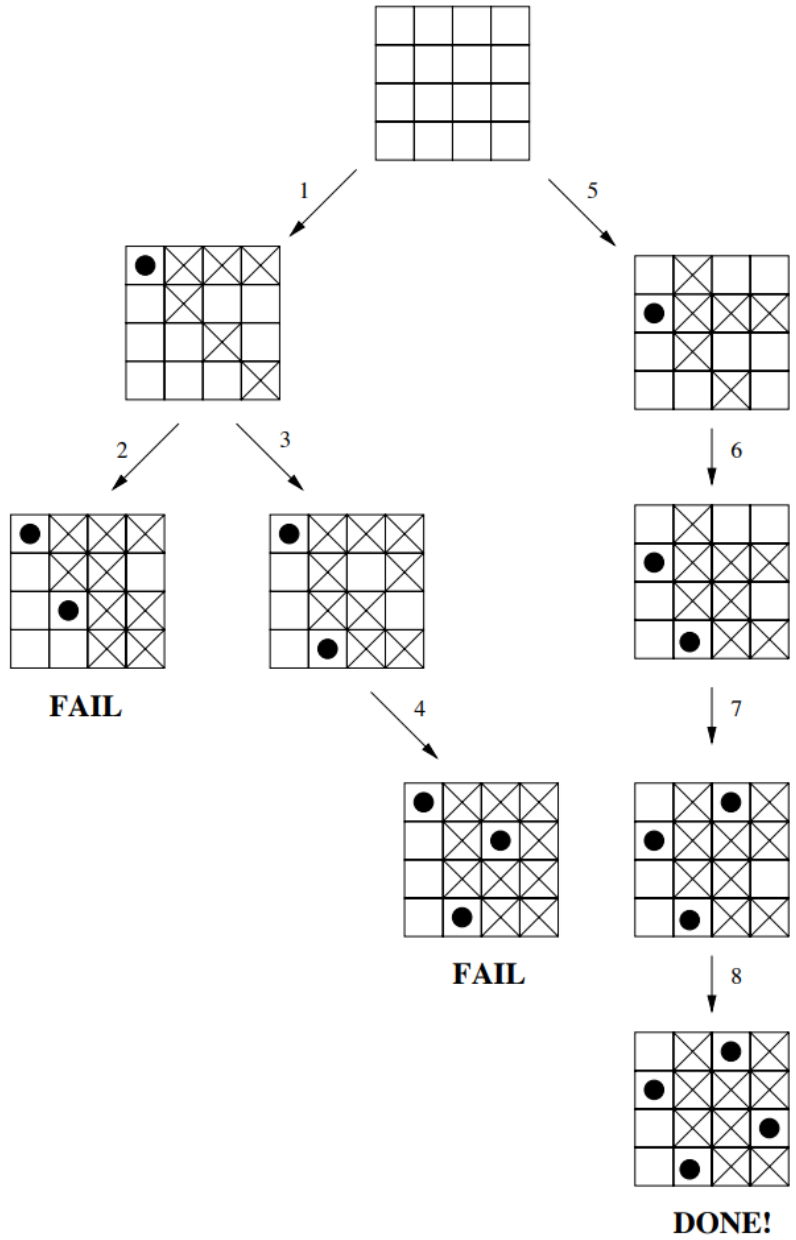
National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 5: Constraint Satisfaction Problems

SOLUTIONS

1. Consider the 4-queens problem on a 4 × 4 chess board. Suppose the leftmost column is column 1, and the topmost row is row 1. Let Q_i denote the row number of the queen in column i, where i = 1, 2, 3, 4. Assume that variables are assigned in the order Q_1, Q_2, Q_3, Q_4, and the domain values of Q_i are applied in the order 1, 2, 3, 4. Show a trace of the backtracking algorithm with forward checking to solve the 4-queens problem.

   Solution: The following is the trace of the search tree.



2. Consider the following constraint satisfaction problem (CSP). We have 3 variables x_1, x_2, and x_3, with domains corresponding to integers ranging from 1 to 7 (inclusive). We also have the following constraints.

- $x_1 = 2x_2$
- $x_2 = x_3 + 1$

(a) Trace the execution of the AC3 algorithm as a pre-processing step. Assume the following initial arc queue: $[(x_1, x_2),(x_2, x_3),(x_3, x_2),(x_2, x_1)]$. Note that if an arc A is already in the queue, we do not re-queue arc A.

Solution: The execution of the AC3 algorithm results in the following trace table.

Popped	Updated Domain	Enqueued
(x_1, x_2)	$x_1 = [2, 4, 6]$	NA
(x_2, x_3)	$x_2 = [2, 3, 4, 5, 6, 7]$	(x_1, x_2)
(x_3, x_2)	$x_3 = [1, 2, 3, 4, 5, 6]$	NA
(x_2, x_1)	$x_2 = [2, 3]$	(x_3, x_2)
(x_1, x_2)	$x_1 = [4, 6]$	NA
(x_3, x_2)	$x_3 = [1, 2]$	NA

(b) Apply the Backtracking algorithm (with forward-checking) on the pre-processed version of the queue. Use the ordering x_1, then x_2, then x_3 when deciding variable order, and use the largest value in the domain first when deciding value order.

Solution: The execution of the Backtracking algorithm results in the following trace table.

Variable to Assign	Assigned Value	Updated Domains
x_1	6	$x_2 = [3]$
x_2	3	$x_3 = [2]$
x_3	2	Solved

3. In a particular single round-robin scheduling problem, there are n teams and $n – 1$ time slots in a tournament, where $n \geq 2$, and n is always even. The objective is to find a round-robin scheduling such that the following constraints are satisfied – i.e., by solving it as a ***constraint satisfaction problem*** (CSP).

- ***Constraint 1***: No team can play itself.
- ***Constraint 2***: For each time slot, each team must play exactly once.
- ***Constraint 3***: All teams must play against every other team exactly once.

For example, suppose that we have $n = 4$ teams. Denote these teams as X_0, X_1, X_2, and X_3. In this case, we will have 3 time slots, which we will denote as T_0, T_1, and T_2. A possible consistent schedule would then be as follows.

- T_0: { (X_0, X_1), (X_2, X_3) }
- T_1: { (X_0, X_2), (X_1, X_3) }
- T_2: { (X_0, X_3), (X_1, X_2) }

Given that each (X_i, X_j) above corresponds to a match between X_i and X_j, we observe the following.

- No team plays against itself (i.e., ***Constraint 1*** is satisfied).
- For each time slot, all teams play exactly once (i.e., ***Constraint 2*** is satisfied).
- All teams play against every other team exactly once (i.e., ***Constraint 3*** is satisfied).

To define the variables in this CSP, we first define a single matrix, $\boldsymbol{A}$, with n rows and $n - 1$ columns. The rows correspond to the teams, while the columns correspond to the time slots. The elements of this matrix thus correspond to the opponents.

For example, $\boldsymbol{A}[0, 2] = 3$ means that team X_0 will compete against team X_3 in time slot T_2.

More generally, notice that $\boldsymbol{A}[i, j] = k$ corresponds to the allocation of a match between X_i and X_k in time slot T_j.

The variables in this scheduling problem thus correspond to the elements of the matrix, while the domains correspond to positive integers from 0 to n (exclusive). The matrix on the right describes the schedule given in the example above.

1	2	3
0	3	2
3	0	1
2	1	0

Use the above variables and domains to define the constraints for this CSP.

You are allowed to use logical operators – i.e., the for-all operator, ∀; the there-exists operator, ∃, etc.

You are also allowed to use the `AllDiff` constraint, with the following syntax: `AllDiff`($\boldsymbol{S}$), where $\boldsymbol{S}$ is the set of variables whose values must be different. However, do note that you ***may not*** use the `ExactlyOnce` function.

For simplicity of definitions, you may use the following sets if necessary:

- $\boldsymbol{X} = \{ 0, 1, 2, \ldots, n - 1 \}$
- $\boldsymbol{T} = \{ 0, 1, 2, \ldots, n - 2 \}$

Solution: The constraints may be defined as follows.

- No team can play itself:
  $\forall x \in \boldsymbol{X}, \forall t \in \boldsymbol{T}: \boldsymbol{A}[x, t] \neq x$
- For each time slot, all teams play exactly once:
  $\forall x \in \boldsymbol{X}, \forall t \in \boldsymbol{T}: \boldsymbol{A}[\boldsymbol{A}[x, t], t] = x$
- All teams play against every other team exactly once:
  $\forall x \in \boldsymbol{X}$: `AllDiff`($\{\boldsymbol{A}[x, t] \mid \forall t \in \boldsymbol{T}\}$)
  // i.e., for each team x, opponents across all time slots are different.

4. Consider the item allocation problem. We have a group of people $N = \{1, \ldots, n\}$, and a group of items $G = \{g_1, \ldots, g_m\}$. Each person $i \in N$ has a utility function $u_i: G \to \mathbb{R}_+$. The constraint is that every person is assigned ***at most one item***, and each item is assigned to ***at most one person***. An allocation simply says which person gets which item (if any).

   In what follows, you ***must*** use ***only*** the binary variables $x_{i,j} \in \{0, 1\}$, where $x_{i,j} = 1$ if person i receives the good g_j, and is 0 otherwise.

   (a) Write out the constraints: '*each person receives no more than item*' and '*each item goes to at most one person*', using only the $x_{i,j}$ variables[1].

[1] You may use simple algebraic functions −, +, ×, ÷, and numbers.

Solution: The constraints may be defined as follows.

$$\forall i \in N: \sum_{g_j \in G} x_{i,j} \leq 1$$

$$\forall g_j \in G: \sum_{i \in N} x_{i,j} \leq 1$$

(b) Suppose that people are divided into ***disjoint types*** $N_1, \ldots, N_k$ (think of, say, genders or ethnicities), and items are divided into disjoint blocks $G_1, \ldots, G_\ell$. We further require that each N_p only be allowed to take no more than $\lambda_{p,q}$ items from block G_q. Write out this constraint using the $x_{i,j}$ variables. (Note that each N_i corresponds to the set of people who are of that person type.)

Solution: The constraint may be defined as follows.

$$\forall p \in [k], q \in [\ell]: \sum_{i \in N_p} \sum_{g_j \in G_q} x_{i,j} \leq \lambda_{p,q}$$

(c) We say that player i envies player i' if the utility that player i has from their assigned item is strictly lower than the utility that player i has from the item assigned to player i'. Write out the constraints that ensure that in the allocation, no player envies any other player. You may assume that the validity constraints from (a) hold.

Solution: Note that for this constraint, the definition requires that the allocation is valid, so you will need to add the constraints from (a) to make either definition below meaningful.

$$\forall i, i' \in N, \forall g_j, g_{j'} \in G: (x_{i,j} \wedge x_{i',j'}) \Rightarrow u_i(g_j) \geq u_i(g_{j'})$$

OR

$$\forall i, i' \in N: \left(\left(\sum_{g_j \in G} x_{i,j} \cdot u_i(g_j)\right) > 0\right) \Rightarrow \left(\left(\sum_{g_j \in G} x_{i,j} \cdot u_i(g_j)\right) \geq \left(\sum_{g_j \in G} x_{i',j} \cdot u_i(g_j)\right)\right)$$