Artificial & Computational Intelligence

AIML CLZG557

M2 : Problem Solving Agent using Search

Dr. Sudheer Reddy

BITS Pilani
Pilani Campus

Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

Module 2 : Problem Solving Agent using Search

A. Uninformed Search

B. Informed Search

C. Heuristic Functions

D. Local Search Algorithms & Optimization Problems

Learning Objective

At the end of this class , students should be able to:

1. Differentiate which local search is best suitable for given problem
2. Design fitness function for a problem
3. Construct a search tree
4. Apply appropriate local search and show the working of algorithm at least for first 2 iterations with atleast four next level successor generation(if search tree is large)
5. Design and show local search Algorithm steps for a given problem

Local Search & Optimization

Task Environment

Goal Formulation

Problem Formulation

Search Phase

Execution Phase

Phases of Solution Search by PSA

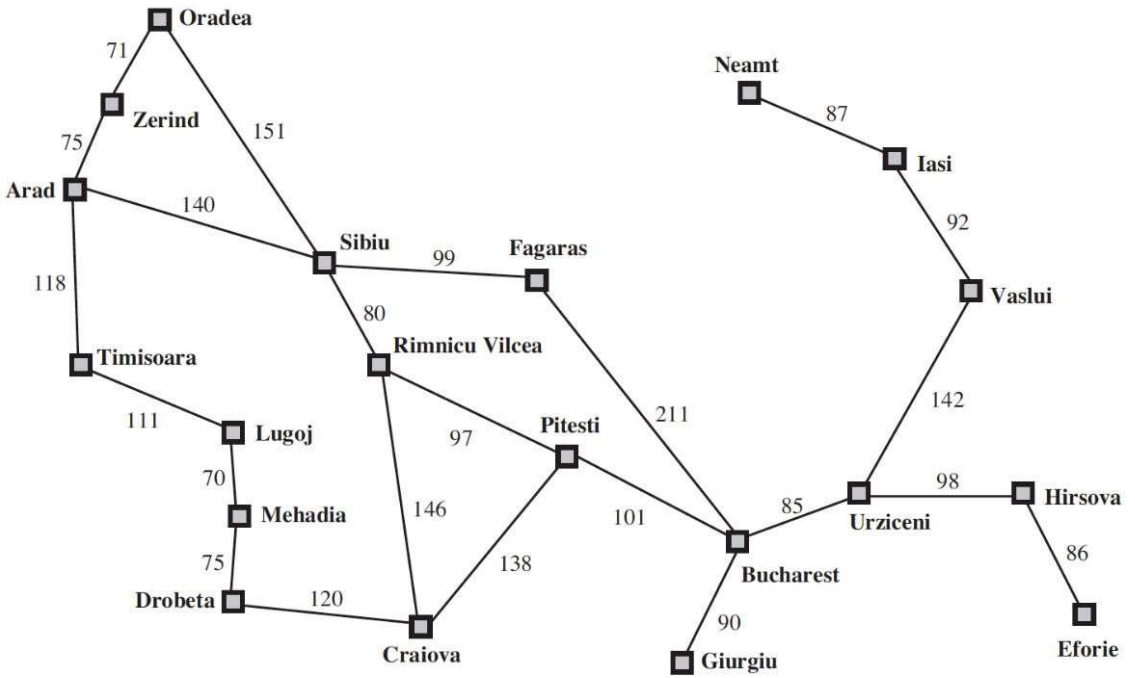
Assumptions – Environment :

Static

Observable

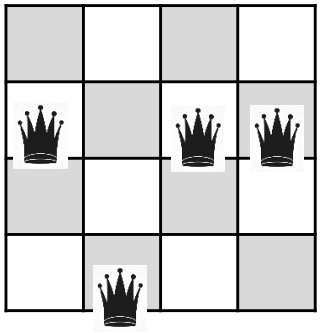
Discrete

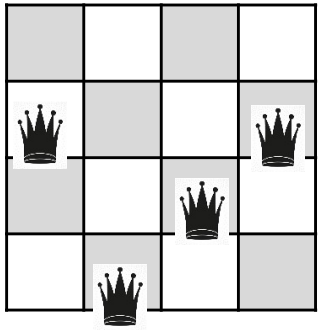
Deterministic

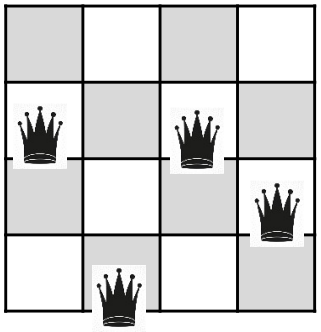


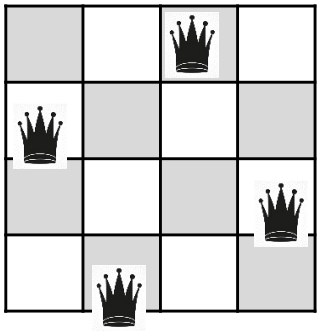
Terminology

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

h(n) = No.of.Conflicting **pairs** of queens

h(n) = 2 h(n) = 2 h(n) = 4 h(n) = 6

h(n) = No.of.**Non**-Conflicting **pairs** of queens.

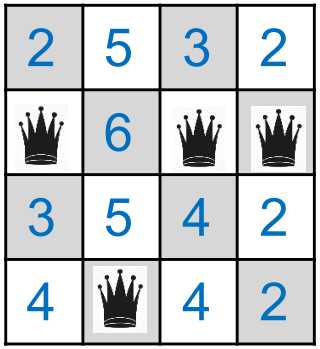
Local Search

Terminology

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

Algorithms:

- Choice of Neighbour
- Looping Condition
- Termination Condition



Local Search

innovate achieve lead

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	

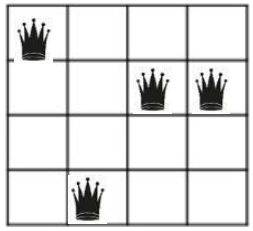
Hill Climbing

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

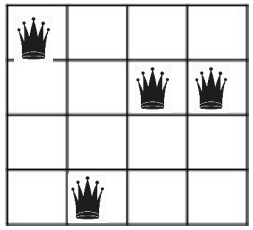
Note : Steps 3 & 4 in the above algorithm will be a part of variation of Hill climbing

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.

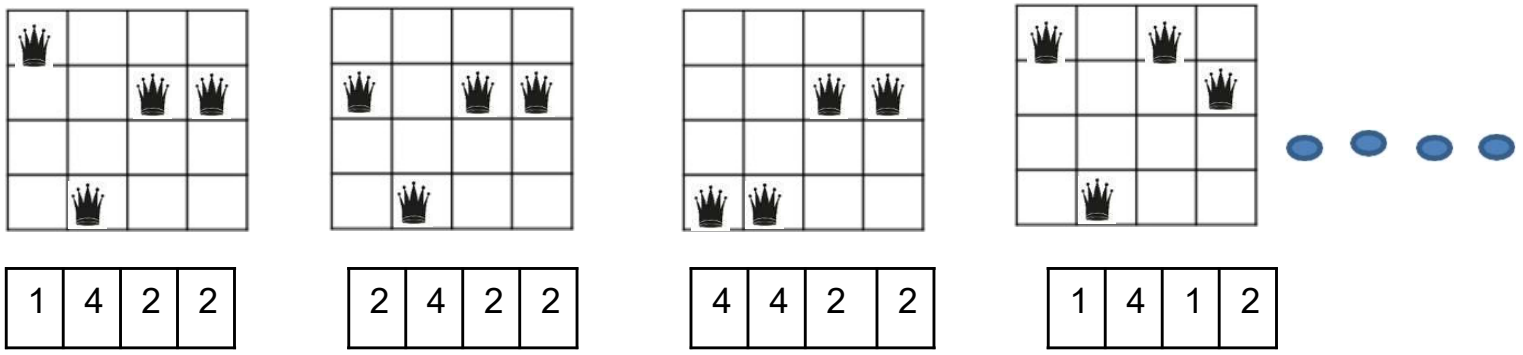
Q1-Q2
Q1-Q3
Q1-Q4
Q2-Q3
Q2-Q4
Q3-Q4

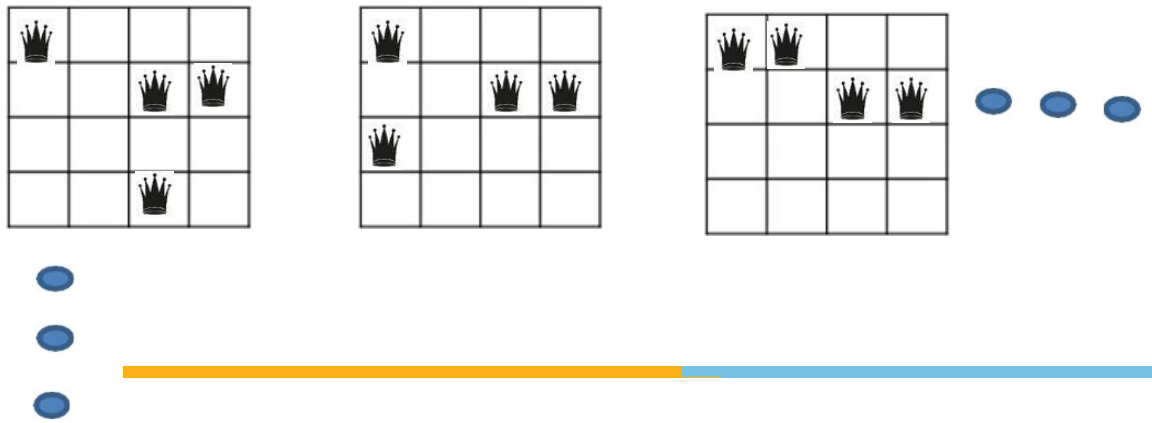


1	4	2	2	4

Hill Climbing

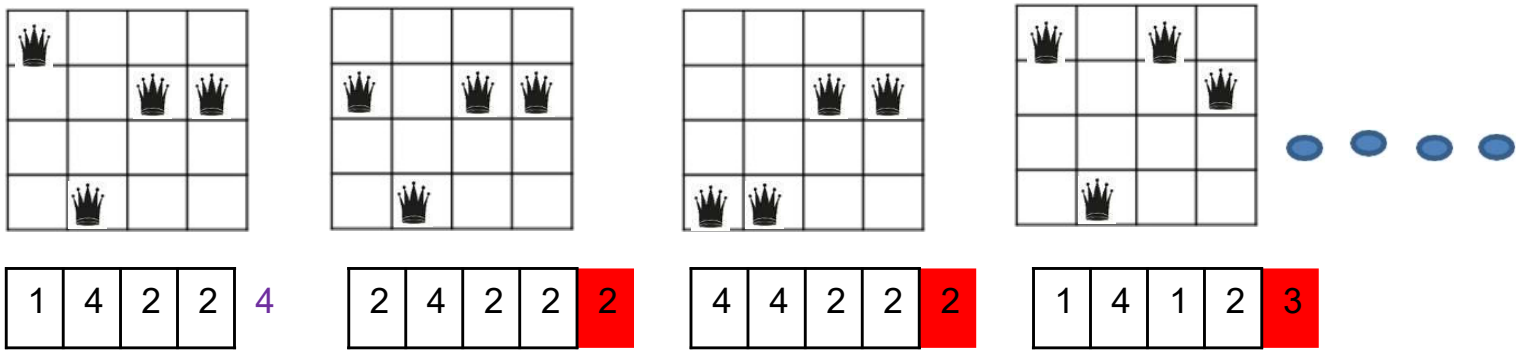
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



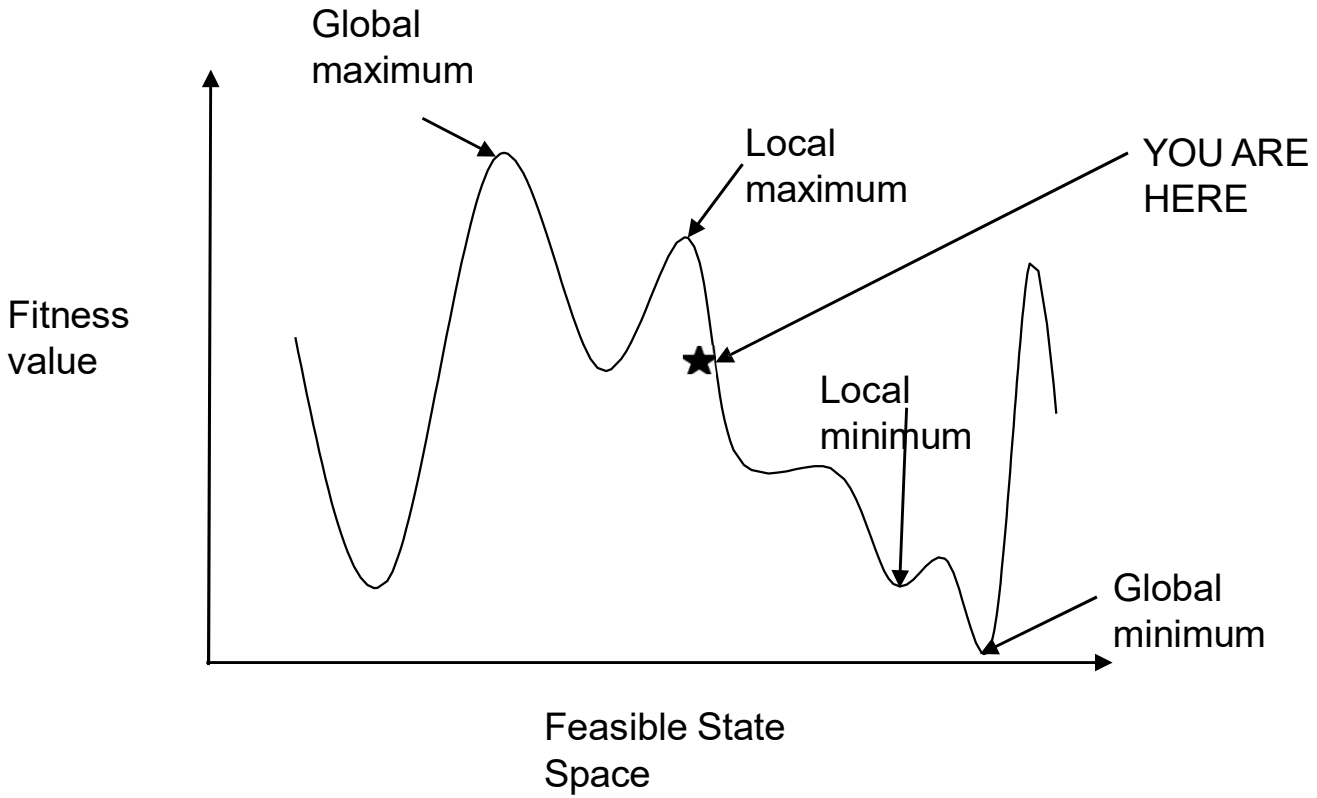


Hill Climbing

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



Hill Climbing

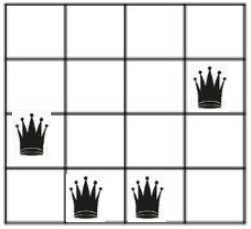


Hill Climbing

innovate achieve lead

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



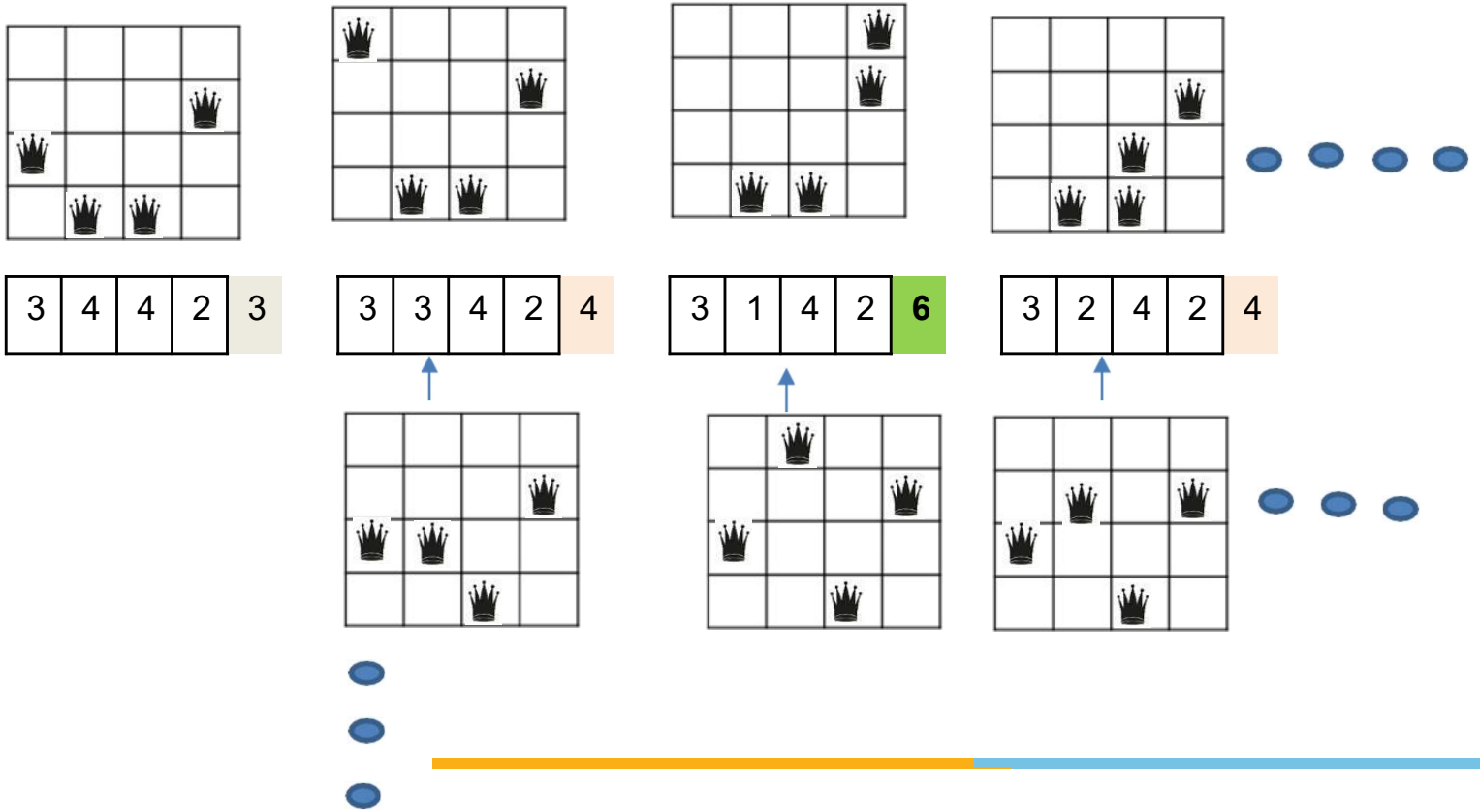
3	4	4	2	3

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

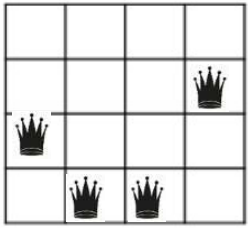
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



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



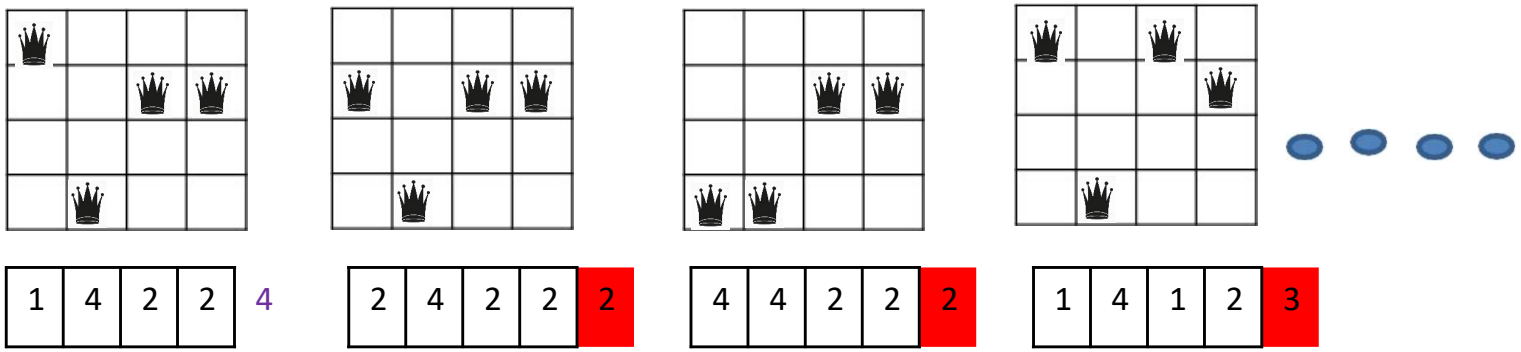
3	4	4	2	3

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*

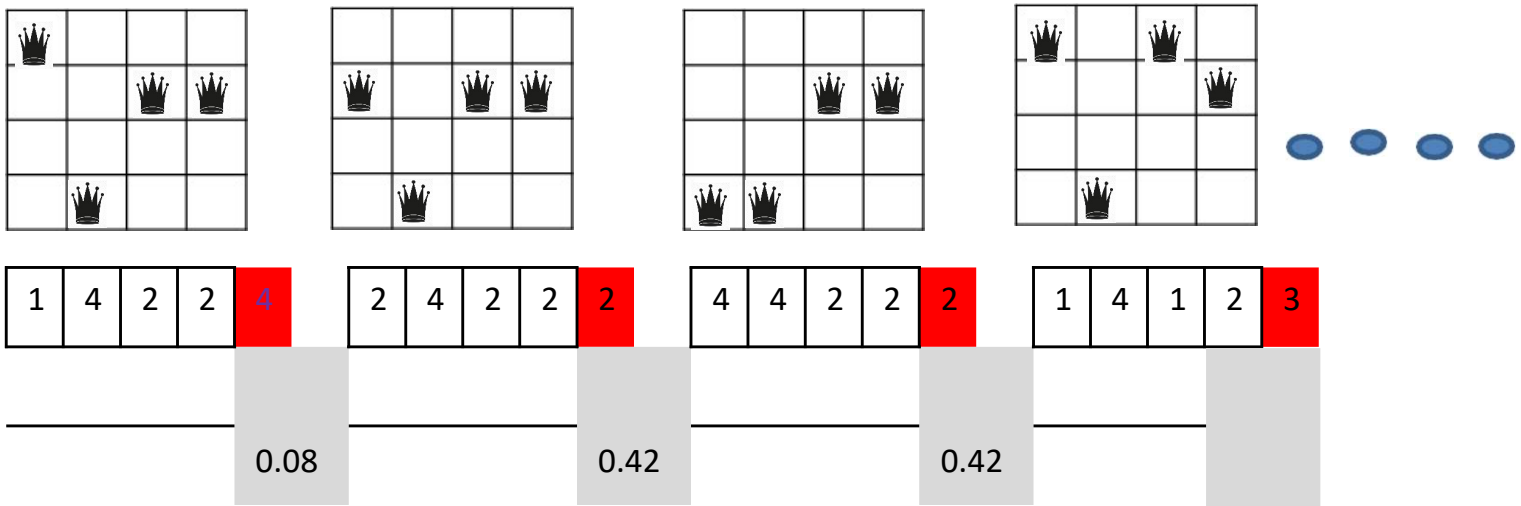
Stochastic Hill Climbing

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



Stochastic 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



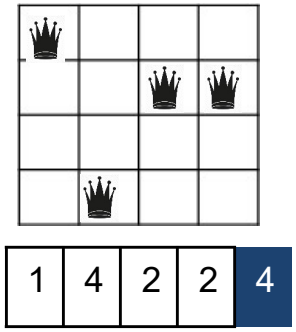
12 N = {4,2,2,3,3,2,2,0,2,1,3,0}

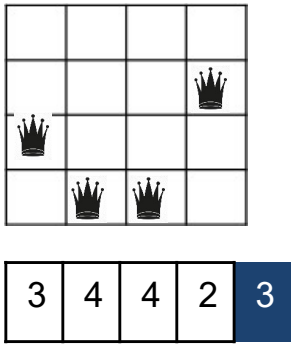
$next \leftarrow$ a randomly selected successor of $current$
$\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
if $\Delta E > 0$ **then** $current \leftarrow next$
else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

Local Beam Search

Beam Search

1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next ‘k’ states randomly based on the probabilit
6. Repeat from Step 2

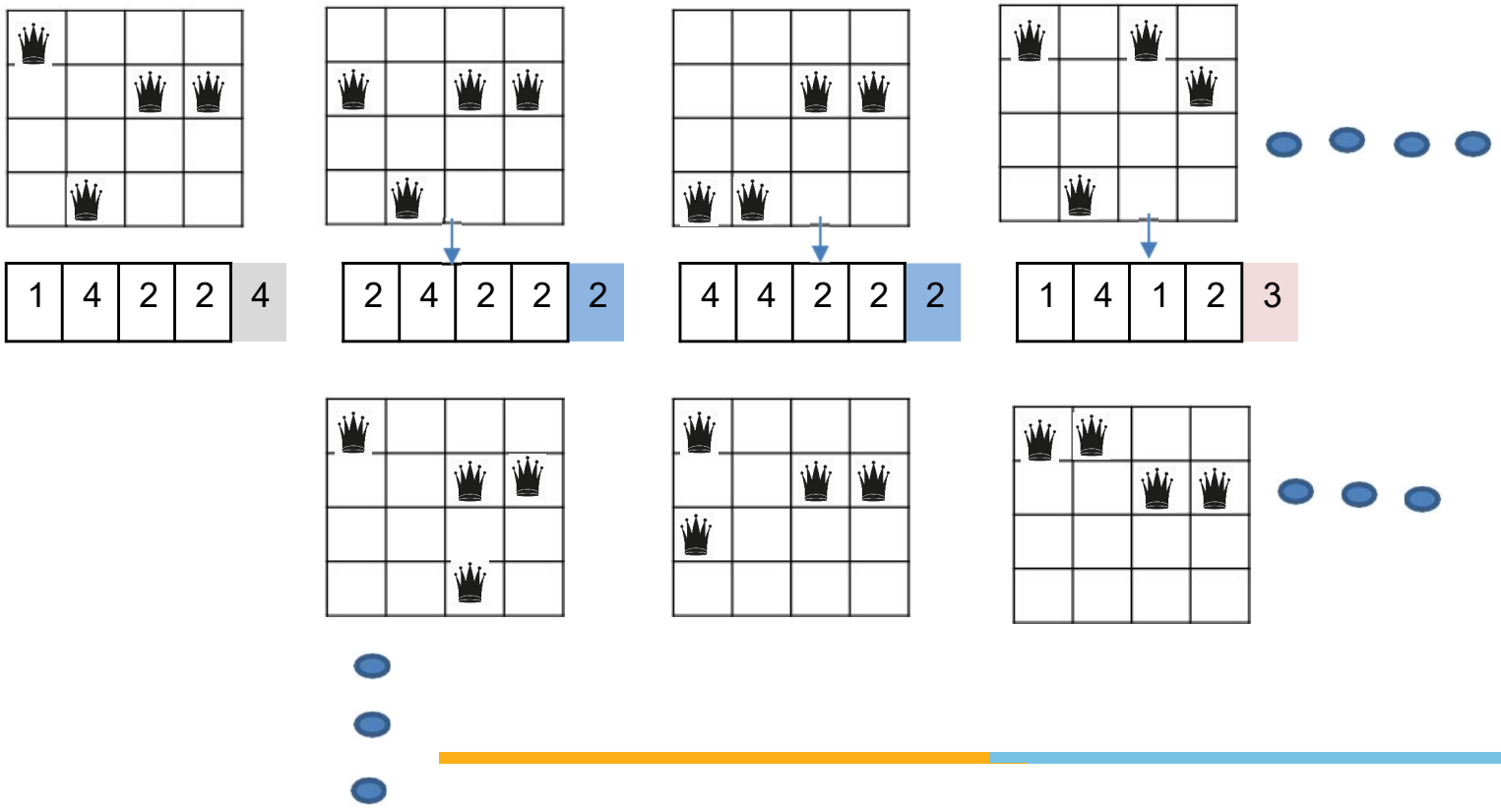




Beam Search

1st State

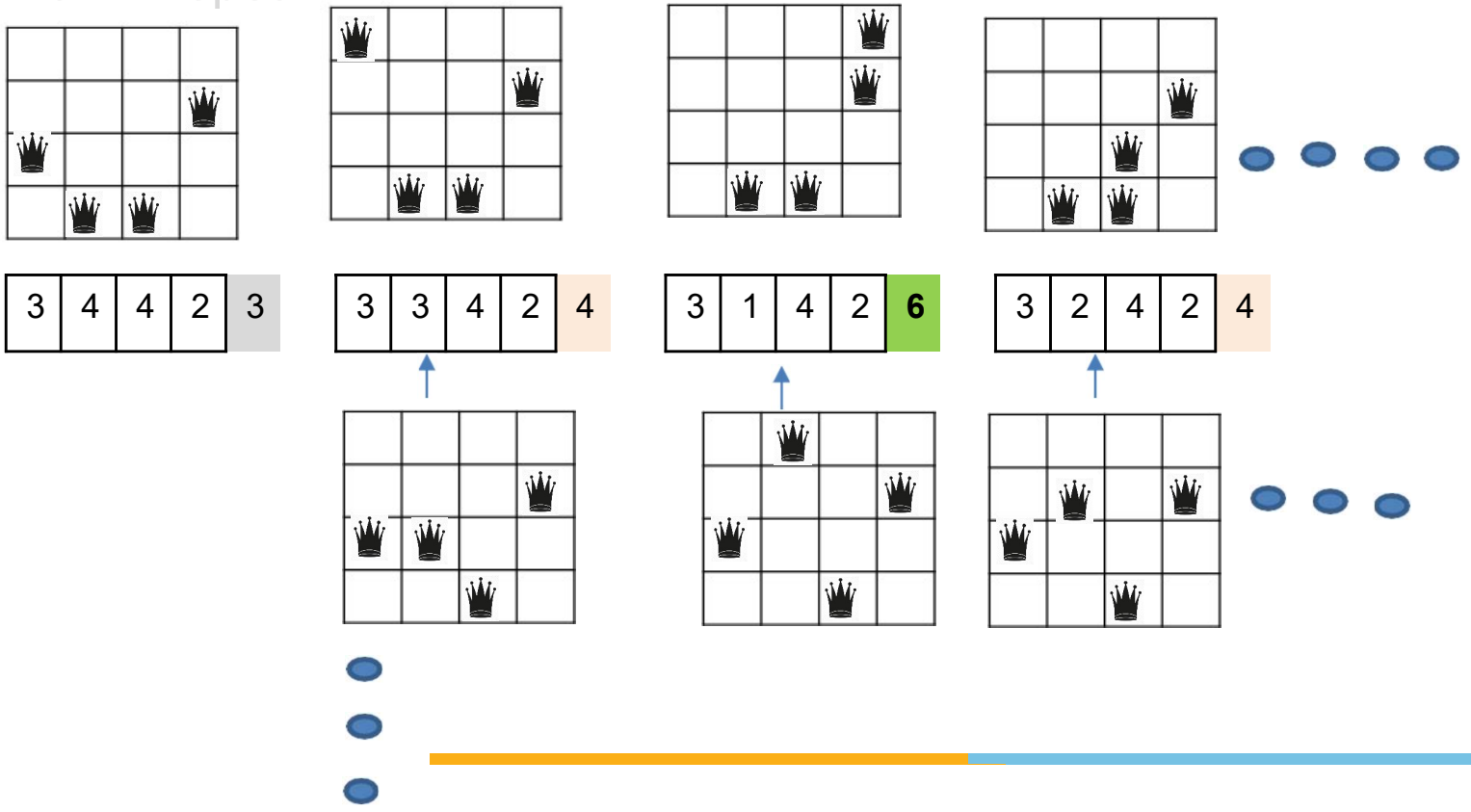
1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next ‘k’ states randomly based on the probability
6. Repeat from Step 2



Beam Search

2nd State

1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next ‘k’ states randomly based on the probability
6. Repeat from Step 2

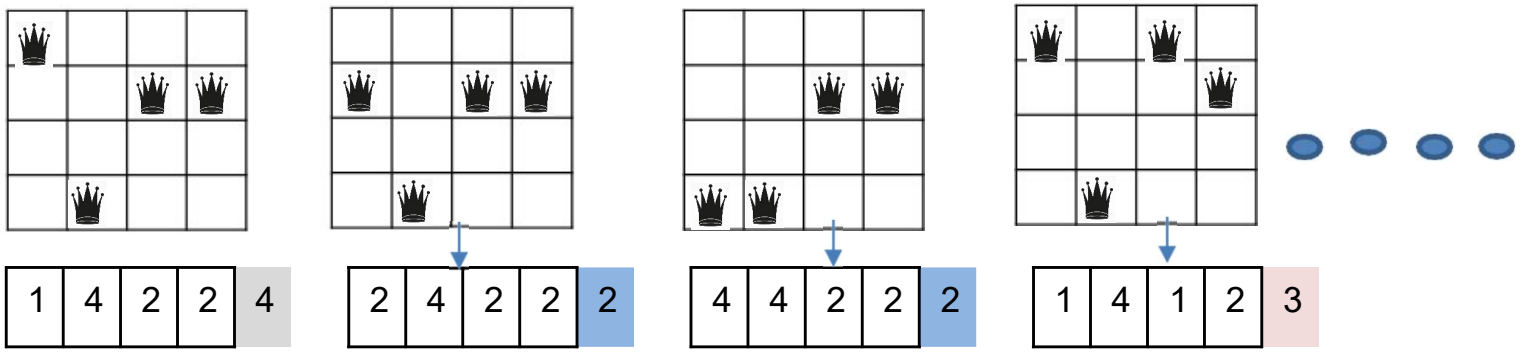


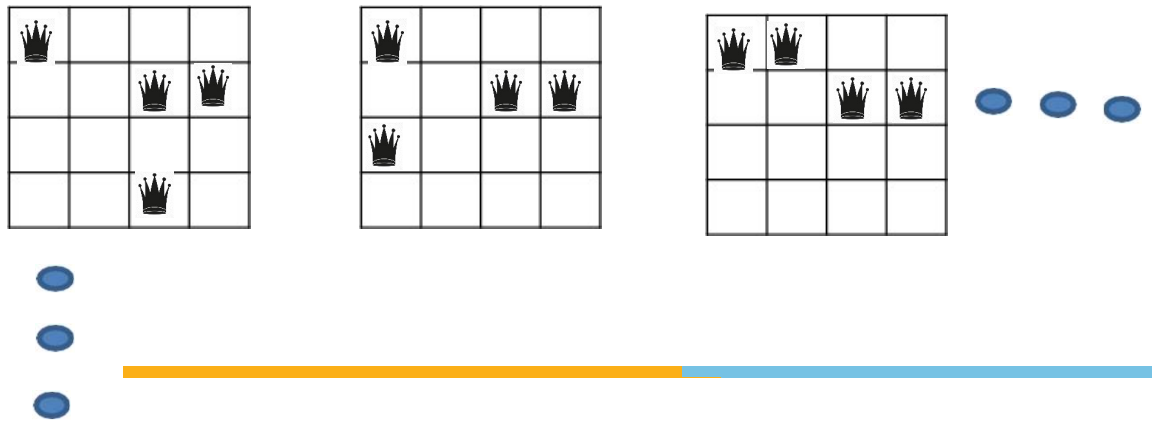
Stochastic Beam Search

innovate achieve lead

Sample from 1st State

1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next 'k' states randomly based on the probability
6. Repeat from Step 2

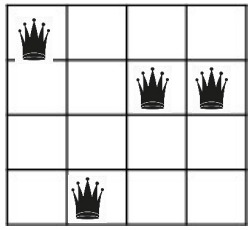




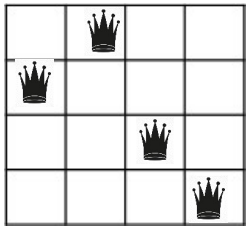
Genetic Algorithm

Genetic Algorithm

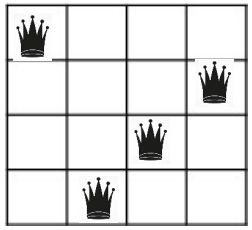
1. Select ‘k’ random states – **Initialization : k=4**
2. Evaluate the fitness value all states
3. If anyone of the state’s has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



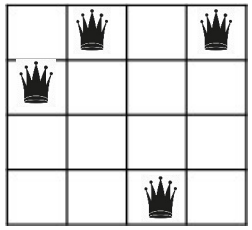
1	4	2	2



2	1	3	4



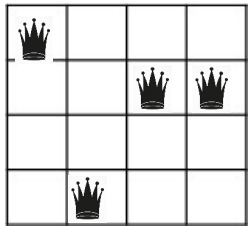
1	4	3	2



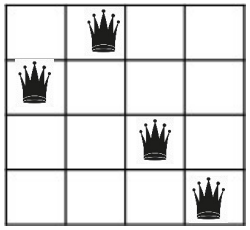
2	1	4	1

Genetic Algorithm

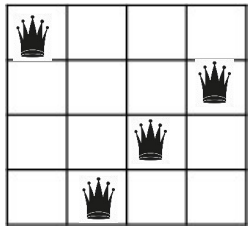
1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



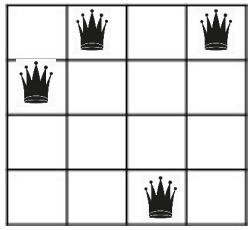
1	4	2	2	4



2	1	3	4	4



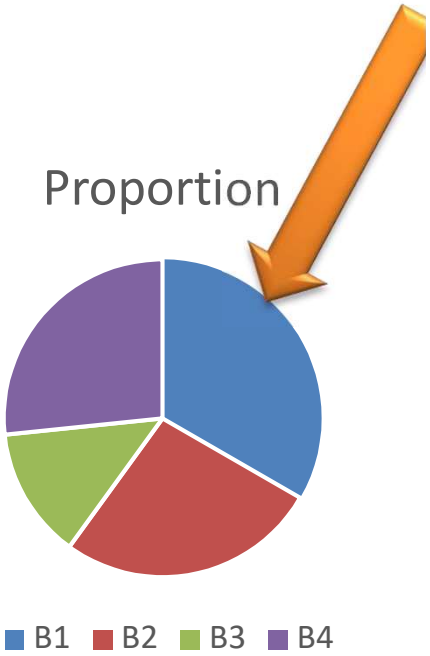
1	4	3	2	2

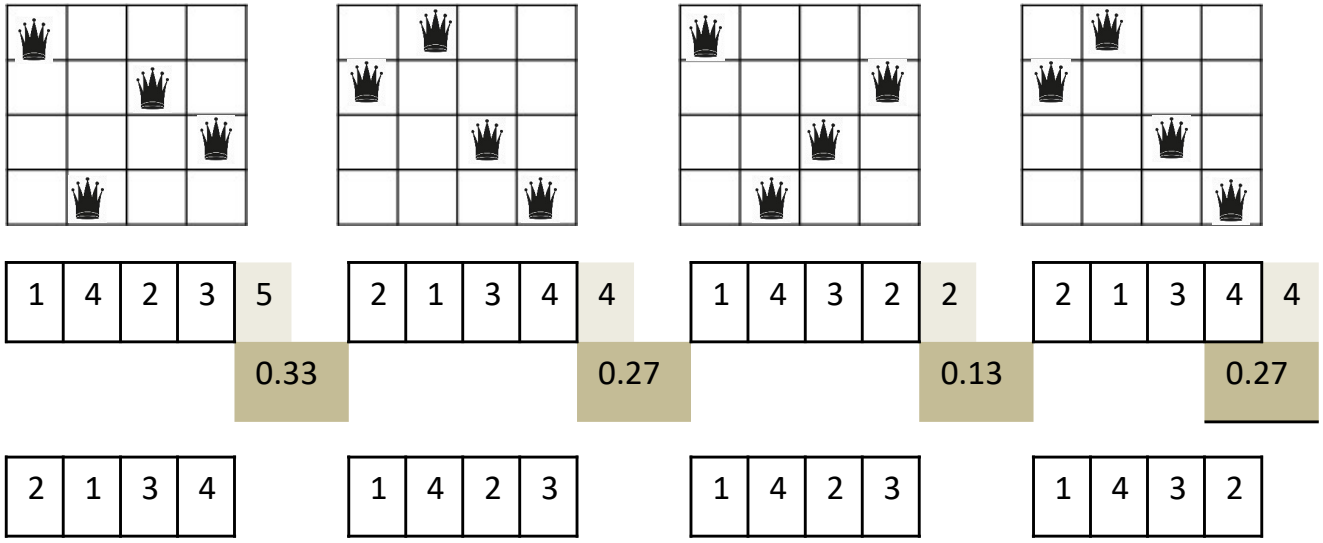


2	1	4	1	3

Genetic Algorithm

Eg., use roulette wheel mechanism to select pair/s

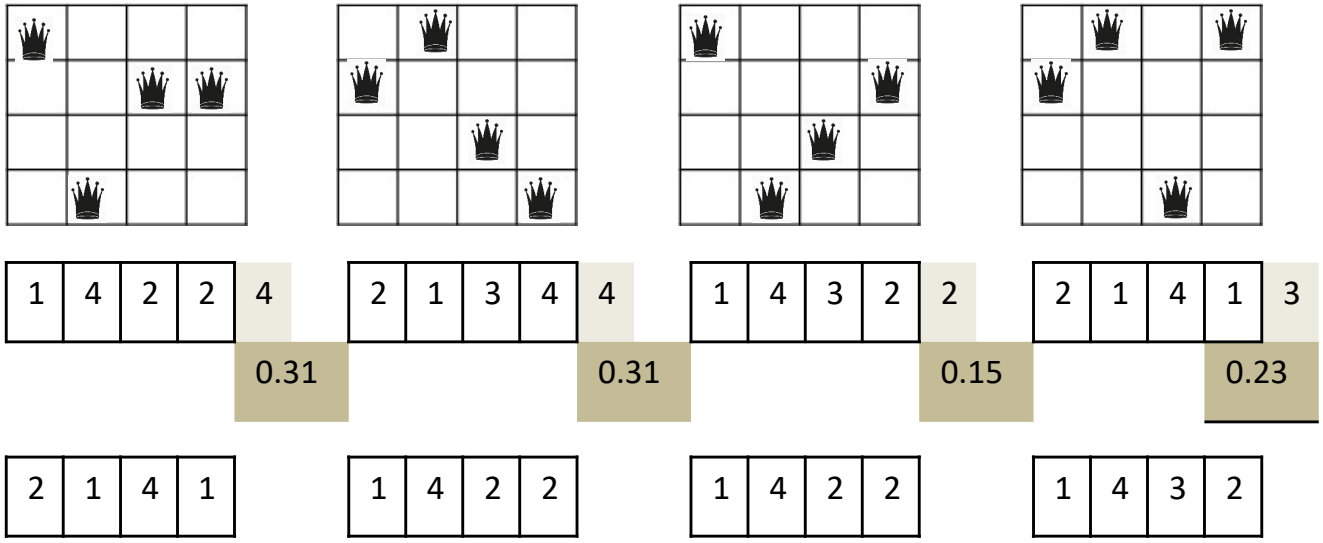




Sample winners of game -1 ,2,3,4 : B4, B1, B1, B3

Genetic Algorithm

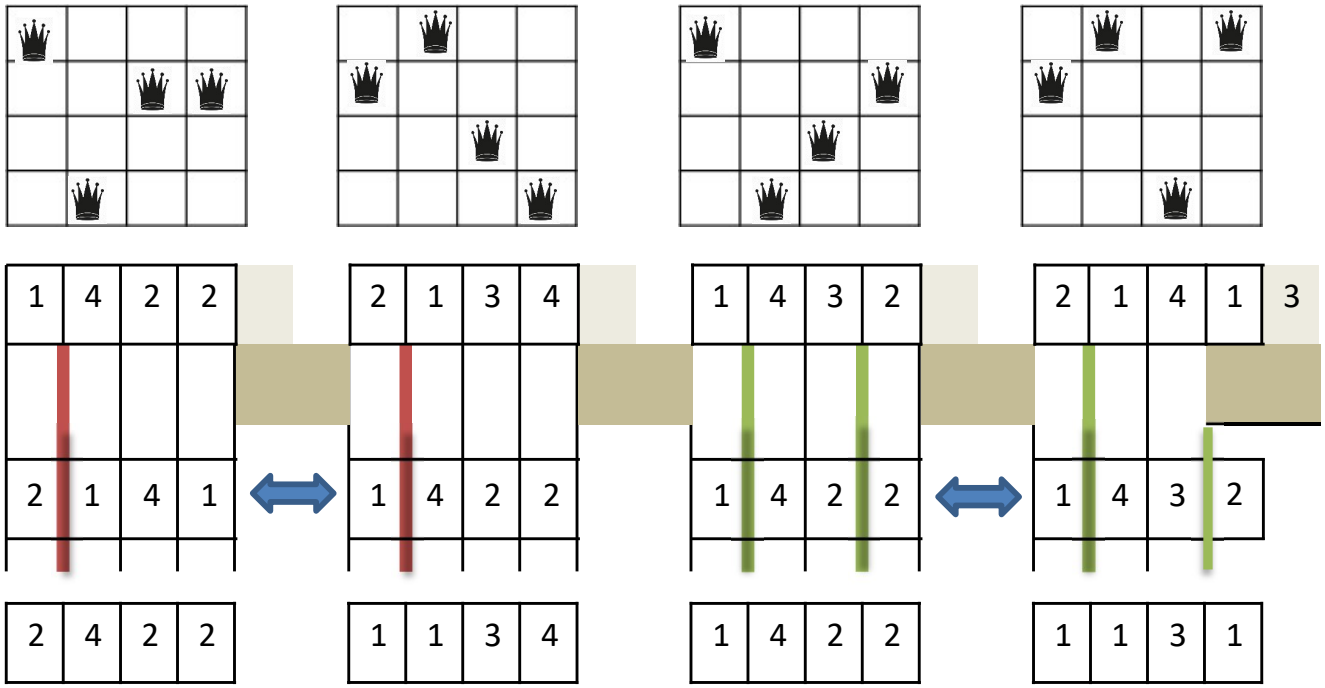
1. Select ‘k’ random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. ~~If anyone of the state’s has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops~~
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



Sample winners of game -1 ,2,3,4 : B4, B1, B1, B3

Genetic Algorithm

1. Select ‘k’ random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. ~~If anyone of the state’s has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops~~
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



Genetic Algorithm – Reference to alternative approaches of crossover

2	4	2	2

1	1	3	4

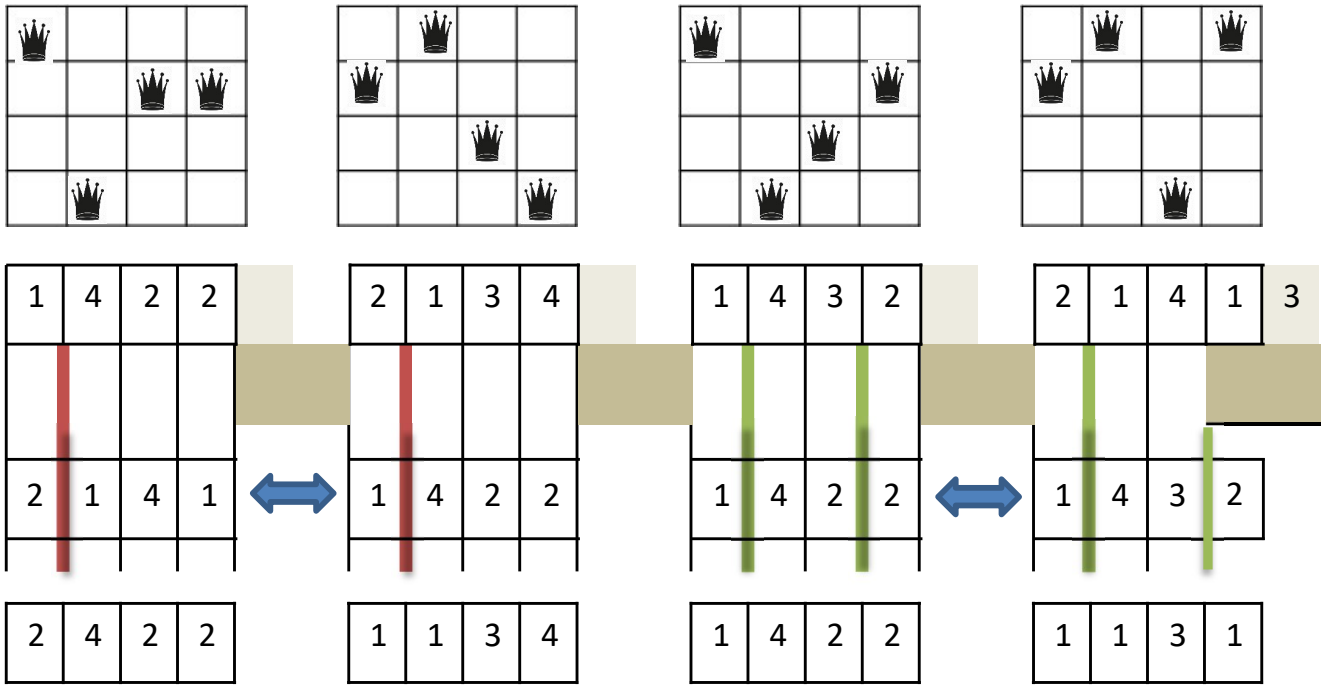
1	4	2	2

1	1	3	1

http://ictactjournals.in/paper/IJSC_V6_I1_paper_4_pp_1083_1092.pdf

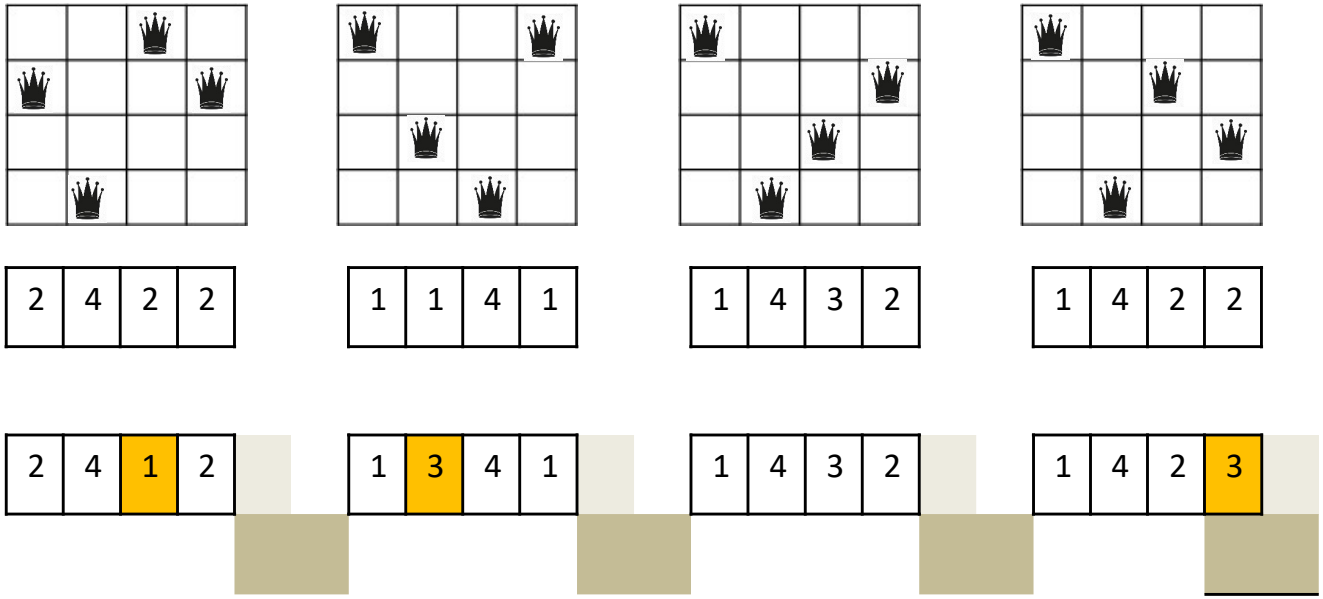
Genetic Algorithm

1. Select ‘k’ random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. ~~If anyone of the state’s has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops~~
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



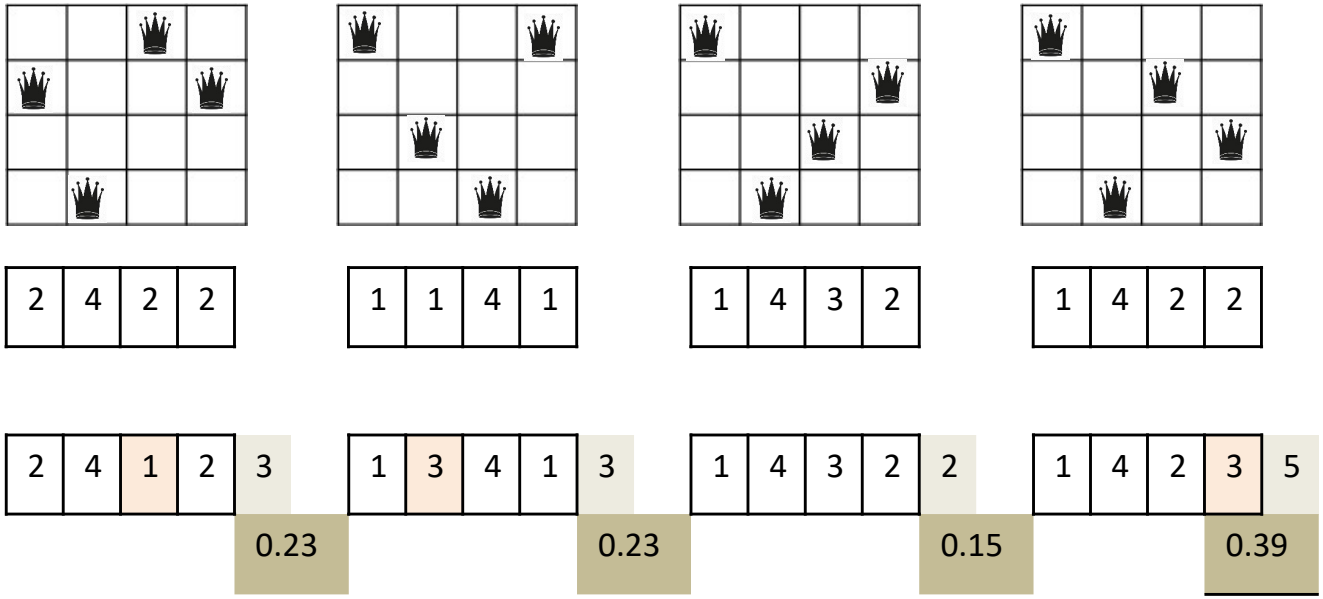
Genetic Algorithm

1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. ~~If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops~~
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



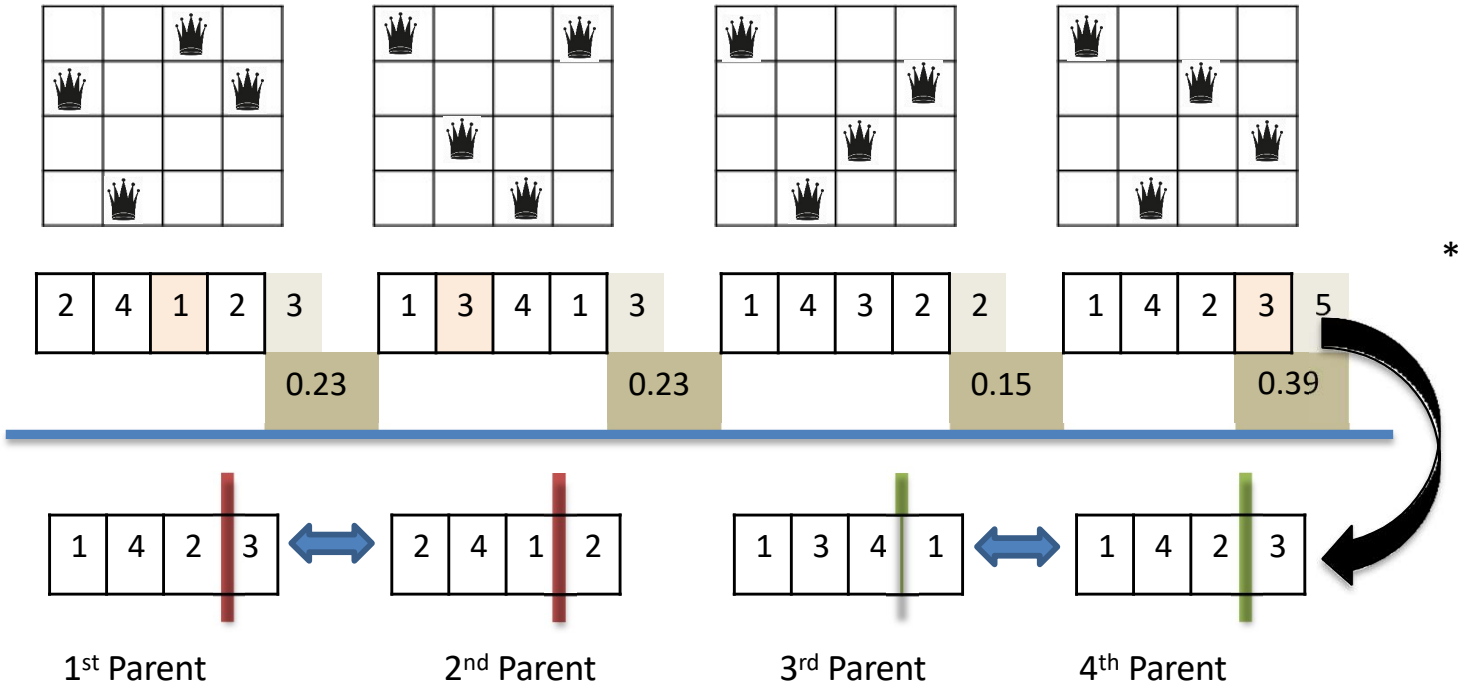
Genetic Algorithm

1. Select ‘k’ random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. If anyone of the state’s has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



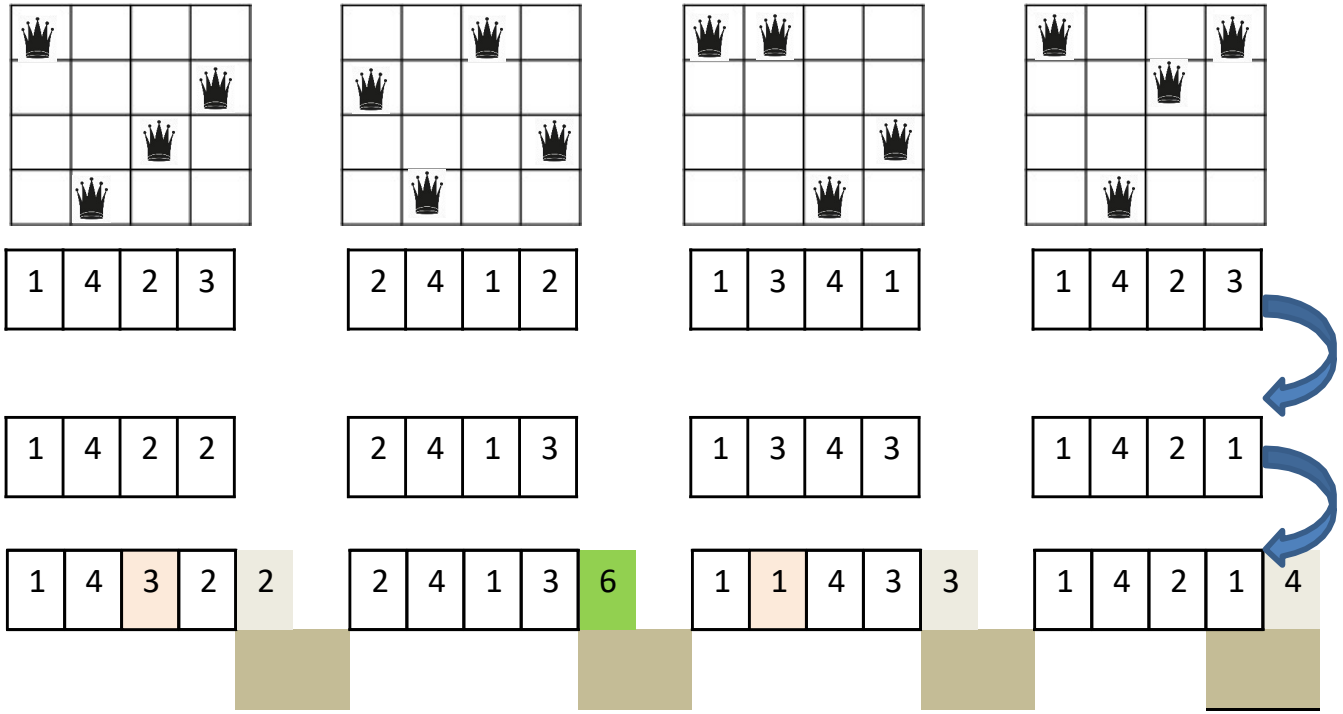
Genetic Algorithm

1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



Genetic Algorithm

1. Select ‘k’ random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens → Threshold = 6
3. If anyone of the state’s has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



Genetic Algorithm

Techniques:

1. Design of the fitness function
2. Diversity in the population to be accounted
3. Randomization

Application:

- Creative tasks
- Exploratory in nature
- Planning problem
- Static Applications

Genetic Algorithm

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Required Reading: AIMA – Chapter 4 - 4.1

Thank You for all your Attention

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