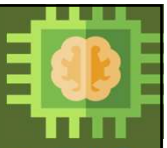
Elective Course
Course Code: CS4103
Autumn 2025-26



Lecture #16

Artificial Intelligence for Data Science

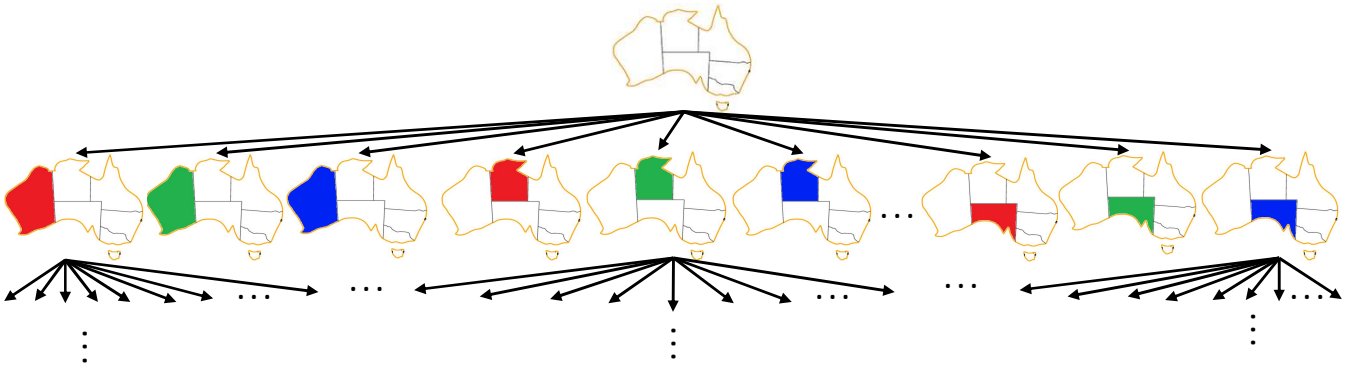
Week-4: CONSTRAINT SATISFACTION PROBLEM (CSP) [Part-II]
(Solving CSPs)

Course Instructor:
Dr. Monidipa Das
Assistant Professor
Department of Computational and Data Sciences
Indian Institute of Science Education and Research Kolkata, India 741246

Standard Search Formulation

- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - **Initial state:** the empty assignment, {}
  - **Successor function:** assign a value to an unassigned variable
  - **Goal test:** the current assignment is complete and satisfies all constraints
- We'll start with the straightforward, naïve approach, then improve it

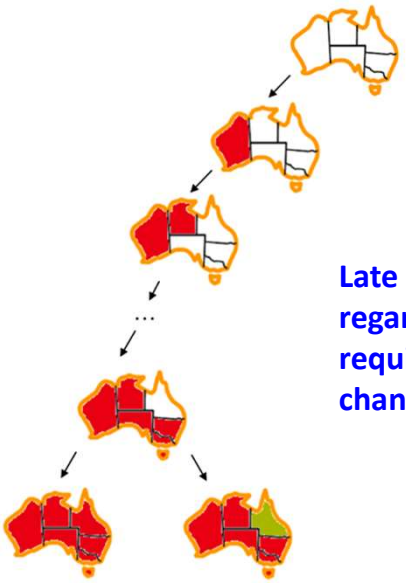
Breadth First Search



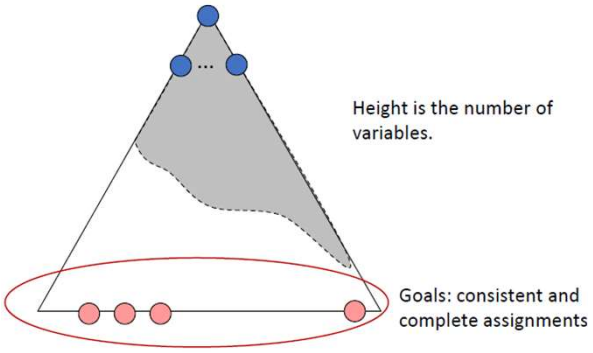
… All possible first variables
Check: Is there a solution?

We generate a tree with $n!\, d^n$ leaves!

Depth First Search



Late realization regarding requirement of change!



Backtracking Search

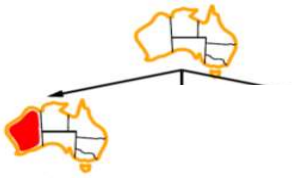
- Backtracking search is the basic uninformed algorithm for solving CSPs
- **Idea 1: One variable at a time**
  - Variable assignments are **commutative**, so fix ordering
  - i.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- **Idea 2: Check constraints as you go and backtrack when no legal values left to assign a variable**
  - i.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search*

Backtracking Search

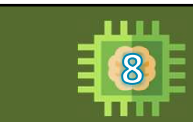
- Informally,
  - Pick a variable to assign.
  - Pick an assignment for the variable.
  - Check if all the constraints are satisfied.
  - If the constraints are not satisfied, then try a different assignment.
  - If no assignments left, need to back track.
  - If the assignment is complete, then we have a solution.
  - …….

Backtracking Search Example

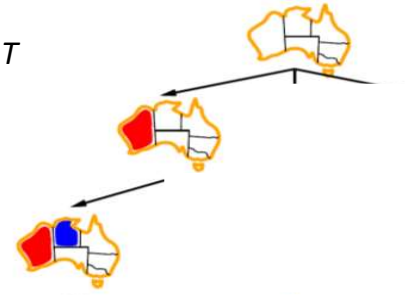
WA, NT, Q, SA, V, NSW, T



Backtracking Search Example

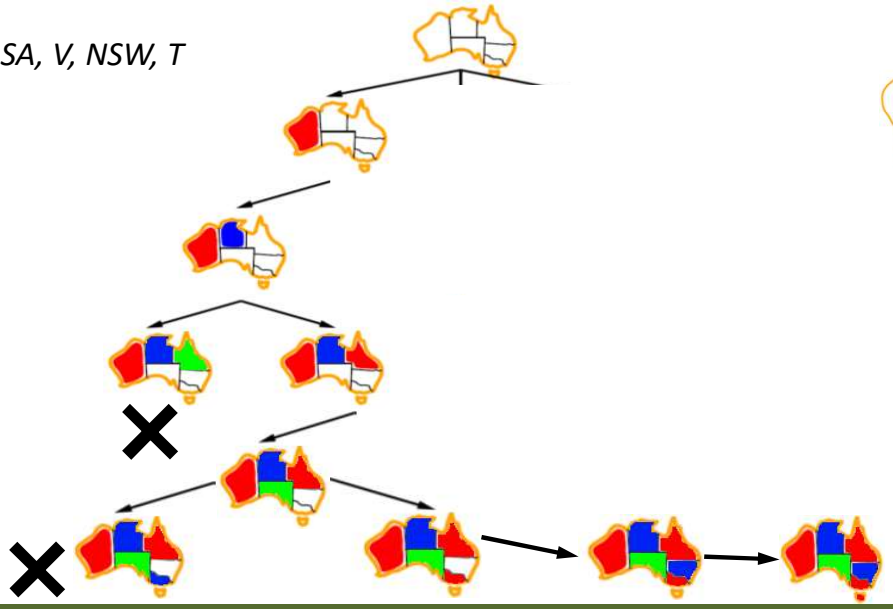


WA, NT, Q, SA, V, NSW, T



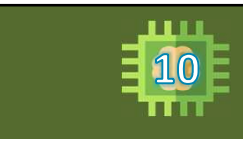
Backtracking Search Example

WA, NT, Q, SA, V, NSW, T



Goal state!

Backtracking Search Algorithm




```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

General Search checks consistency on full assignment

Backtracking Search Algorithm


```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

Backtracking Search checks consistency at each assignment

Backtracking Search Algorithm


```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

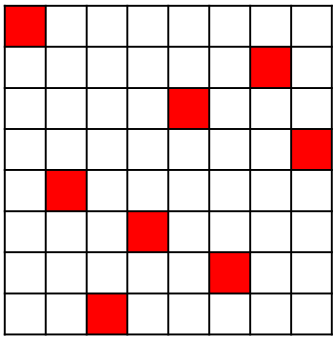
- Backtracking = DFS + variable-ordering + fail-on-violation

N-Queens Problem as CSP

- **Modeling:**
  - N variables Qi, one per row.
  - Value of Qi is the column the Queen in row i is placed; possible values {1, …, N}.

Example: (8-Queens)

Q1 = 1, Q2 = 7, Q3 = 5, Q4 = 8, Q5 = 2, Q6 = 4, Q7 = 6, Q8 = 3



- **This representation has N^N states:**
  - For 8-Queens: 8^8 = 16,777,216

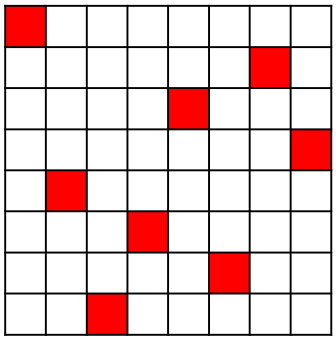
N-Queens Problem as CSP

OR

- **Modeling:**
  - N variables Qi, one per column.
  - Value of Qi is the row the Queen in column i is placed; possible values {1, …, N}.

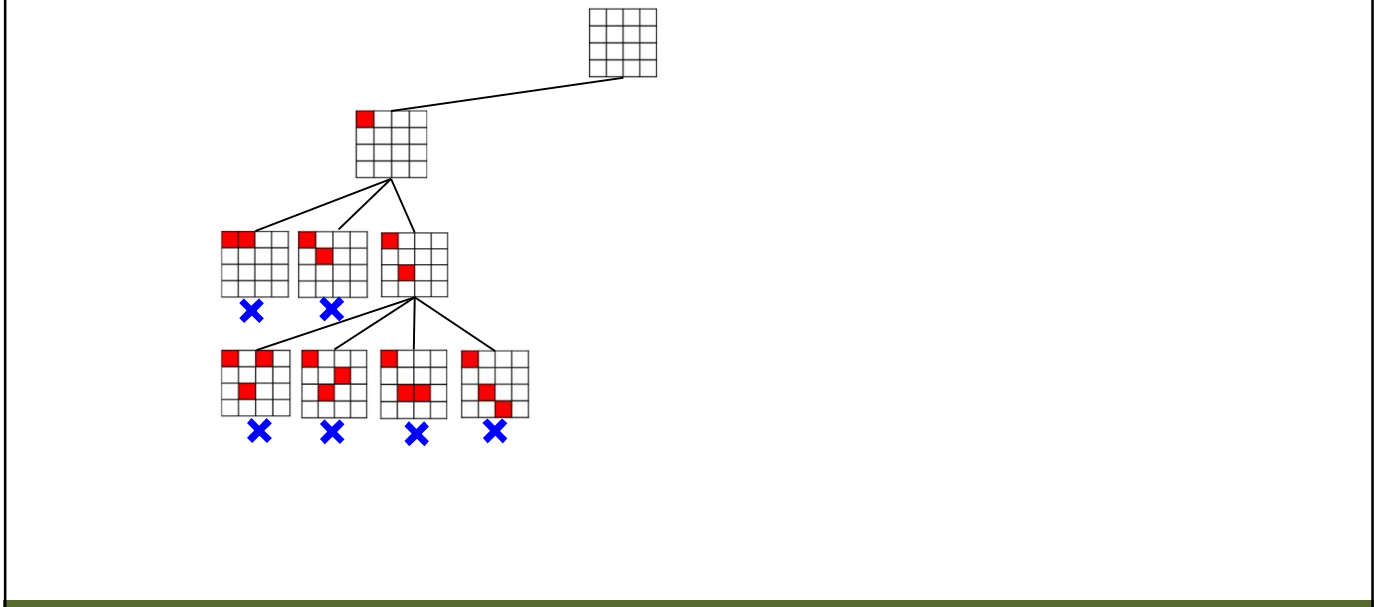
Example: (8-Queens)

Q1 = 1, Q2 = 5, Q3 = 8, Q4 = 6, Q5 = 3, Q6 = 7, Q7 = 2, Q8 = 4

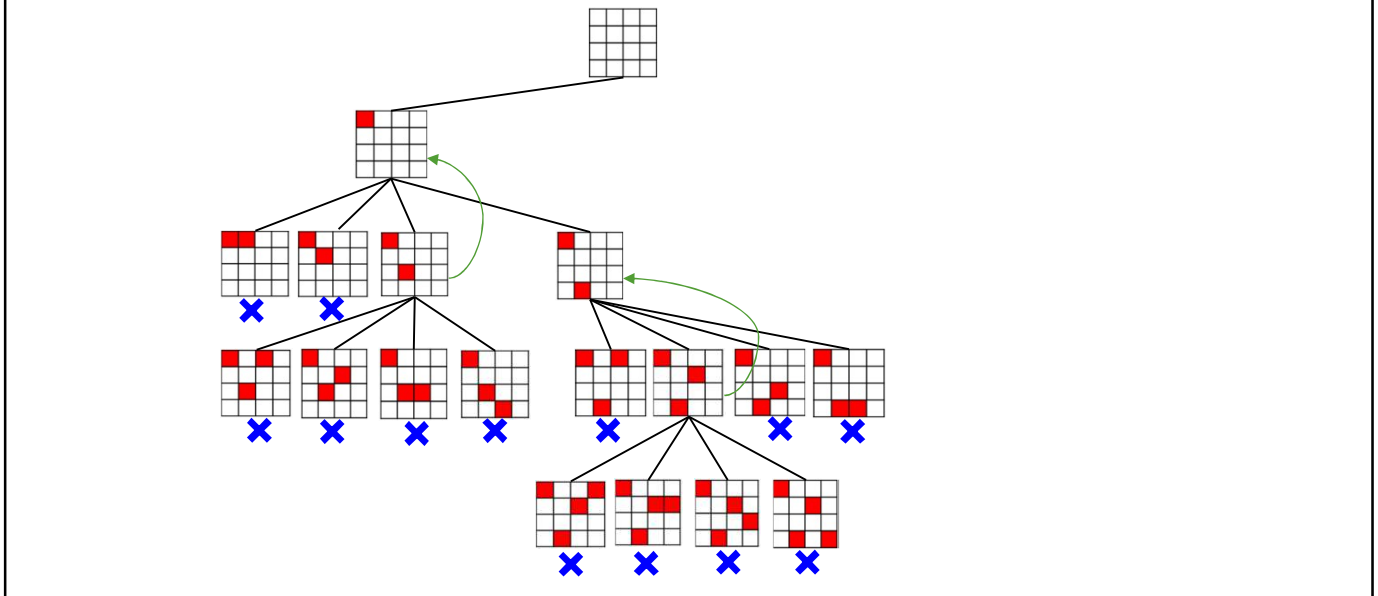


- **This representation has N^N states:**
  - For 8-Queens: 8^8 = 16,777,216

Backtracking Search Example: 4-Queens Problem

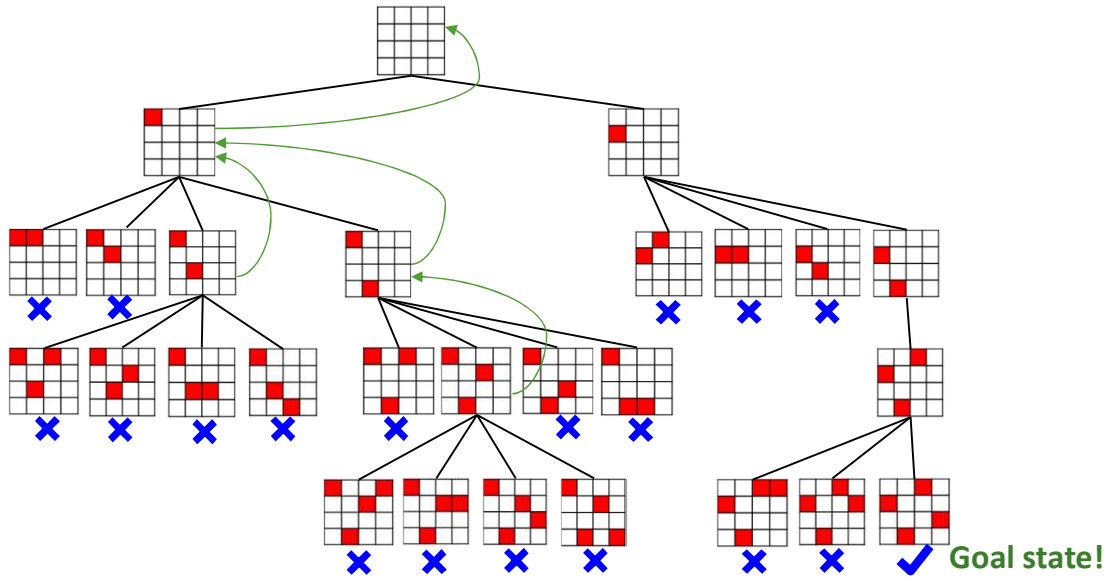


Backtracking Search Example: 4-Queens Problem



Backtracking Search Example: 4-Queens Problem

17



Solving CSPs: Improving Efficiency

18

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failures early?
- Can we take advantage of the problem structure?

Backtracking Search Algorithm

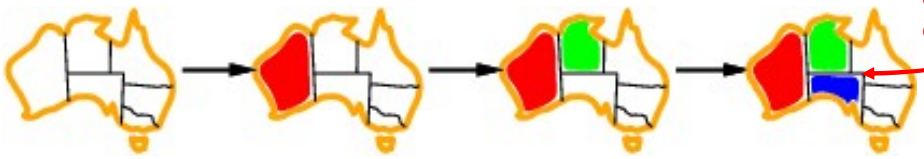

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```


Most constrained variable

- Most constrained variable:

  choose the variable with the fewest legal values

  a.k.a. minimum remaining values (MRV) heuristic



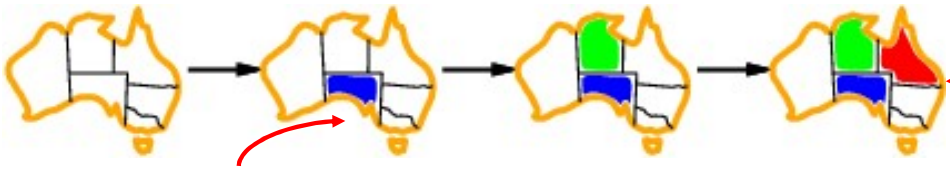
Why SA is chosen for assignment at this step?

- ***Try the variables likely to fail early rather than late.***
- ***Fail fast.***

Most constraining variable

- **Tie-breaker** among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables

a.k.a. Degree heuristic



Why Q is chosen for assignment at this step?

Why SA is chosen for assignment at this step?

This will reduce possible values for others. In effect, reduces branching factor.

Solving CSPs: Improving Efficiency

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failures early?
- Can we take advantage of the problem structure?

Backtracking Search Algorithm


```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```


Least constraining value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables

Leave maximum flexibility for subsequent assignments.

- Combining these heuristics makes 1000 queens feasible for N-queens problem

Solving CSPs: Improving Efficiency

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we **detect** inevitable failures early?
- Can we take advantage of the **problem structure**?

…to be discussed in the next class

Questions?