Performance comparison of arc consistency algorithms

Abdullahil Baki Arif

Roll: 36, Reg: 2015-216-797
Department of CSE, University of Dhaka

Introduction

A Constraint Satisfaction Problem can be defined by a tuple $\langle X, D, C \rangle$. Here, X is a set of variables, D is a set of domains and C is a set of constraints which specify relation among pair of variables.
Constraint graph of a CSP is the graph whose nodes are the variables and an arc joins a pair of variables if the two variables occur together in a constraint.
In this assignment, I implemented and compared the performances of different arc consistency algorithms AC1, AC2, AC3 and AC4.

Constraint graph generation

Graph generation: For random graph generation, I have used networkx library of python and Erdos-Renyi model with a probability of edge = 0.40.
Domain generation: For each node, the values of its domain are selected randomly from the range [1, 1000].
Assigning constraints: For each arc, a random constraint from predefined constraint list is assigned. The constraints are: $X > Y$, $3X < 4Y$, $(X+Y) \bmod 10 = 0$ and $gcd(X, Y) = 1$.

Comparison

After generating constraint graph, I implemented the arc consistency algorithms and compared their performances based on different metrics.

Number of nodes vs running time: For n from 10 to 110 with a increment of 3, I ran the algorithms 20 times in different constraint graph with n nodes and recorded average time. For this comparison, domain size was fixed at 30 and edge probability was fixed at 0.40.

Domain size vs running time: For this comparison, number of nodes was fixed at 40 and edge probability was fixed at 0.30 and domain size was varied from 30 to 100. I ran the algorithms 20 times and recorded average running time.

Graph density vs running time: For this comparison, number of nodes was fixed at 50 and domain size was fixed at 30. For edge probability from 0.10 to 0.60, I ran the algorithms 20 times and recorded average running time.

Results

The comparison graphs are shown below:

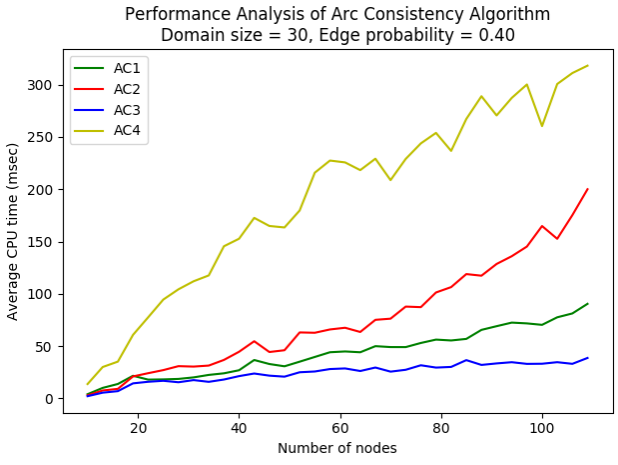

Figure 1: Number of nodes vs average running time

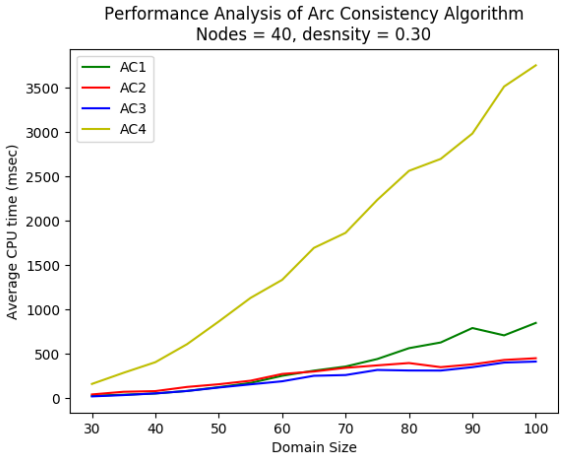

Figure 2: Domain size vs average running time

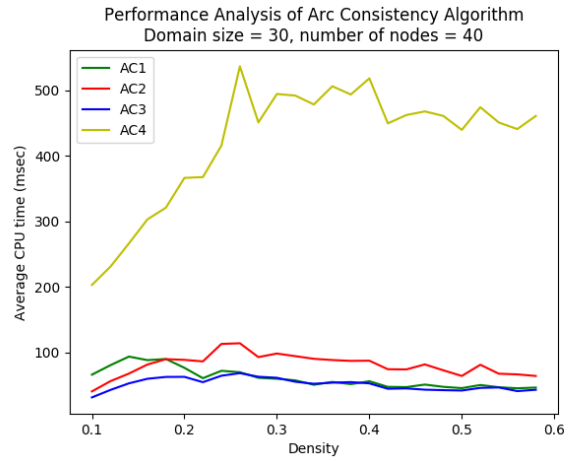

Figure 3: Density vs average running time

Statistical significance test

To determine whether the result is statistically significant, I performed One-way ANOVA with post-hoc Tukey HSD Test for number of nodes vs average running time comparison. The results are shown below:

treatment pair	Tukey HSD Q statistic	Tukey HSD p-value	Tukey HSD inference
A vs B	3.7134	0.0470791	**p<0.05
A vs C	2.1023	0.4498175	insignificant
A vs D	16.5363	0.0010053	**p<0.01
B vs C	5.8157	0.0010053	**p<0.01
B vs D	12.8229	0.0010053	**p<0.01
C vs D	18.6386	0.0010053	**p<0.01

Table 1: Tukey HSD results

Findings

From the results, we can compare the performances of different arc consistency algorithms based on different metrics although the performances depend on different factors such as types of constraints, domain size and can vary for different data structures used in implementation.

From the first graph, we can see that AC 3 is performing better than other algorithms. Running time of AC3 is increasing very slowly than others.

From the domain size vs time graph, we can see that AC2 is performing better than AC1 but running time of AC 4 increases drastically for large domains. Because in AC4's preprocessing step, the amount of support from variable vj is calculated for each ai in the domain of vi which is costly but helps to avoid having to redo the same constraint checking several times during the propagation of deletions. So AC4 is best performing in worst case with a complexity of $O(e.k^2)$ where e = number of edges and k = size of domain.

From third graph we can see that in sparse graphs, AC3 is performs better than AC1. Because in AC3, when a node's domain is revised, only it's neighbors are pushed in queue. But in AC1, all arcs are checked again if a domain is revised.