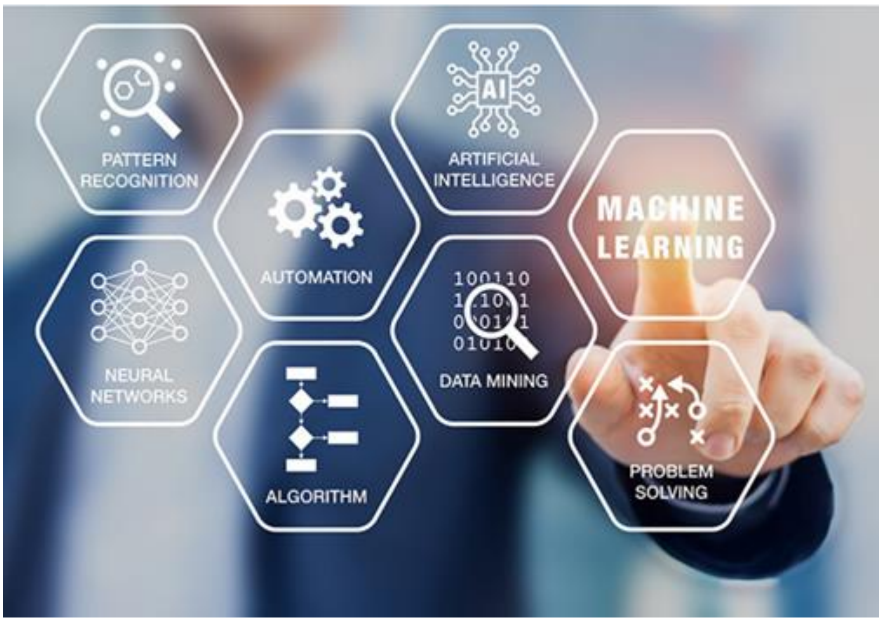
# National University of Computer & Emerging Sciences




# Lab Manual

# CS461: Artificial Intelligence Lab

| | |
|---|---|
| Course Instructor | Dr. Hafeez-Ur-Rehman |
| Lab Instructor | Muhammad Yousaf |
| Semester | Spring 2021 |

# Lab Manual # 10 (Local Searches)

## Outline:

- Local Search Algorithms
- Hill Climbing
    - Stochastic
    - First Choice
    - Random-Restart
- Simulated Annealing
- Local Beam Search
    - Stochastic
- Tasks

# History

We have studied and seen that both the **informed** and **uninformed** search algorithms require two things;

- Finding a Goal node?
- And Path to the goal node as a *solution?*

But there are some problems where path to the goal state is irrelevant. For example, in the 8-queens problem in the book (see page 71), what matters is the final configuration of queens, not the order in which they are added.

# Local Search Algorithms

Problems, where the path to the goal does not matter, which operate on a single **current node** and generally move only to neighbours of that node.

**Local search algorithms** are useful for solving pure **optimization problems**, in which the aim is to find the best state/node according to an **objective function**.

# Applications of Local Search

Local search algorithms have many important applications in optimization problems such as integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecommunications network optimization, vehicle routing, and portfolio management.

# Hill Climbing Search

Given the initial state, at each step, move to a neighbour of higher value in hopes of getting to a solution having the highest possible value.

| |
|---|
| Hill climbing is sometimes called **greedy local search** because it grabs a good neighbour state without thinking ahead about where to go next. |

**Stochastic hill climbing:** chooses at random from among the uphill moves (neighbors); the probability of selection can vary with the steepness of the uphill move.
Drawback: It is a **complete** (guarantees solution) but inefficient.

**First-choice hill climbing** implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g., thousands) of successors.
Drawback: It always selects better, so can stuck in local maxima leading towards an **incomplete** solution.

__Random-restart hill climbing__ ("If at first you don't succeed, try, try again.")
For each iteration randomly generate initial state and apply hill-climbing search algorithm until a goal is found. It is trivially **complete** with **probability** approaching **1**, because it will eventually generate a goal state as the initial state.

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

# __Simulated Annealing__

It's stochastic hill climbing search algorithm with an extra schedule function.
**Idea**: escape local maxima/minima by allowing some "bad" moves. Then, gradually decrease the probability of bad moves by decreasing the temperature (T).

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps
   current ← MAKE-NODE(INITIAL-STATE[problem])
   for i ← 1 to ∞ do
      T ← schedule[i]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else with probability e^(ΔE/T), set current ← next
```

## Local Beam Search

In stochastic hill climbing, instead of picking only one neighbor **local beam search** algorithm keeps track of k states. Keeping more than one node in a memory at the same time.
Local beam search is an adaptation of **beam search**, which is a path-based algorithm.

1. Randomly generate **k** states/nodes.
2. At each step, all the successors of all k states are generated.
3. If any one is a goal, then algorithm returns Otherwise, it selects the **k best successors** from the complete list and repeat the process.

```
function BEAM-SEARCH( problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

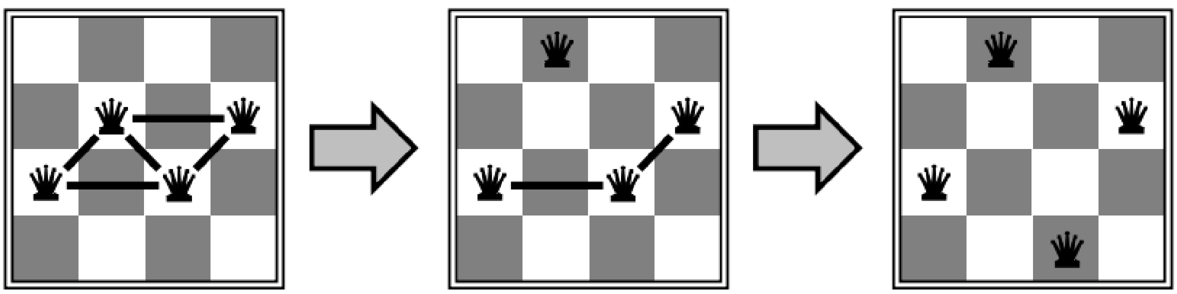
Drawback: often all **k** states end up on same local maxima/minima
Solution:
**Stochastic beam search**: choose **k successors randomly** in point (3), biased towards good ones. Close analogy to natural selection.

## N Queen's Problem

1. Put **n** queens on an **n × n** chessboard
2. No two queens on the same row, column, or diagonal

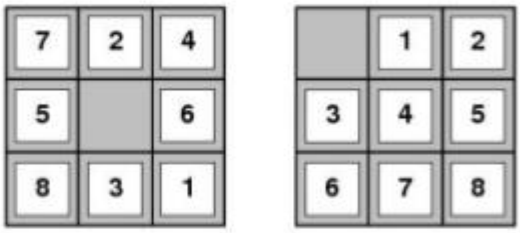
Solution: Only Move each queen vertically step-by-step for each column. Check if conflicts are resolved, otherwise repeat the process. **You can also apply one of the above local search algorithms.**

Note: The heuristic cost function "**h**" is the number of pairs of queens that are attacking each other, either directly or indirectly.



**h = 5** **h = 2** **h = 0**

# 8 Puzzle Problem




Start State    Goal State

- States?? Integer location of each tile
- Initial state?? Any state can be initial
- Actions?? {*Left*, *Right*, *Up*, *Down*}
- Goal test?? Check whether goal configuration is reached
- Path cost?? Number of actions to reach goal

## Tasks

1. Solve the N Queen's problem for n=8;
2. Solve the 8 Puzzle problem;

Apply stochastic hill climbing and local beam search algorithms on above problems. Find the solution and compare which one has lowest cost.