COMP 557: Assignment #4a

Due on Wednesday, October 5, 2016

Misiura Mikita & Lee Call

Problem 1. Sudoku and constraint satisfaction

Part 1

1. For $i \in \{1..9\}$: $AllDiff(V_{ij})$, where $j \in \{1..9\}$
2. For $j \in \{1..9\}$: $AllDiff(V_{ij})$, where $i \in \{1..9\}$
3. For $n, k \in \{0..2\}$: $AllDiff(V_{3n+i,3k+j})$, where $i, j \in \{1..3\}$

Part 2

For each unassigned cell, forward checking will exclude all values, that do not satisfy our constraints.

For example, cell X_{74}:

1. Initial $D_{74} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
2. After checking row 7: $D_{74} = \{1, 2, 3, 4, 5, 6\}$
3. After checking column 4: $D_{74} = \{1, 2, 4, 5\}$
4. After checking 3x3 square: $D_{74} = \{1, 4, 5\}$

Part 3

With most-constrained variable heuristic our flow will consist of repetition of two following steps:

1. Given partially solved puzzle, run forward checking from Part 2 to determine number of possible values for each unassigned cell.
2. Select the cell with the smallest number of possible values and assign it.

For example, X_{96} has only one possible value - 1, and should be assigned next. After that, we can assign cells X_{74} and X_{75}, and so on.

Part 4

Forward checking will not find this. It will only look at our constraints and define that our domain is $D_{48} = \{4, 5, 7\}$. So, it will show that 7 is just among possible values of this cell.

Arc consistency will remove at least the same values from X_{48}'s domain, so it will leave us $D_{48} = \{4, 5, 7\}$ to begin with. Now, lets us make D_{48} fully consistent with neighbours. Below is the list of X_{48}'s neighbours with deleted values due to constraints for simplicity (these are now in our queue):

$D_{18} = \{9\}$

$D_{58} = \{5\}$

$D_{98} = \{4, 9\}$

$D_{41} = \{1, 6\}$

$D_{42} = \{1, 5, 6\}$

$D_{45} = \{1, 5, 7\}$

$D_{46} = \{1, 2, 5, 7\}$

$D_{49} = \{1, 2, 4, 6\}$

1. Lets check 4 as perspective value for X_{48}. This will remove 4 from D_{98} and D_{49}, so we add them to the queue:

   $D_{98} = \{9\}$
   $D_{49} = \{1, 2, 6\}$

   Now if we will take a look at X_{98}'s neighbours, we will see that X_{18} with $D_{18} = \{9\}$ (see above) is among them, so that leaves us without choices for X_{18}! As a result $-$ 4 is deleted from D_{48}.

2. Lets check 5 as a perspective value for X_{48}. In the list above we have a following neighbour:

   $D_{58} = \{5\}$

   So we reject 5 as a possible value for X_{48}.

3. Lets check 7 as a perspective value for X_{48}. This choice will remove 7 from D_{45} and D_{46}, so we add them to the queue:

   $D_{45} = \{1, 5\}$
   $D_{46} = \{1, 2, 5\}$

   Then we have to check their neighbours and so on...

But if there is indeed exists a solution for this Sudoku (and it **does** exist), arc consistency will find that the only possible value for X_{48} is 7.

Problem 2. Constraint satisfaction with non-binary factors

Part 1

For each non-binary factor in the original CSP, on the set of k variables $X_1, ..., X_k$, create a new auxiliary variable whose domain is the set of k-tuples which satisfy the original factor. This creates a new CSP whose set of variables is composed of all the original variables plus the set of newly constructed variables. In this new CSP the original factor is replaced by k new binary factors relating the new variable to each of the original variables $X_1, ..., X_k$.

Part 2

The new variable A has domain D_A = {(green, green, red), (green, green, blue), (green, red, green), (green, blue, green), (red, green, green), (blue, green, green)}.

We now have these new binary factors which together replace the original non-binary factor:

$F_1(A, X_1) = 1[A_i == X_1]$ where $A_i = i^{th}$ element of A and i=1.

$F_2(A, X_2) = 1[A_i == X_2]$ where $A_i = i^{th}$ element of A and i=2.

$F_3(A, X_3) = 1[A_i == X_3]$ where $A_i = i^{th}$ element of A and i=3.

Part 3

X_1	X_2	X_3
X_4		X_5
X_6	X_7	X_8

Formal description of our CSP:

1. X: $\{X_1, ..., X_8\}$
2. D_i: {All letters of the alphabet}
3. C: We need four 3-ary factors and one 4-ary factor:

$f_1(X_1, X_2, X_3) = 1[w_1 \in D \text{ and } w_1 : \{w_{11} = X_1, w_{12} = X_2, w_{13} = X_3\}]$
$f_2(X_6, X_7, X_8) = 1[w_2 \in D \text{ and } w_2 : \{w_{21} = X_6, w_{22} = X_7, w_{23} = X_8\}]$
$f_3(X_1, X_4, X_6) = 1[w_3 \in D \text{ and } w_3 : \{w_{31} = X_1, w_{32} = X_4, w_{33} = X_6\}]$
$f_4(X_3, X_5, X_8) = 1[w_4 \in D \text{ and } w_4 : \{w_{41} = X_3, w_{42} = X_5, w_{43} = X_8\}]$
$f_5 = alldiff(w_1, w_2, w_3, w_4)$

Part 4

The domains for each of the 3-ary factors above will be identical and will consist of n sets of tuples, each tuple being a set of letters from a word in the dictionary, n being the number of three-letter words in the dictionary. For example:

$$D_i = \{('C','A','P'), ('C','A','T'), ...\} \quad (1)$$

Problem 4. Sudoku and repair algorithms

The general repair algorithm - changing variables one by one using min-conflict heuristic - is certainly will not suit here.

We can suggest a following modification of the repair algorithm to make it more or less suitable to solve Sudoku:

1. Start from some partial assignment (like in the example in Problem 1 in this HW) and complete it randomly.
2. Select the first column and re-assign it. If the first column has some pre-assigned cells, we keep them as they are, and the rest of the cells are re-assigned randomly until we found an assignment which has minimum number of violated constraints. We can use random search here, of we can try all assignments in some order. In the worst case, if this column has no pre-assigned cells, we have 9! different assignments. In principle, we can search through them all and select the best one.

3. Repeat step 2 for all other columns. If after one run through all columns we still have some violated constraints, we can start again from the column 1 and continue in this way until we found a solution.

Usually Sudoku puzzle has one solution (anyway, a small number of solutions at least), but the search space is quite large. To estimate - for empty board it's 9! of possible values for each column, 9 columns:

$$9 \cdot 9! = 3,265,920 \tag{2}$$

So the probability that our random assignment will be somewhat close to the solution is negligible and we will often stuck in various local minima.

As a conclusion, we can say that we would recommend a constructive search instead of a repair algorithm due to small number of solutions, though with a quite small state space of this problem both algorithms will be quite fast.