CS2180-Artificial Intelligence

Lab 3

By, Pratham Ravindra Nagpure , roll no. : 112001054

Formulating the constraint satisfaction problem:

The variables are the empty cells in the kakuro puzzle

Each of the cell can take a value from the set of integers from 1 to 9 so, the domains of these variables are set of all integers from 1 to 9

The constraints on the empty cells are that each continuous horizontal or vertical run of empty cells should add up to the value to the left of it or above it respectively such that no two entries in a run of empty cells are equal

For example:

If there is a continuous line of empty cells of length n and the corresponding sum is S

Then x_1, x_2, x_3 ..., x_n are the variables such that they form the continuous empty cells line in order

Where domain of every x_i is integers 1 to 9 such that i belong to integers 1 to n

Then constraints on x_1, x_2, x_3..., x_n are

x_1 + x_2 + x_3 + ... + x_n = S

such that for every pair (x_i,x_j) such that i != j, x_i != x_j where i and j are from the set of integers 1 to n

We can observe that the sum constraint is n-ary constraint and other are binary so now we have to convert them

Converting n-ary constraints to binary constraints:

For every sum constraint a new variable U can be introduced

The domain of that U is set of all tuples such that this tuple has the all the ordered combinations of the n empty cells which satisfy the constraint

And the neighbouring variables of U are all variables of empty cells and these empty cells will have

For example:

If there are variables x1, x2, x3, x4, x5 having constraints such that

x1, x2, x3 form a block of empty cells with sum 6

x2, x4, x5 form a block of empty cells with sum 7

Then U1 and U2 are the new variables with domain as

[(1,2,3),(1,3,2),(2,1,3),(2,3,1),(3,1,2),(3,2,1)] and [(1,2,4),(1,4,2),(2,1,4),(2,4,1),(4,1,2),(4,2,1)]

And there are constraints between U1 and x1, U1 and x2, U1 and x3, U2 and x2, U2 and x4, U2 and x5 and vice-versa among them

For example U1 and x1 constraint is x1's position in the tuple and similarly for other constraints

Or we can say the constraint is where the value of U1 is tuple and it can be indexed

Value_of_U1[position_of_x1_in_U1] = Value_of_x1

The constraint that all values should be distinct in a sum are handled by keeping the domain of the new introduced variables such that the tuples are distinct valued so there is no need of this constraint

In the python program:

The variables is a list and will have the variables as strings:

For an empty cells at position (i,j) in grid is Xi,j

For new introduced sum variables at position (i,j) in grid URi,j for right-wards empty cells and UDi,j for downwards empty cells

The domains are a dictionary with key values as the string variable name and domain is a list of the values

The constraints is a dictionary if Xi,j and URi,j have a constraint then there will be key such that the joined string URi,j Xi,j is mapped to position of Xi,j in tuples in domain of URi,j

The neighbours of some variable v are variables that are constrained with v, this is a dictionary where the keys are variable names and are mapped to the list of neighbours

All these are in a class CSP

Applying node and arc consistency

A unary constraint can be applied as in a block of empty cells with a sum each entry in the empty cell will be less than sum as we don't have a 0 or negative entry. Then assign values of the domain of the empty cell variables from 1 to min(sum-1,9)

In the arc consistency if the input to function is only the csp then do arc consistency on all the arcs, in mac the input queue is given

Generic Backtracking search

Generic search works only on small inputs as it is not able to prune anything

Backtracking with MAC

This works much faster than the generic backtracking search as after a value is assigned to a variable we are able to prune the domain for further values

Algorithms analysis for inputs

BS is generic backtracking search

In BS-AC3, first AC3 is applied on all arcs to get reduced domain and then generic backtracking

In BS-AC3, first AC3 is applied on all arcs to get reduced domain and then backtracking while maintaining arc consistency

Note:

input1.txt and input4.txt were same files so input4.txt is not in table.

Entries with ' - ' means did not solve within 10 minutes so search was stopped so no table value to show.

Input file	Algorithm	Backtracking calls	consistency checks by backtracking	Time taken in seconds
simple/input0.txt	BS	1205	4530	0.126
simple/input0.txt	BS-AC3	37	51	0.118
simple/input0.txt	BS-MAC	22	22	0.124
simple/input1.txt	BS	-	-	-
simple/input1.txt	BS-AC3	12554	27540	0.250
simple/input1.txt	BS-MAC	61	68	0.208
simple/input2.txt	BS	-	-	-
simple/input2.txt	BS-AC3	-	-	-
simple/input2.txt	BS-MAC	228	291	24.937
simple/input3.txt	BS	-	-	-
simple/input3.txt	BS-AC3	49	52	0.284
simple/input3.txt	BS-MAC	49	48	0.291
simple/input5.txt	BS	-	-	-
simple/input5.txt	BS-AC3	-	-	-
simple/input5.txt	BS-MAC	562	1103	5.052
simple/input6.txt	BS	-	-	-
simple/input6.txt	BS-AC3	-	-	-
simple/input6.txt	BS-MAC	255	473	3.234

Inferences:

Generic backtracking works for small inputs but takes a lot of time for larger variables, it can be improved by running ac3 on all arcs by which domain is pruned so backtrack works faster.

Further improvement can be done by maintaining arc consistency during the backtrack which helps pruning the domains in search and lesser consistency checks and value assignments