BITS Pilani
Pilani Campus

Artificial & Computational Intelligence

DSE CLZG557

M2 : Problem Solving Agent using Search

Raja vadhana P

Assistant Professor,

BITS - CSIS

Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing, Constraint Satisfaction Problem

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time, Reinforcement Learning

M7 AI Trends and Applications, Philosophical foundations

Module 2 : Problem Solving Agent using Search

A. Uninformed Search

B. Informed Search

C. Heuristic Functions

D. Local Search Algorithms & Optimization Problems

Problem Formulation

Learning Objective

1. Compare given heuristics for a problem and analyze which is the best fit
2. Design relaxed problem with appropriate heuristic design
3. Prove the designed relaxed problem heuristic is admissible
4. Identify the appropriate local search algorithm
5. Ability to design fitness and implement planning problems

Design of Heuristics

Heuristic Design

- **Effective Branching Factor**
- Good Heuristics
- Notion of Relaxed Problems
- Generating Admissible Heuristics

Effective branching factor (b^*):

If the algorithm generates N number of nodes and the solution is found at depth d, then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + \dots + (b^*)^d$$

Heuristic Design

- Effective Branching Factor
- Good Heuristics
- **Notion of Relaxed Problems**
- Generating Admissible Heuristics

Simplify the problem

Assume no constraints

Cost of optimal solution to relaxed problem ≤ Cost of optimal solution for real problem

Heuristic Design

- Effective Branching Factor
- Good Heuristics
- Notion of Relaxed Problems
- **Generating Admissible Heuristics**

Derive admissible heuristic from exact cost of a solution to a relaxed version of problem

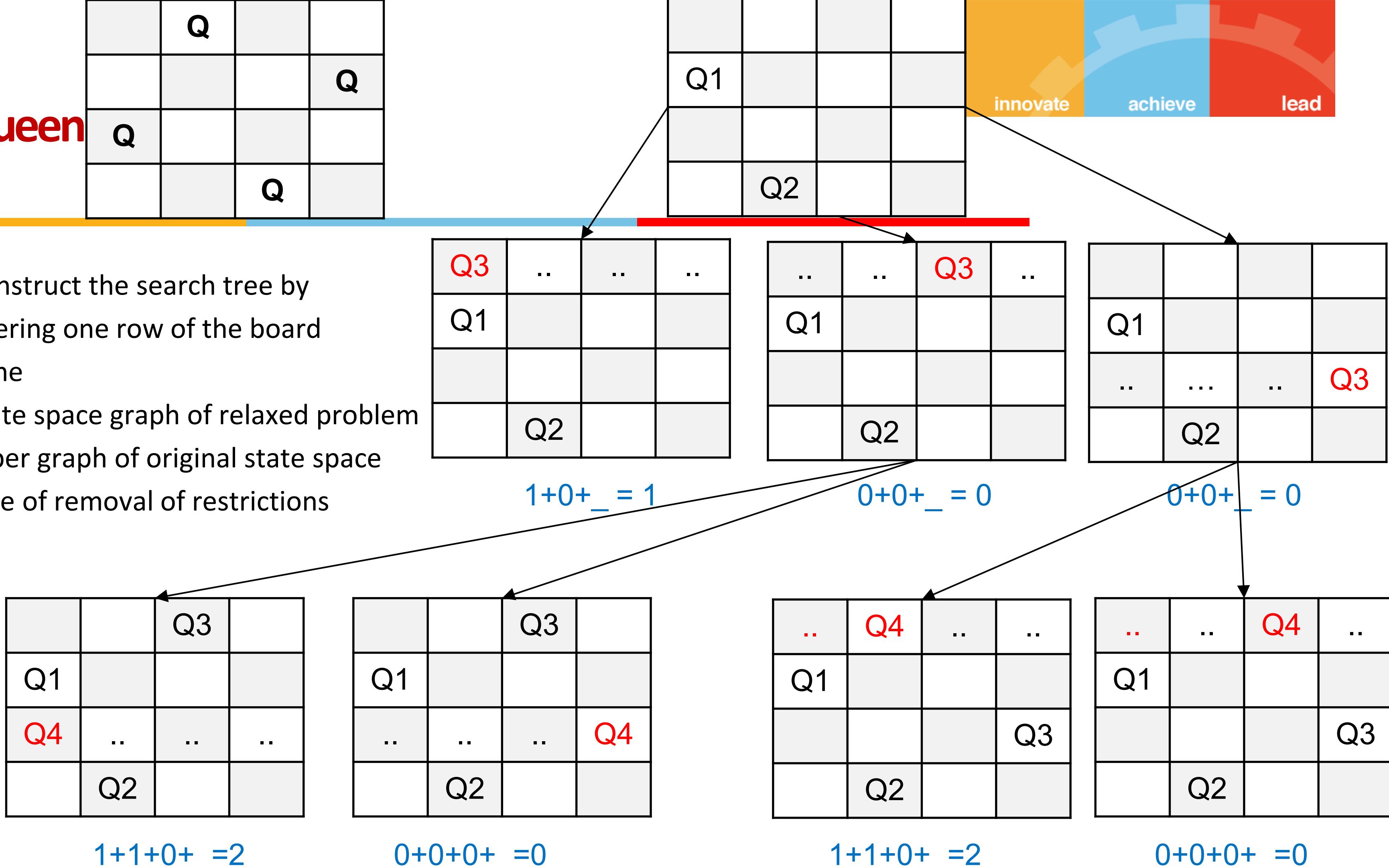
Rule of Dominance : If h2(n)>=h1(n) for all n, then choose h2

Choose h(n) : max{ h1(n) , h2(n) ……… hi(n) }

Break the Tie : Combination of h1 & h2

Design of Heuristics

N-Queen

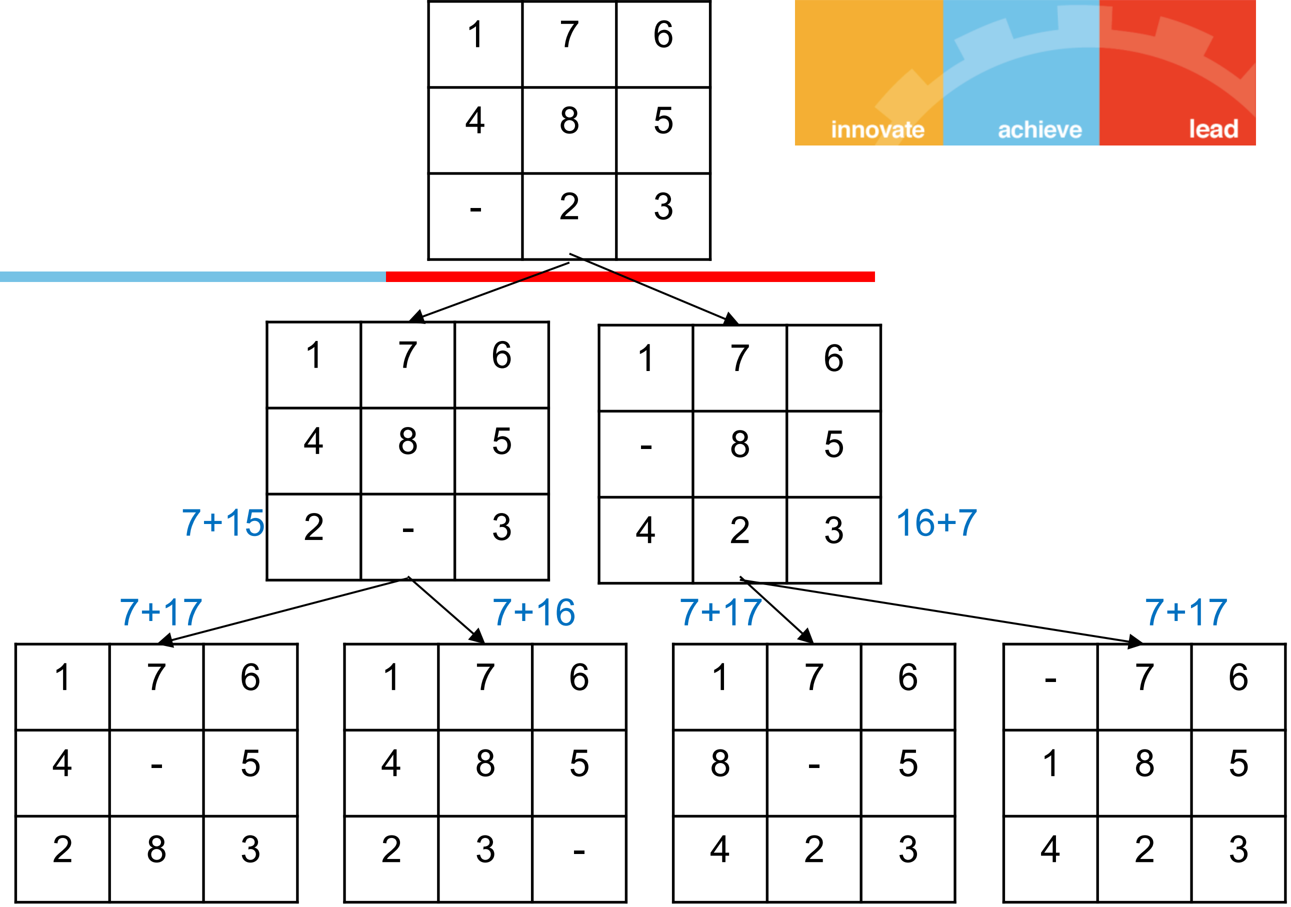


- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
< Xi , Yi >	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	n!

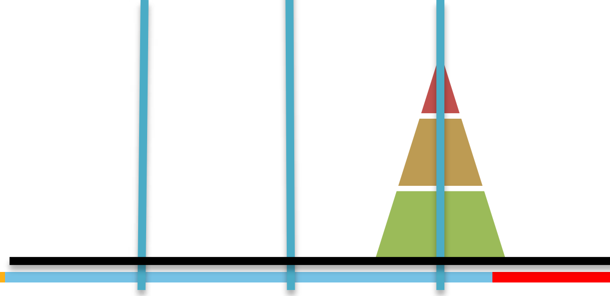
N-Tile

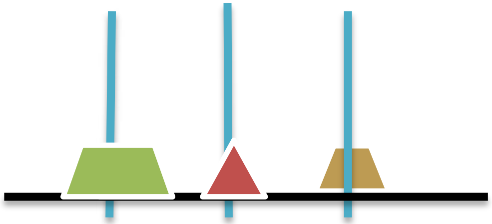
-	1	2
3	4	5
6	7	8

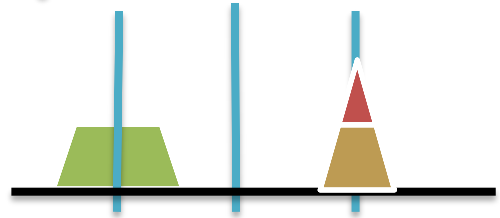


Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
<LOC, ID>	Move Empty to near by Tile		ID=LOC+1	Transition + Positional + Distance+ Other approaches	9!

Tower of Hanoi





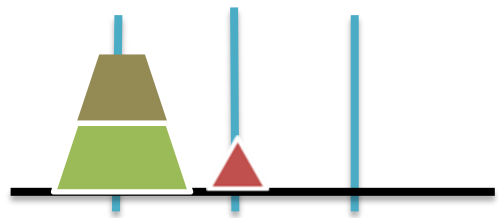


H1: Number of incorrect Discs in Destination

H2: Sum Number on the Largest Disc in Source

H3: Frontier's Legal Movement

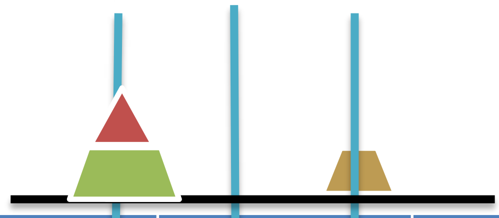
H4: Goal Orientation



<u>Transition Model Idea:</u>

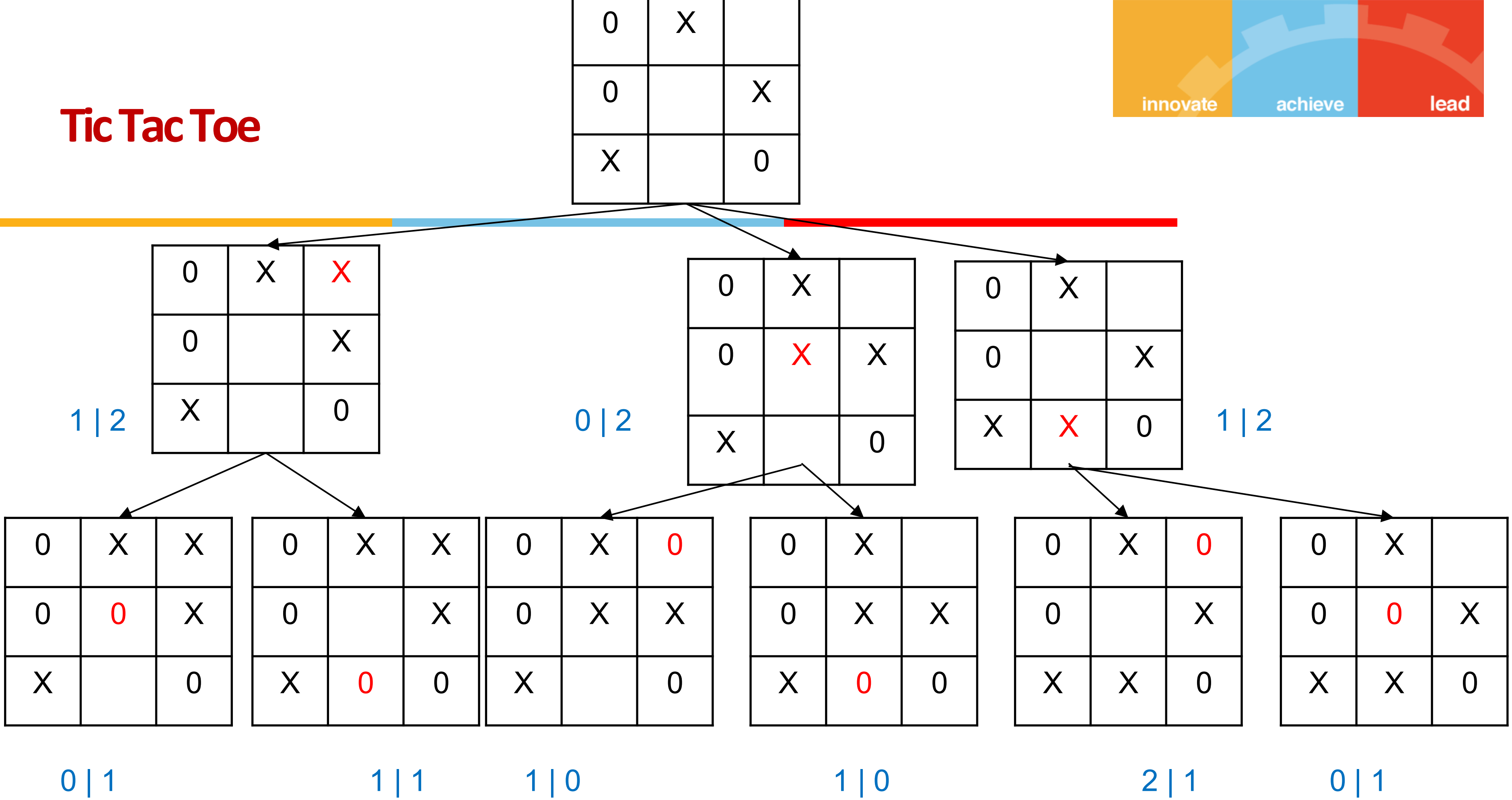
ODD : M – R – L

EVEN : R – M – L



Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
([n], [n-1], [1])	➢ Move X on-top Y : X<Y ➢ Move X on empty Peg		([], [], [1, 2,… n-1, n])	Optimal : Tested: $\mathbf{2^n-1}$ **No.of.Steps + Direction Cost + Positional Cost**	

Tic Tac Toe



Opposite Win | Player Win

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
([Xij], [Yij])	Place a coin in unoccupied (i,j)		N : i's N : j's N : i=j	No.of.Steps + Opp.Win + (N-1-Curr.Win)	$19{,}683=3^9$

Learn from experience

Trail / Puzzle	X1(n) : No.of.Misplaced Tiles	X2(n): Pair of adjacent tiles that are not in goal	X3(n): Position of the empty tile	………..h`(n)
Example 1	7	10	7	…………
Example 2	5	6	6	…………
………	..	..	..	…………

Create a suitable model:
h(n) = c1*X1(n) + c2*X2(n) + ……….

-	1	2
3	4	5
6	7	8

1	7	6
4	8	5
-	2	3

Pattern Database

innovate achieve lead

Divide and Conquer

Formulate Disjoint sub problem	Subset of Tiles 1,3,5 Subset of Tiles 6,7,8
	Compute a DB of tables
Compute Optimal Cost of this	Eg.,f(n) = 10
	Store Problem : Solution Cost ****Do not compute the cost of tiles not in the set**
Repeat for multiple combination	L1:1, L4:3, L3:5
	L3:1, L5:3, L1:5
	L5:8, L2:7, L3:6
Map the solution to the original problem	Use BFS from GOAL
	Find best set of states of the same sub problem nearest to G
Lower Limit of h(n)	Find Maximum h(n) among set of states chosen for a particular partition set Add the h(n) for portioned best solution

-	1	2
3	4	5
6	7	8

1	7	6
4	8	5
-	2	3

Local Search & Optimization

Optimization Problem

Goal : Navigate through a state space for a given problem such that an optimal solution can be found

Objective : Minimize or Maximize the objective evaluation function value

Scope : Local

Objective Function : Fitness Value evaluates the goodness of current solution

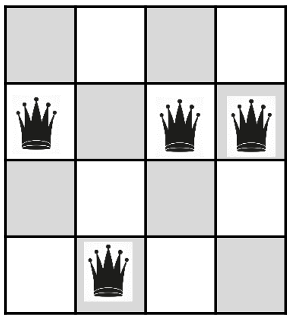
Local Search : Search in the state-space in the neighbourhood of current position until an optimal solution is found

Single Instance Based	Multiple Instance Based
Hill Climbing	Genetic Algorithm
Simulated Annealing	Particle Swarm Optimization
Local Beam Search	Ant Colony Optimization
Tabu Search	

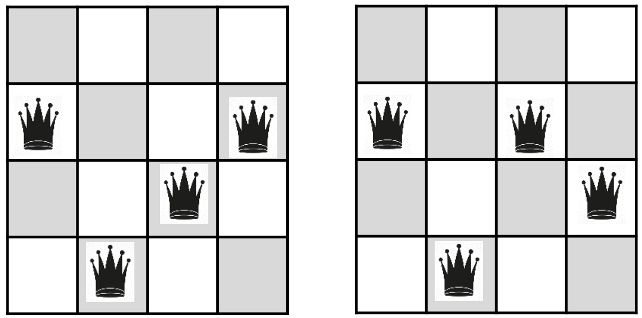
Local Search

Terminology

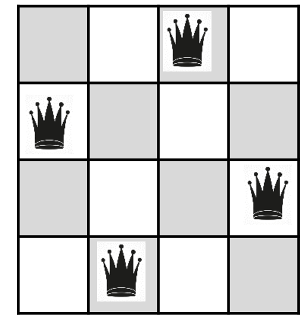
Local Search : Search in the state-space in the neighbourhood of current position until an optimal solution is found



Feasible State/Solution



Neighboring States



Optimal Solution

Fitness Value:

h(n) = 4 h(n) = 4 h(n) = 2 h(n) = 0

Above is an example of h(n) = No.of.Conflicting **pairs** of queens

h(n) = 0 h(n) = 0 h(n) = 1 h(n) = 0

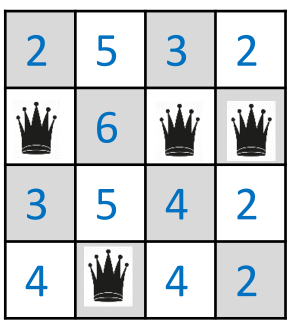
Above is an example of h(n) = No.of.Non-Conflicting Single queens with other queens in the board.

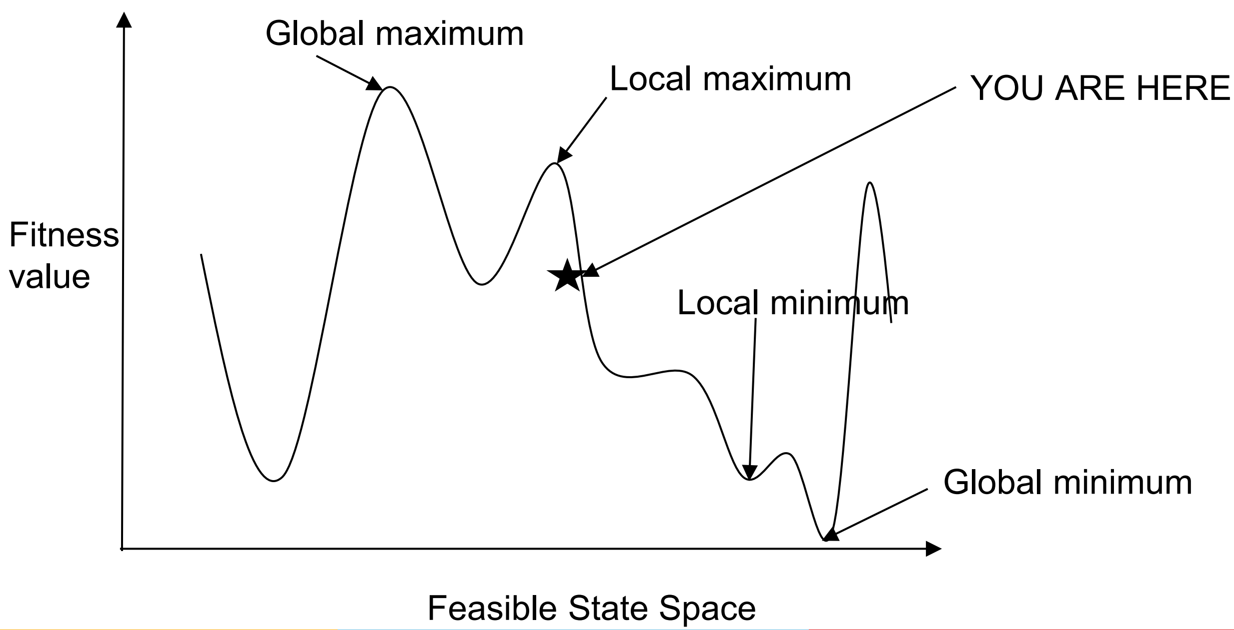
Terminology

Local Search : Search in the state-space in the neighbourhood of current position until an optimal solution is found

Algorithms:

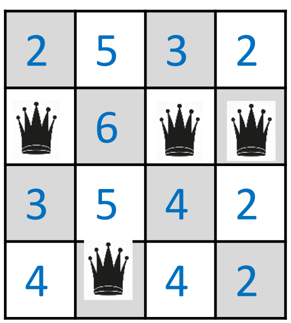
- Choice of Neighbor
- Looping Condition
- Termination Condition

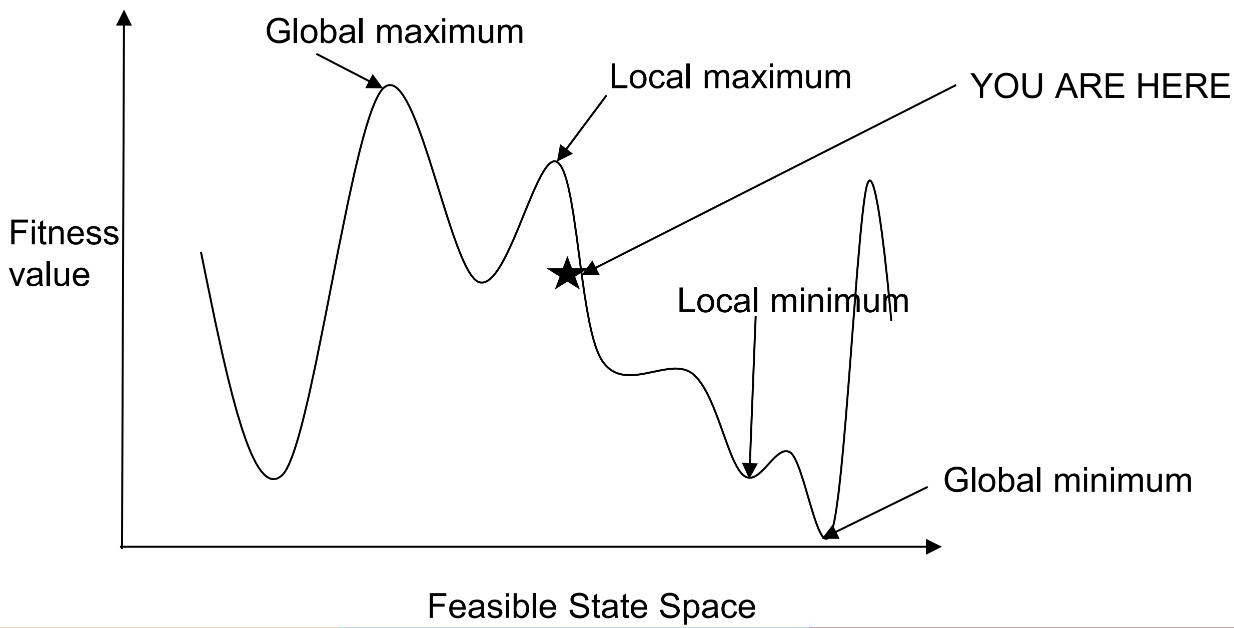




Hill Climbing

Hill Climbing



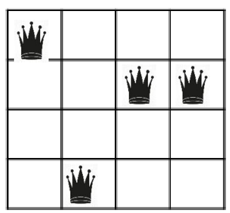


Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

h(n) = No.of non-conflicting pairs of queens in the board.

Q1-Q2
Q1-Q3
Q1-Q4
Q2-Q3
Q2-Q4
Q3-Q4



1	4	2	2	4

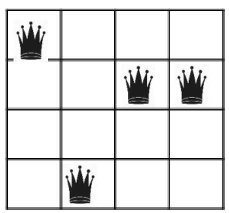
Note : Steps 3 & 4 in the above algorithm will be a part of variation of Hill climbing

Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

h(n) = No.of non-conflicting pairs of queens in the board.

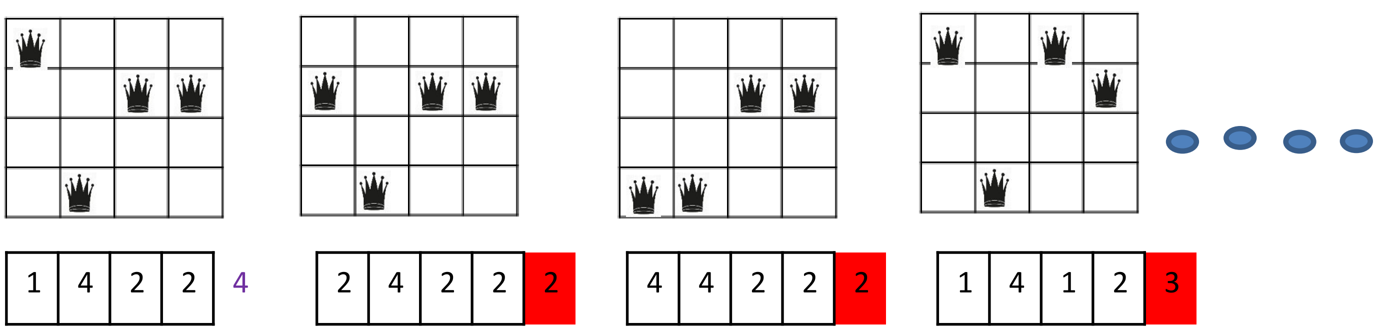
Q1-Q2
Q1-Q3
Q1-Q4
Q2-Q3
Q2-Q4
Q3-Q4



1	4	2	2	4

Hill Climbing

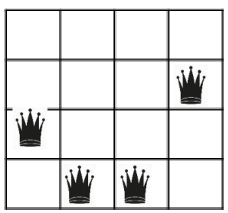
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the best next state
5. Repeat from Step 2



Hill Climbing

Random Restart

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



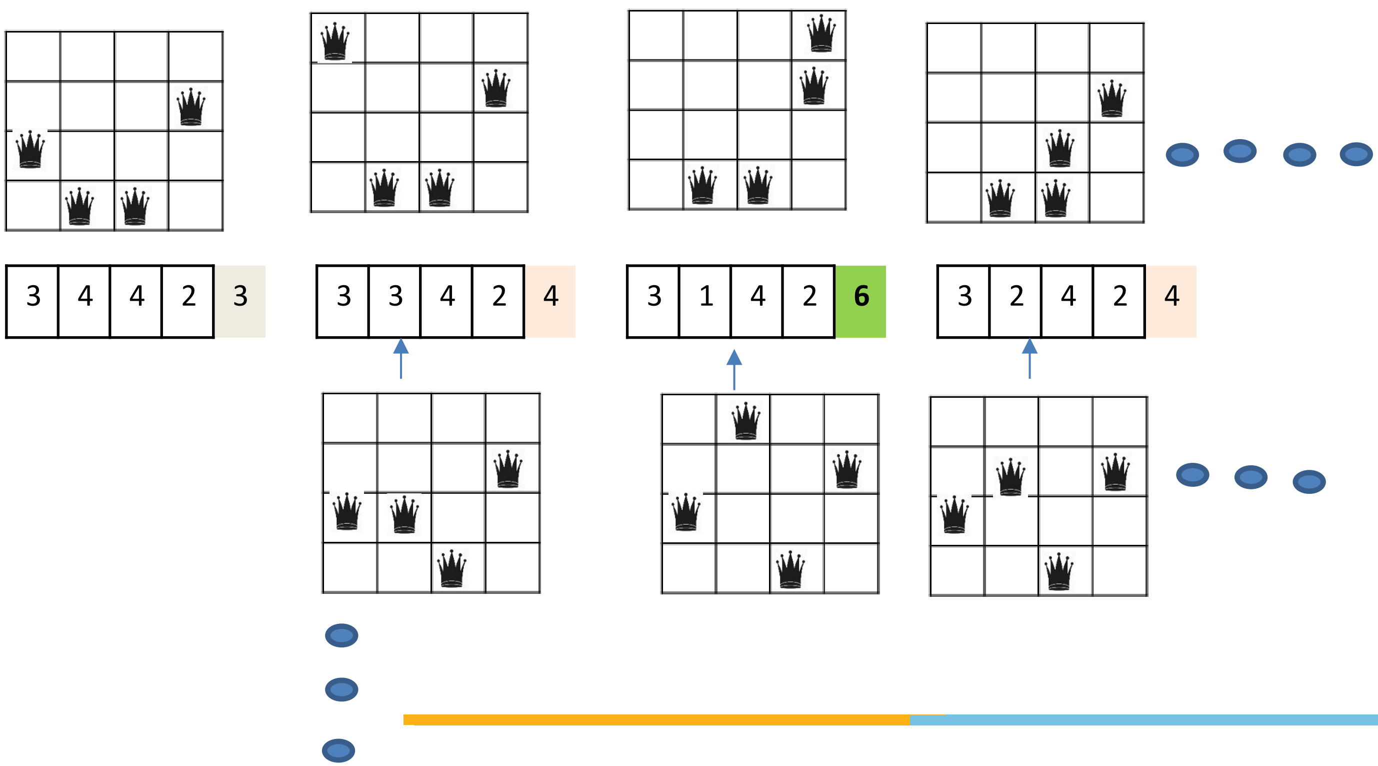
3	4	4	2	3

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of *current*
    if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
    current ← *neighbor*

Hill Climbing –Restart round =2

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



Required Reading: AIMA - Chapter #4.2 , #4.3 (will be continued in next class)

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials