# CS3243 : Introduction to Artificial Intelligence

Tutorial 4

NUS School of Computing

July 14, 2022

# Admin

- Midterm review
- Tutorial Assignment due tonight

# Review

- Constraint Satisfaction Problems
- In some problems, we need to search for the solution subjected to certain constraints/rules.
- It may be more natural to express such problems in the form of their constraints in order to capture these restrictions fully when we are solving the problem, hence Constraint Satisfaction Problems!
- Main Goal : Find a satisfying solution, where there is a valid assignment to each variable such that all constraints are satisfied. In other words, find a complete (all variables are set) and consistent (no constraint is violated) assignment.

# Review

- Formal Representation of CSPs
- A set of variables $X = \{x_1, x_2, ..x_n\}$
- Each of the variables have a domains which contain legal assignable values

$$D_{x_i} = \{.\}$$

- Set of constraints that are to be satisfied. Examples include, but not limited to : $x + y \leq 5, x_1 \neq x_2$
- The goal is basically to find a set of assignments to all variables $x_i = y_i$, where $y_i \in D_{x_i}$ $\forall i$

# Review

- Solving CSP with what seems obvious : Backtracking Search
  - Depth-First-Search Style Approach
  - Traverse along the depth of the search tree to find the solution
  - At every level of depth of the search, assignment is considered for a single variable
  - If the search arrives at a failure node (example due to the inference made, or illegal assignment), backtrack and traverse the next path
  - Repeat until a solution is found (or return no solution)
- Thing to note : The order of assignment is irrelevant (we don't care about the path, we only care about the goal)
- Scope for improvement : Use of heuristics to improve search efficiency

# Review

- The simple Backtracking Search algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
        remove {var = value} and inferences from assignment
    return failure
```

# Review

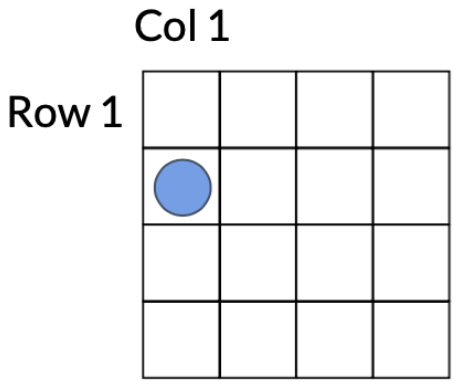
- SELECT-UNASSIGNED-VARIABLE(...)
  - Determines the order in which variables are assigned
  - Outputs the next variable to be assigned a value
  - Heuristics : Minimum Remaining Value (MRV) Heuristic, and Degree Heuristic
- ORDER-DOMAIN-VALUES(...)
  - Determines the order of the values to be tried for a variable
  - Picks out the legal values from the domain of a variable to be checked for assignment
  - Heuristics : Least Constraining Value Heuristic
- INFERENCE(...)
  - Infers the domain reductions of variables that are possible upon a value assignment that is made for a particular variable
  - Advantage is the ability to gauge the occurence of a terminal state early on during the search
  - Strategies : Forward Checking, and AC-3

# Heuristics and Strategies

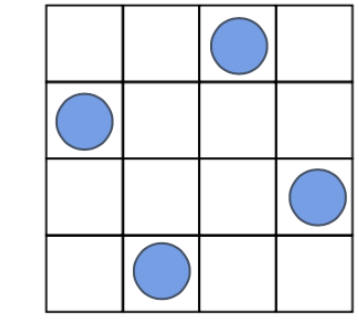
- Minimum Remaining Value (MRV) Heuristic
- Degree Heuristic
- Least Constraining Value Heuristic
- Forward Checking
- Arc Consistency and AC-3

# Tutorial Question 1

- 4-Queens Problem
- Trace Backtracking with Forward Checking
- Given :
  - Check the Variables, Domains, and Constraints
  - Variables : $Q_1,\ Q_2,\ Q_3,\ Q_4$
  - Domains for each of the variables : $D_{Q_i} = \{1, 2, 3, 4\}$
  - Notation that we are following : $Q_1 = 2$ means that the Queen is in 1st Column, 2nd Row




Example queen position at $Q_1 = 2$



Recap: Example solution for 4-queens problem

# Tutorial Question 1

- Backtracking Algorithm : DFS-like algorithm for CSPs to find consistent and complete assignments
- A few metrics :
  - Maximum depth : Number of variables to be assigned
  - Branching factor : Number of legal domain values
- For this question, we are supposed to use Forward Checking

# The Trace

# Tutorial Question 2

- Given Information
- The classes
  - $C_1$ : Programming Methodology : 8:00am to 9:00am
  - $C_2$ : Discrete Structures : 8:30am to 9:30am
  - $C_3$ : Data Structures and Algorithms : 9:00am to 10:00am
  - $C_4$ : Introduction to Artificial Intelligence : 9:00am to 10:00am
  - $C_5$ : Machine Learning : 9:30am to 10:30am
- The professors and their availabilities
  - Professor Tess : $C_3,\ C_4$
  - Professor Jill : $C_2,\ C_3,\ C_4,\ C_5$
  - Professor Bell : $C_1,\ C_2,\ C_3,\ C_4,\ C_5$

# Tutorial Question 2(a)

- Formulating CSPs
- Using the given information, we have to figure out the variables, domains, and constraints

# Tutorial Question 2(a)

- Formulating CSPs
- Using the given information, we have to figure out the variables, domains, and constraints

| Variables | Domains |
|---|---|
| $C_1$ | Bell |
| $C_2$ | Jill, Bell |
| $C_3$ | Tess, Jill, Bell |
| $C_4$ | Tess, Jill, Bell |
| $C_5$ | Jill, Bell |

- Constraints :
  $C_1 \neq C_2,\ C_2 \neq C_3,\ C_3 \neq C_4,\ C_4 \neq C_5,\ C_2 \neq C_4,\ C_3 \neq C_5$

# Tutorial Question 2(b)

- One possible solution for the problem :
- $C_1 =$ Bell
- $C_2 =$ Jill
- $C_3 =$ Bell
- $C_4 =$ Tess
- $C_5 =$ Jill

# Tutorial Question 4

- Formulating Constraints in CSP
- Given Information :
- People : $N = \{1, .., n\}$
- Items : $G = \{g_1, .., g_n\}$
- For each person $i \in N$, there is a utility function $u_i$
- Binary Variable $x_{i,j} \in \{0, 1\}$, representing whether person $i$ received item $g_j$

# Tutorial Question 4(a)

- "Each person receives no more than one item"
- "Each item goes to at most one person"
- Need to show these both constraints only in terms of $x_{i,j}$ variables
- Any thoughts?

# Tutorial Question 4(a)

- "Each person receives no more than one item"
- "Each item goes to at most one person"
- Need to show these both constraints only in terms of $x_{i,j}$ variables
- Any thoughts?

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

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

# Tutorial Question 4(b)

- People are divided into disjoint types $N_1, .., N_k$
- Items are divided into disjoint blocks $G_1, .., G_l$
- Constraint : Each $N_p$ should not take more than $\lambda_{pq}$ items from block $G_q$
- Any thoughts?

# Tutorial Question 4(b)

- People are divided into disjoint types $N_1, .., N_k$
- Items are divided into disjoint blocks $G_1, .., G_l$
- Constraint : Each $N_p$ should not take more than $\lambda_{pq}$ items from block $G_q$
- Any thoughts?

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

# Tutorial Question 4(c)

- The "envying" relation
- Player $i$ envies $i'$ if the utility of player $i$ from assigned item is strictly lower than the utility of player $i'$
- Need to write down constraints so that no player envies any other player
- Any thoughts?

# Tutorial Question 4(c)

- The "envying" relation
- Player $i$ envies $i'$ if the utility of player $i$ from assigned item is strictly lower than the utility of player $i'$
- Need to write down constraints so that no player envies any other player
- Any thoughts?
- Main idea : Each person has utility defined for every item, by the means of a utility function; different people can have different utility for the same item, although no 2 people can be assigned the same item
- For 2 people $i,\ i'$, assume that they get items $j,\ j'$ respectively, the utility of person $i$ receiving item $j$ should be greater than or equal to utility of person $i$ receiving item $j'$

# Tutorial Question 4(c)

- Mathematically, this can be shown as :
  $\forall i, i' \in N, \forall g_j, g_{j'} \in G : (x_{i,j} \wedge x_{i',j'}) \rightarrow u_i(g_j) \geq u_i(g_{j'})$

# Thank you!

If you have any questions, please don't hesitate. Feel free to ask!
We are here to learn together!