

National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 5: Constraint Satisfaction Problems

Issued: Week 6

Discussion in: Week 7

Important Instructions:

- **Tutorial Assignment 5** consists of **Question 2** from this tutorial.
- Your solution(s) must be TYPE-WRITTEN, though diagrams may be hand-drawn.
- You must submit your solution(s) via **Canvas > Assignments**, satisfying the deadlines:
 - **Pre-tutorial 5 submission by Recess Week Sunday, 2359 hrs.**
 - **Post-tutorial 5 submission by Week 7 Friday, 2359 hrs.**
- You must make both submissions for your assignment score to be counted.

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.

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 .
 - (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.

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, 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, $A[0, 2] = 3$ means that team X_0 will compete against team X_3 in time slot T_2 .

More generally, notice that $A[i, j] = k$ 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, \forall ; the there-exists operator, \exists , etc.

You are also allowed to use the `AllDiff` constraint, with the following syntax: `AllDiff(S)`, where 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:

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

4. Consider the item allocation problem. We have a group of people $N = \{1, \dots, n\}$, and a group of items $G = \{g_1, \dots, g_m\}$. Each person $i \in N$ has a utility function $u_i: G \rightarrow \mathbb{R}_+$. 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 $x_{ij} \in \{0, 1\}$, where $x_{ij} = 1$ 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_{ij} variables¹.
- (b) Suppose that people are divided into **disjoint types** N_1, \dots, N_k (think of, say, genders or ethnicities), and items are divided into disjoint blocks G_1, \dots, G_t . 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_{ij} variables. (Note that each N_i corresponds to the set of people who are of that person type.)
- (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.

¹ You may use simple algebraic functions $-$, $+$, \times , \div , and numbers.