.1 to 0.9*, increased by *0.1*. I have fixed the number of total nodes to *20* for this case. The graphs are shown below:

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

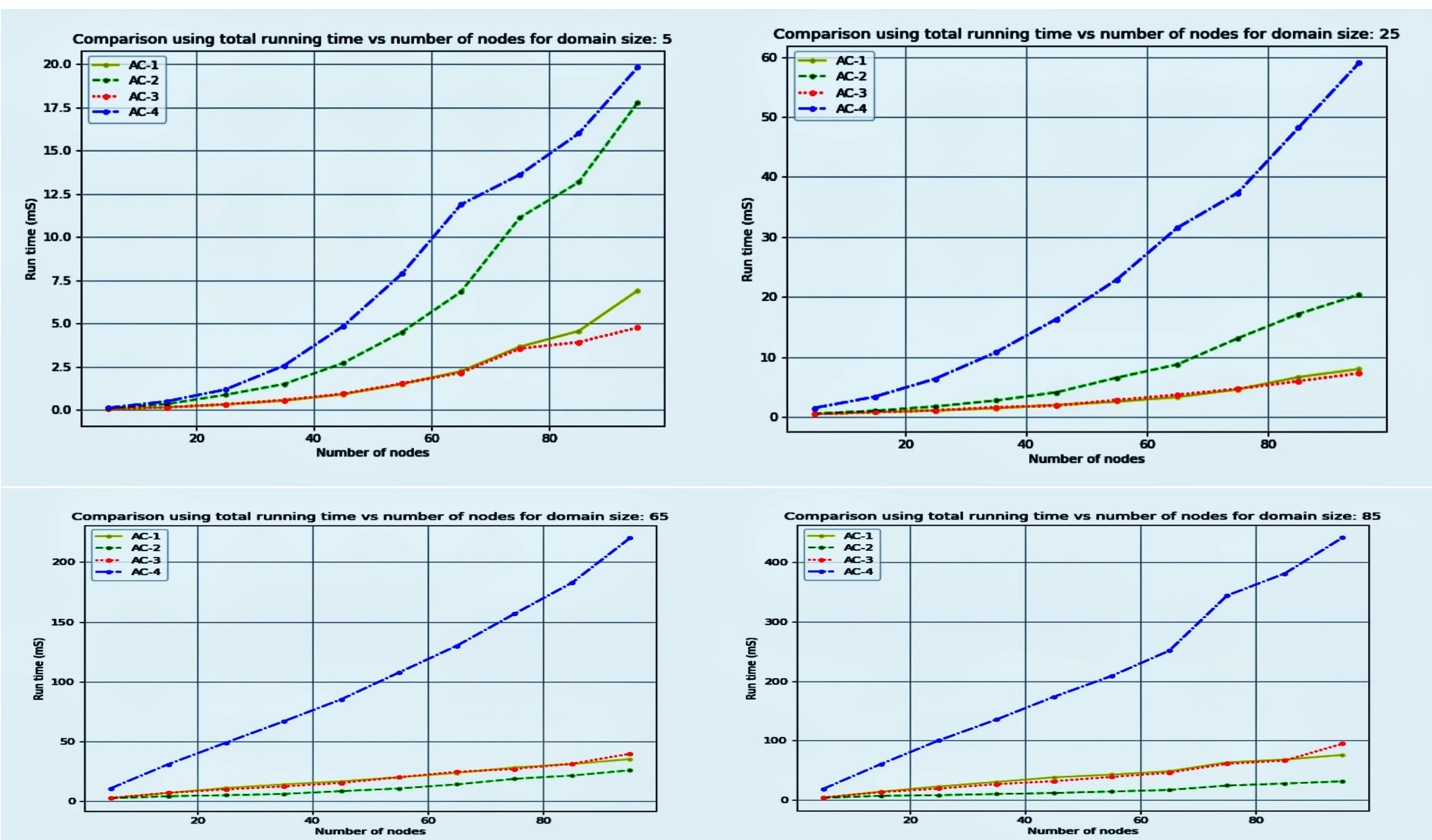

Figure 1: **Total number of nodes vs Run-time for domain size 5, 25, 65, 85**

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

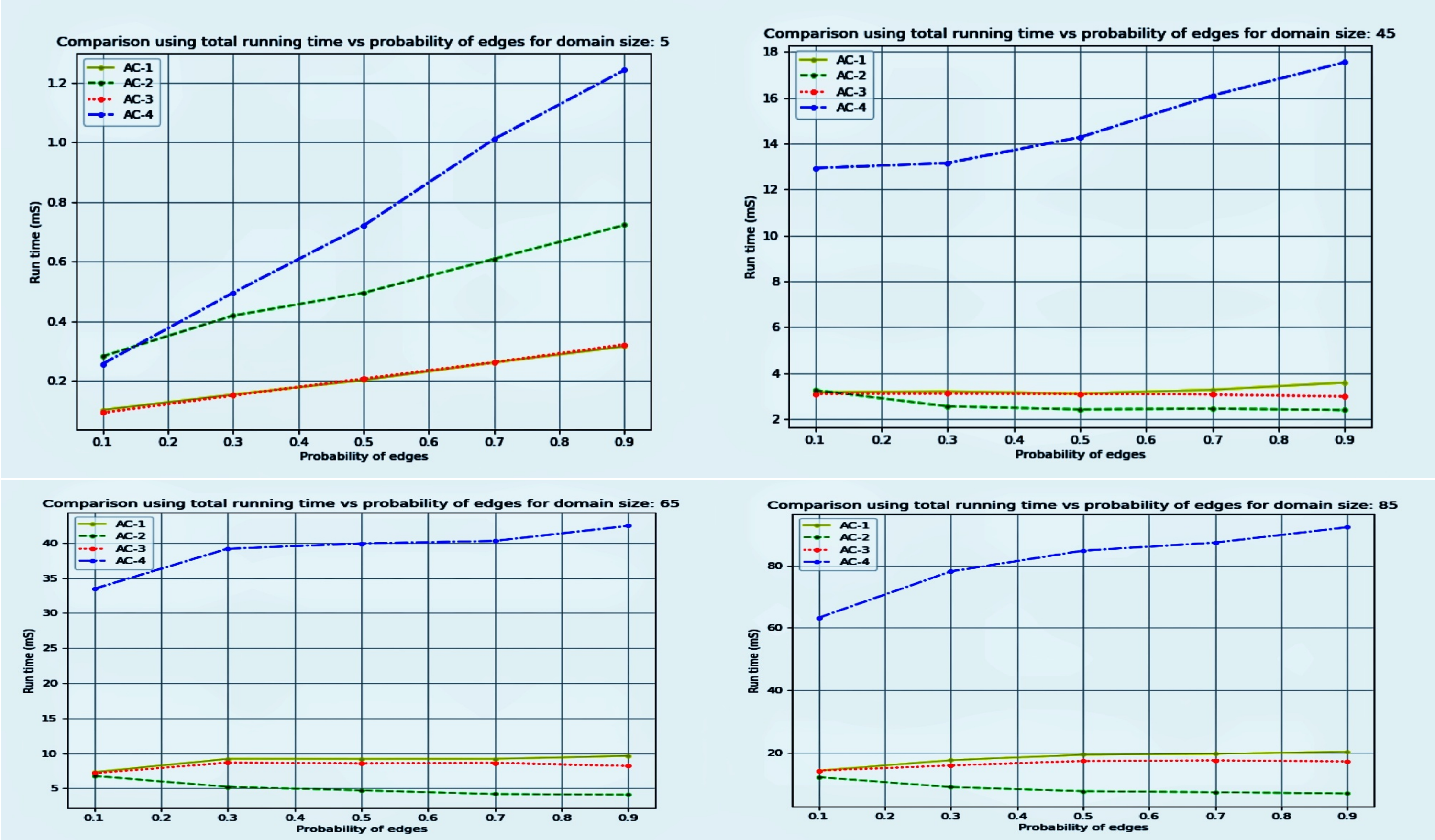

Figure 2: **Probability of edges vs Run-time for domain size 5, 45, 65, 85**

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 conclude that, for small domain size, performances are close to one another. Both **AC-1** and **AC-3** performs well in that case. But for larger domain size, **AC-2** performs best and **AC-4** worst. Running time increases dramatically for **AC-4** when we use larger domain size.

In Figure 2, conditions are very similar to that of Figure 1. Running time of **AC-4** increases dramatically and **AC-2** performs best for larger domains. The performance of **AC-1** and **AC-3** are very similar in this case. From these findings, we can conclude that **AC-2** is the best performer in my case. Although **AC-4** has the best run time *(ek^2)* in the worst case scenario, in the best case, **AC-1** and **AC-3** outperform **AC-4**.