Constraint Satisfaction Problem(CSP)
Comparative Performance Analysis of Arc-Consistency Algorithms
AC-1, AC-2, AC-3, AC4

Amit Roy, Roll: JH-40
Department of Computer Science and Engineering,
University of Dhaka,
Email: aroy7298@gmail.com

I. Implementation Details:

In this assignment, our goal was to implement the arc consistency algorithms **(AC-1, AC-2, AC-3, AC-4)** to reduce the domain of the variables of **Constraint Satisfaction Problem(CSP)** by applying constraint propagation.
According to the previously submitted concise problem definition we will have a set of **variables(eg. 50,100,150,200)** which will have a domain of random size in the range **1~50** where the values of the domain will contain integers from the range **1~1000**. The variables of the CSP are treated as the nodes of a graph.
The graph is generated by a **random matrix** using **numpy**, a python library. The random adjacency matrix should be a **symmetric matrix** (i.e. if there is an edge between **u** $\rightarrow$ **v** there should be an edge between **v** $\rightarrow$ **u**) without any self loop because all the constraints are binary constraints so we don't need edge between a node and itself. Since there is a **constraint** for every edge between any two nodes in the graph so for **n** nodes I have generated a **n * n** adjacency matrix where the each entry of the matrix could be an integer between 0 to 11 where 0 denotes no edge and other nonzero entries denote the constraint between two adjacent nodes. I used the following constrains in my implementation:

Constraints:

$$y \leq x \tag{1}$$

$$y \geq x \tag{2}$$

$$y = x \tag{3}$$

$$y \neq x \tag{4}$$

$$y\%x = 0, \text{ y divides x} \tag{5}$$

$$gcd(x, y) = 1 \text{ i.e. x and y are co-prime} \tag{6}$$

$$y = mx + c, \text{ m and c are constants (m=2 and c = 1)} \tag{7}$$

$$y = ax^2 + bx + c, \text{a,b and c are constant (a=1,b=1,c=1)} \tag{8}$$

For the constraints **1,2,5,7,8** we need to map them to different value for different pair of variables because they don't follow the commutative rule that is if **u**$\leq$**v** then **v**$\geq$**u** . For each algorithm, I have take 20 iterations and take the average to maintain the smoothness of the graph.

II. Comparative Performance Analysis

To compare the performance of the arc consistency algorithms we used three performance metrics of the graph.

(a) ***Running Time (msec) vs Number of nodes***
(b) ***Running Time (msec) vs Percentage of edges***
(c) ***Running Time (msec) vs domain size***

A. Running Time (msec) vs Number of nodes

I have used the Alan Mackworth's **Consistency in Network Relations** [1] as a source of the **AC1**, **AC2**, **AC3** algorithms and [2] for implementing **AC4**. While comparing the algorithms we have noticed that 1 **AC3** performs better than **AC1**. These two algorithms are almost same except the fact that **AC1** checks consistency of all the arcs even if only one edge is revised. But **AC3** checks the consistency only the adjacent edges of the first node of the revised edge.So, it's a reason behind **AC3** performing better another reason is when the domain of any node is empty **AC3** terminates.**AC2** also performs better than **AC1** but it's performance is almost same **AC3** in most of the cases. While **AC3** almost always better than **AC4** [3] because AC4 takes a lot of time while pre-processing.

B. Running Time (msec) vs Percentage of edges

As I have increased the percentage of edges in the graph keeping the number of nodes fixed(eg. 100,200). I got the same result as the previous one. The running time of **AC3** and **AC1** is quite better than the **AC2** and **AC4**. For **AC1** and **AC3** we have noticed that their performance are almost similar for small percentage of edges but when edge percentages then **AC3** performs slightly better than **AC1**. But **AC2** and **AC4** are not so good even for small edge percentages.

C. Running Time (msec) vs Domain Size

According to the graph 3, **AC4** performs worse than other algorithms in average case but performs better in the worst case. **AC4** performs worse for the large domains but **AC1**,**AC2** and **AC3** performs almost similar for the large domain.

TABLE I : Descriptive statistics of k=4 independent treatments:

Treatment $\rightarrow$	AC1	AC2	AC3	AC4	Pooled Total
observations N	101	101	101	101	404
sum $\sum x_i$	961.4854	1,117.2210	650.8736	4,131.0253	6,860.6053
mean $\overline{x}$	9.5197	11.0616	6.4443	40.9012	16.9817
sum of squares $\sum x_i^2$	9,195.8574	12,393.0952	4,201.5039	169,963.7484	195,754.2049
sample variance s^2	0.4285	0.3485	0.0708	9.9969	196.6489
sample std. dev. s	0.6546	0.5903	0.2662	3.1618	14.0232
std. dev. of mean $SE_{\overline{x}}$	0.0651	0.0587	0.0265	0.3146	0.6977

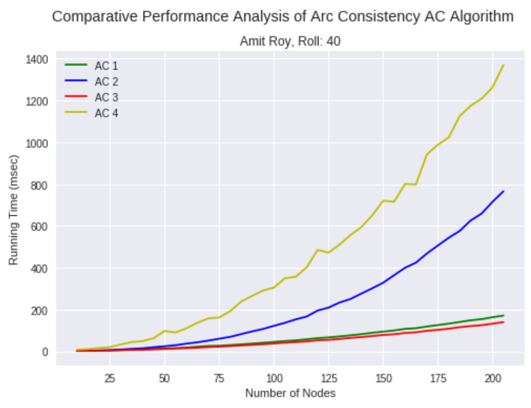

Fig. 1 Running time vs number of nodes (1~200)

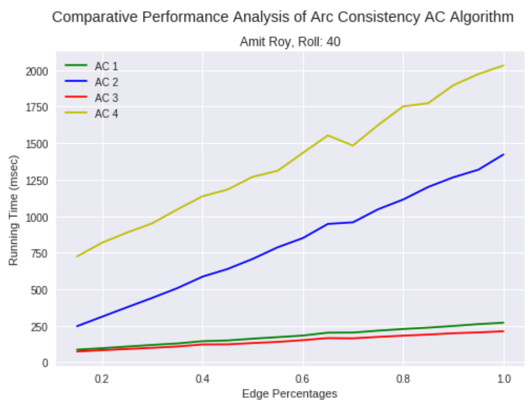

Fig. 2 Running Time vs Edge Percentage (Nodes: 200)

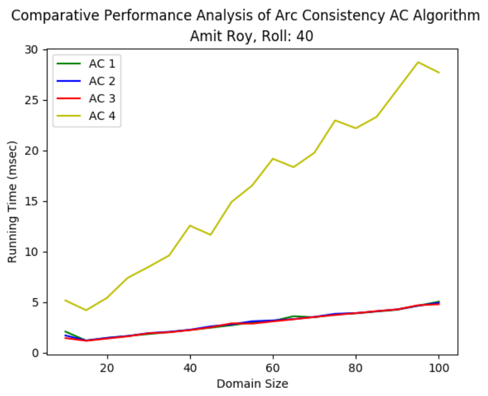

Fig. 3 Running Time vs Domain Size (Nodes: 30, Edge Probability: 0.5)

TABLE II : Tukey HSD results

Treatments Pair	Tukey HSD Q statistic	Tukey HSD p-value	Tukey HSD inferfence	Result
AC1 vs AC2	9.4113	0.0010053	** $p<0.01$	Significant
AC1 vs AC3	18.7706	0.0010053	** $p<0.01$	Significant
AC1 vs AC4	191.5389	0.0010053	** $p<0.01$	Significant
AC2 vs AC3	28.1819	0.0010053	** $p<0.01$	Significant
AC2 vs AC4	182.1276	0.0010053	** $p<0.01$	Significant
AC3 vs AC4	210.3095	0.0010053	** $p<0.01$	Significant

III. Results and Findings

Since all the algorithms reduce the domain of the variables by eliminating inconsistent values from the domain, all algorithms will return the **same domain** if the input graph has a consistent solution. But when the input graph is not consistent then, different algorithms may return different domain due to the termination when some of the variable's domain is empty.
To verify the correctness of the implementation, I have run all 4 algorithms for small input until finding a consistent graph. What we have noticed is all 4 algorithms provide same domain for each variable/node. which proves the correctness of the implementation. I have run all the algorithms in the platform **"google colab"**. You can find and test the implementation here
I have also performed **Anova's test with HSD tuckey** I II on my data of the running time vs number of edges for 50 nodes and edges in range (100~200) Here, A,B,C,D corresponds to AC1,AC2,AC3,AC4. It shows significant results comparing among all the results.

References

[1] A. K. Mackworth, "Consistency in networks of relations," vol. 8, pp. 99–118, Elsevier, 1977.

[2] M. Arangú and M. Salido, "A fine-grained arc-consistency algorithm for non-normalized constraint satisfaction problems," *International Journal of Applied Mathematics and Computer Science*, vol. 21, no. 4, pp. 733–744, 2011.

[3] R. J. Wallace, "Why ac-3 is almost always better than ac-4 for establishing arc consistency in csps," in *IJCAI*, vol. 93, pp. 239–245, Citeseer, 1993.