Verification of the generality of your A* algorithm

The setup used is that the A* star instance(of class Seach) gets called after the gac instance (GAC class) is done the firste iteration. A* then makes guesses and calls domain_filtering_loop which is a method inside the GAC-class

Verification of the generality of your A*-GAC algorithm

The algorithm runs A* and GAC are both different classes. GAC's domainfiltering method gets called multiple times by A*. The classe can be expanded by giving the same input types and by modifying the createconstraint method.

Verification of a clean separation between your Constraint Network (CNET) and the VIs and CIs of the search states

An instance of GAC is given as an argument when constructing an instance of the Search Class(which contains teh function a_star()). a_star() is run from an instance of Search and is only given the parameter of the start state. The a_start() sends the state of the nodes it poppes from the openList to the GAC

Explanation of code chunk creation

The graph object, which contains all the graph information creates all contraints before the GAC and A* algorithms run. By combining variable names such as n0_n1 the function createConstraints creates the constraint beteen the two variables both ways. All the constraints are precomputed before the GAC algorithm runs

```
# In this case all constraints are the same
def createConstraints(self,edges):
    constraints = {}
    for e in edges:
        var1 = 'n' + str(e[0])
        var2 = 'n' + str(e[1])
        func = self.makefunc(['a','b'],"a!=b")
        constraint = [[var1,var2],func]
        # Adding constraints both ways
        constraints[str(var1)+'_'+str(var2)] = constraint
        constraints[str(var2)+'_'+str(var1)] = constraint


    return constraints


def makefunc(self, var_names, expression, envir=globals()):
    args = ""
    for n in var_names: args = args + "," + n
    return eval("(lambda " + args[1:] + ": " + expression + ")", envir)
```