CS2180 - Lab3

Roll No : 112001028,112001010

REPORT

File name	Count BS	Count MAC	Time BS	Time MAC
input0	13	13	0.57	0.61
input1	7576	37	1.48	1.51
input2	-	131	-	5.97
input3	29	29	1.3	1.25
input4	7576	37	1.58	1.51
input5	36159	62	2.44	2.46
input6	-	74	-	2.92
input7	-	140	-	8.12

Count BS - No.of backtrack calls in generic backtracking search with AC-3 applied before to reduce domain.
Count MAC - No.of backtrack calls in backtracking search with Maintaining Arc Consistency after each variable assignment.
Time BS - Time taken in seconds for generic backtracking search.
Time MAC - Time taken in seconds for backtracking search with Maintaining arc consistency.

Outputs generated for all given inputs with bs_mac are present in the folder **outputs**.

Formulating the puzzle as general CSP:

In kakuro we have row and column constraints where a sum is given and a certain number of variables should sum up to the given constraint and the value of variables should be distinct. (All diff constraint)

Domain - to satisfy the sum constraint the variables can take values from 1 to 9

Example:

Say the row sum constraint for some row is 7 and the number of variables whose sum needs to be 7 is 3. Say, x1,x2,x3 are the three variables. The first constraint on them is x1+x2+x3 = 7. The second constraint is x1!=x2 and x2!=x3 and x1!=x3. The domain of these variables can be from 1 to 9

N-ary to binary constraint conversion:

From the above example we have constraints as x1+x2+x3=7 and all of x1,x2,x3 should be distinct. But it is a N-ary constraint as all three variables are involved in the constraint , here N = 3.
To make it into a binary constraint we can generate all possible permutations of three values which satisfies the given constraints. For above example , one such permutation is 1,2,4. So the binary constraints are (x1,1,x2,2) , (x2,2,x3,4) , (x1,1,x3,4) which is of the form (var1,value1,var2,value2) which means there is a possible permutation in which var1 can take value1 and var2 can take value2. Here both sum and all diff constraints are maintained. These constraints are stored in a dictionary. Neighbours of each variable are also stored in a dictionary. In the above example neighbours of x1 are x2 and x3, neighbours of x2 are x1 and x3 and neighbours of x3 are x1 and x2.

Node consistency:

For each variable domain of that variable is min(9,sum) , where sum is the sum constraint of that variable.

Arc consistency:

Standard AC-3 algorithm is applied , initially all arcs are put in the queue. It is used to reduce the domain. If we consider the arc Xi,Xj then for some value in domain of Xi if there is no value which satisfies the csp is in Xj then the value is removed from domain for Xi.

Generic Backtracking search:

First an unassigned variable is taken and for each value in the domain of that variable if the value is consistent with assigned variables then we assign this value to this variable and backtracking search is called again with this assignment. If assignment of some variable leads to Failure then we backtrack from there by removing that assignment to that variable and then trying the next value for that variable.

Backtracking search with MAC:

It is same as generic backtracking search but additionally when we assign some value to a variable we call the AC-3 algorithm in which the queue initially has the arcs (Xj,Xi) for all Xj which is unassigned neighbour of Xi. Xi is the variable to which value was assigned. AC-3 returns false when after some assignment of value to a variable, some of its neighbour's domain becomes empty. When AC-3 returns false the assignment is not possible so we backtrack.