National University of Computer & Emerging Sciences

AL2002 – Artificial Intelligence – Lab (Spring 2025)

BSCS-6B

Lab Work 7 (Constraints Satisfaction Problem (CSP))

Lab Instructor	Momna Javaid
Department	Computer Science

Instructions:

Note: Carefully read the following instructions

1. You also have to submit .ipynb file.
2. Comments in the code explaining chunks of the code are important.
3. Plagiarism is strictly prohibited, 0 marks would be given to students who cheat.
4. First think about the problems statements then you may start your programming.
5. At the end when you done your tasks, attached .py or .ipynb files on google classroom.
6. Please submit your file in this format 22Fxxxx_Name_SecB_Lab#
7. Do not submit your assignment after deadline. Late and email submission is not accepted.

Lab Tasks:

Task 1:

You are given a 5×5 grid where you can move up, down, left, or right. The goal is to reach the target position from the start position using **Local Beam Search** with k=2 (i.e., track only two best positions at each step).

```
S . . . .
. # # . .
. . . . .
. # # . T
. . . . .
```

- S = Start position
- T = Target position
- # = Blocked cells (cannot pass through)
- . = Free cells

Steps to Follow:

1. Initialize: Start with two random valid positions near S.
2. Generate Successors: Move to adjacent valid (non-blocked) cells.
3. Evaluate Heuristic: Use Manhattan Distance to T (lower is better).
4. Select Best k Positions: Keep only the two best positions at each step.
5. Repeat Until Target is Reached.

Expected Output:

The algorithm should output a valid path from S to T avoiding obstacles.

Task 2:

A 9×9 Sudoku puzzle is a perfect example of a Constraint Satisfaction Problem (CSP). The objective is to fill the grid such that:

1. Each row contains the numbers 1-9 without repetition.
2. Each column contains the numbers 1-9 without repetition.
3. Each 3×3 subgrid contains the numbers 1-9 without repetition.

- Variables: Each empty cell in the 9×9 grid is a variable.
- Domain: The numbers {1,2,3,4,5,6,7,8,9}.
- Constraints:
    - No number appears twice in a row.
    - No number appears twice in a column.
    - No number appears twice in a 3×3 subgrid.

Implement these algorithms to solve the puzzle.
A simple **backtracking csp** algorithm assigns numbers and backtracks when an invalid assignment is found.
For **forward checking csp** before assigning a number, **prune invalid choices** from future cells.
Uses **arc consistency** to remove invalid numbers from the domain **before solving**.
Your task is to implement these algorithms and find the following:

- **Calculate time complexity.**
- **In AC3 display that Problem is a CSP Arc Consistent or not.**
- **Also display the solution, if exists.**

Pseudocode for Algorithms for your understanding:

Pseudocode for Backtracking CSP.

```
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
```

Pseudocode for Arc consistency:

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
    inputs: csp, a binary csp with components {X, D, C}
    local variables: queue, a queue of arcs initially the arcs in csp
    while queue is not empty do
        (Xi, Xj) ← REMOVE-FIRST(queue)
        if REVISE(csp, Xi, Xj) then
                if size of Di=0 then return false
                for each Xk in Xi.NEIGHBORS – {Xj} do
                add (Xi, Xj) to queue
    return true
function REVISE(csp, Xi, Xj) returns true iff we revise the domain of Xi
    revised ← false
    for each x in Di do
        if no value y in Di allows (x,y) to satisfy the constraints between Xi and Xj
        then delete x from Di
        revised ← true
    return revised
```