



Artificial and Computational Intelligence

AIMLCLZG557

Contributors & Designers of document content : Cluster Course Faculty Team

M2 : : Problem Solving Agent using Search

BITS Pilani

Pilani Campus

Presented by

Faculty Name

BITS Email ID

Artificial and Computational Intelligence

Disclaimer and Acknowledgement



- Few content for these slides may have been obtained from prescribed books and various other source on the Internet
- I hereby acknowledge all the contributors for their material and inputs and gratefully acknowledge people others who made their course materials freely available online.
- .I have provided source information wherever necessary
- This is not a full fledged reading materials. Students are requested to refer to the textbook w.r.t detailed content of the presentation deck that is expected to be shared over e-learning portal - taxilla.
- I have added and modified the content to suit the requirements of the class dynamics & live session's lecture delivery flow for presentation
- **Slide Source / Preparation / Review:**
- From BITS Pilani WILP: Prof.Raja vadhana, Prof. Indumathi, Prof.Sangeetha
- From BITS Oncampus & External : Mr.Santosh GSK

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, Reinforcement Learning
- M7 Ethics in AI

Learning Objective

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

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. Differentiate which local search is best suitable for given problem
5. Design fitness function for a problem
6. Construct a search tree
7. 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)
8. Design and show Genetic Algorithm steps for a given problem

Module 2 : Problem Solving Agent using Search

- A. Uninformed Search
- B. Informed Search
- C. Heuristic Functions
- D. Local Search Algorithms & Optimization 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 \leq Cost of optimal solution for real problem

N-Queen

	Q		
			Q
Q			
		Q	

Q1			
	Q2		



Initial

State

Goal State

□ 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

Q3
Q1			
	Q2		

	..	Q3	..
Q1			
	Q2		

..	Q3
Q1			
	Q2		

Q1			
..	Q3
	Q2		

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-Queen

	Q		
			Q
Q			
		Q	

Goal State

Q1			
	Q2		



Initial State

Q3
Q1			
	Q2		

	..	Q3	..
Q1			
	Q2		

..	Q3
Q1			
	Q2		

Q1			
..	Q3
	Q2		

Possible Heuristic Design:

H1(n) : Number of Non conflicting Pairs of Queen (MAX)				
H2(n) : Number of Conflicting Pairs of Queen (MIN)				
H1(n) : Number of safest Queen (MAX)				

N-Queen

	Q		
			Q
Q			
		Q	

Goal State

Q1			
	Q2		

State

Initial



Q3
Q1			
	Q2		

	..	Q3	..
Q1			
	Q2		

..	Q3
Q1			
	Q2		

Q1			
..	Q3
	Q2		

		Q3	
Q1			
Q4
	Q2		

		Q3	
Q1			
..	Q4
	Q2		

..	Q4
Q1			
			Q3
	Q2		

..	..	Q4	..
Q1			
			Q3
	Q2		

N-Tile

-	1	2
3	4	5
6	7	8

Goal State

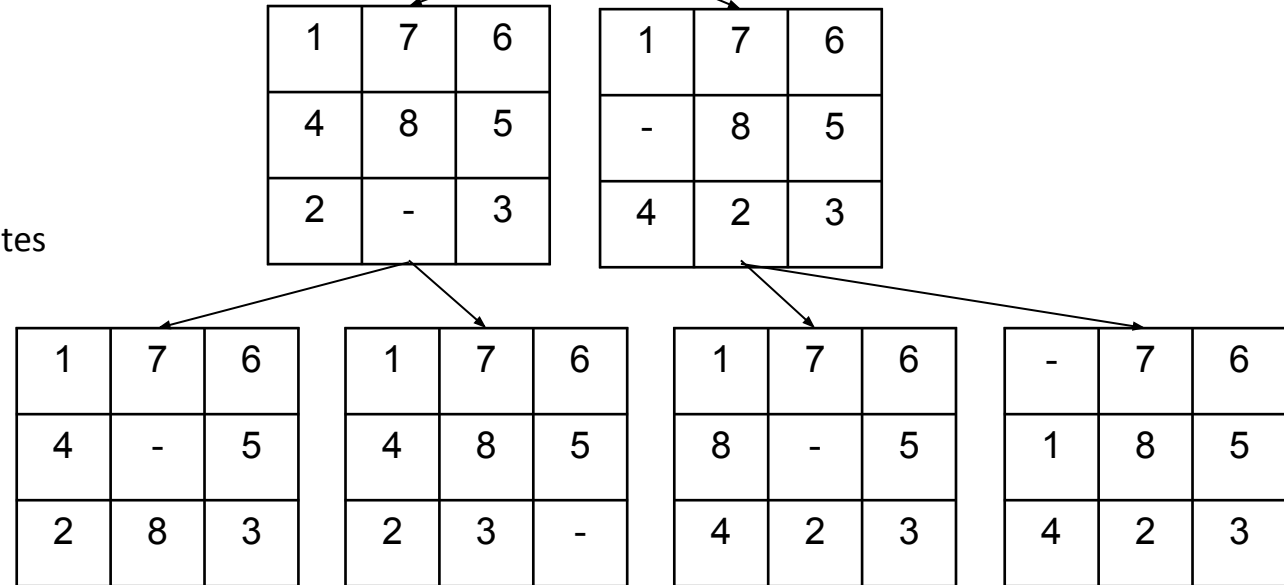
- Effective Branching Factor
- : ~3
- : Avg.cost = 18
- : No.of.States = $\sim 3^{18}$
- : Graph states : $9!/2 = 181,440$ states

1	7	6
4	8	5
-	2	3

Initial State



State



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!

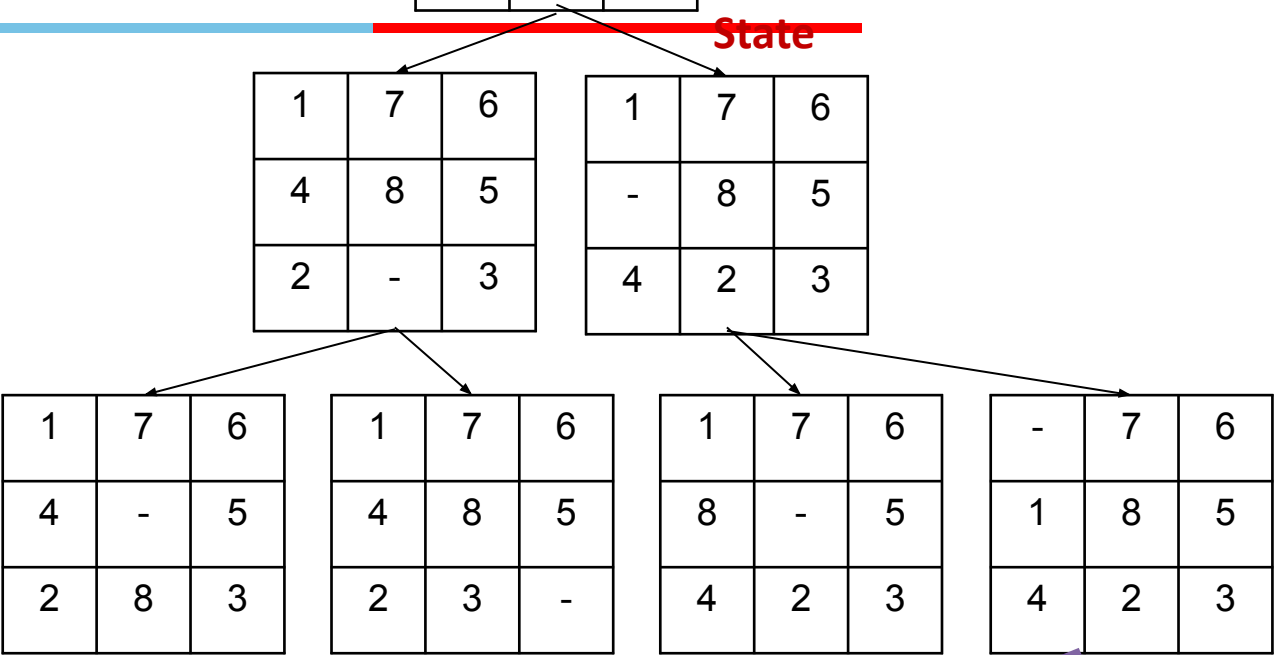
N-Tile

-	1	2
3	4	5
6	7	8

Goal State

1	7	6
4	8	5
-	2	3

Initial State



Possible Heuristic Design:

H1(n) : Manhattan distance of Empty tile	
H1(n) : Manhattan distance of all labelled tile	
H1(n) : Number of Misplaced tile (MAX)	

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 tileh` (n)
Example 1	7	10	7
Example 2	5	6	6
.....

-	1	2
3	4	5
6	7	8

1	7	6
4	8	5
-	2	3

Create a suitable model:

$$h(n) = c1*X1(n) + c2*X2(n) +$$

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

Hill Climbing

Simulated Annealing

Local Beam Search

Tabu Search

Multiple Instance Based

Genetic Algorithm

Particle Swarm Optimization

Ant Colony Optimization

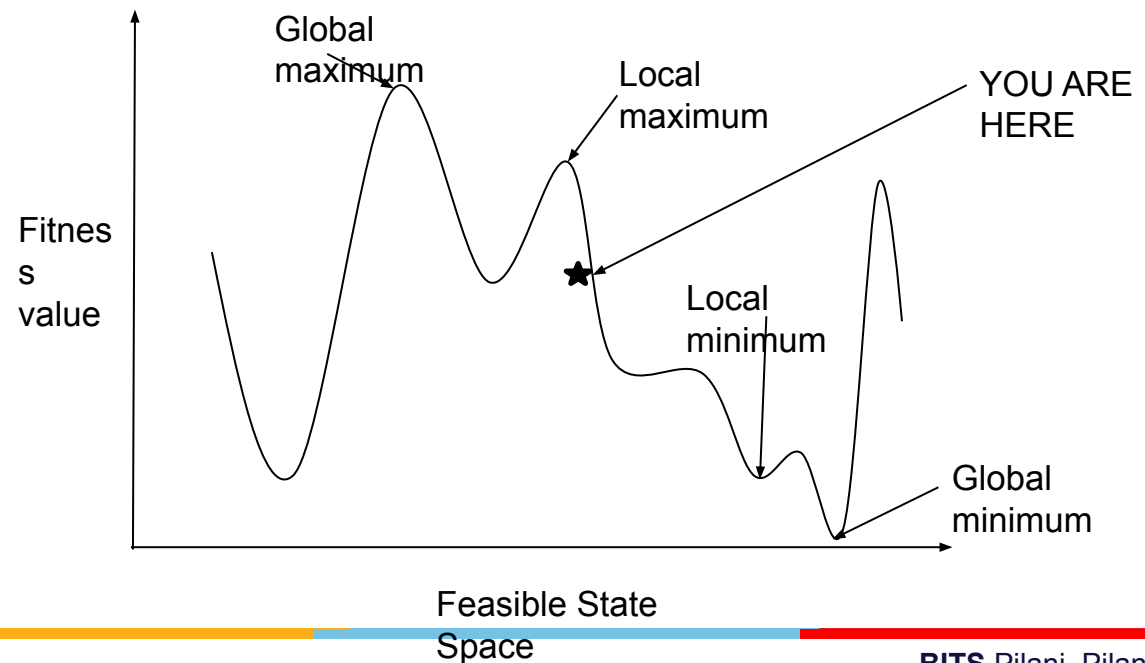
Terminology

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

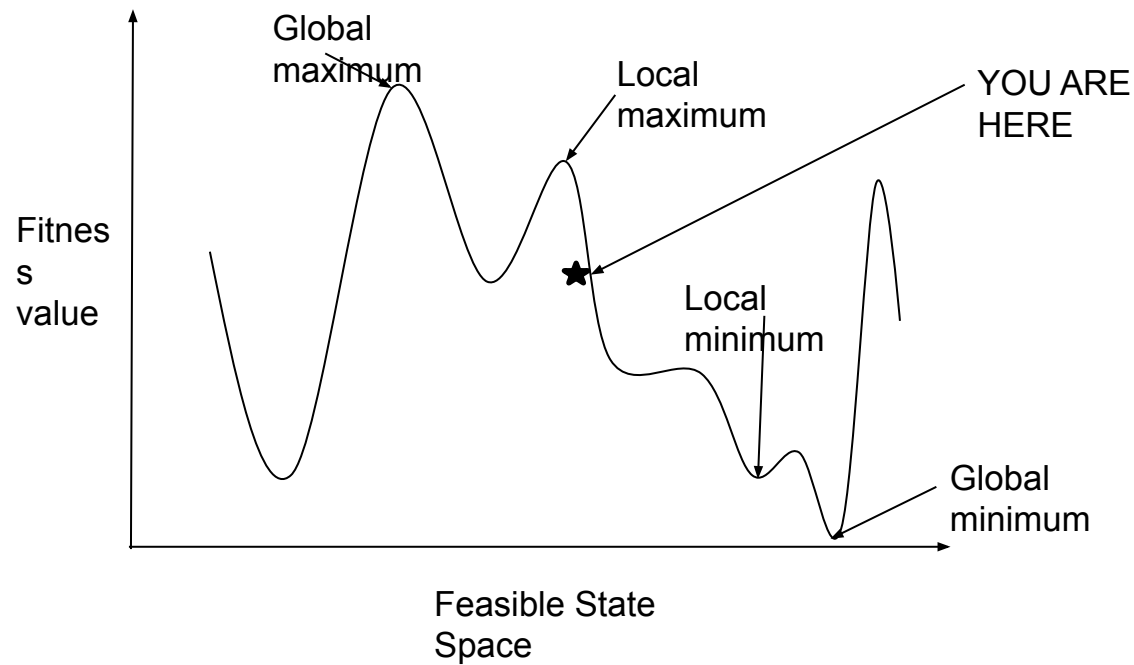
2	5	3	2
♠	6	♠	♠
3	5	4	2
4	♠	4	2





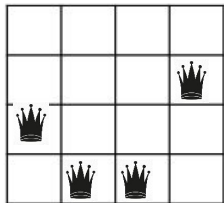
Hill Climbing

Hill Climbing



Random Restart

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



3	4	4	2	3
---	---	---	---	---

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

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

loop do

neighbor \leftarrow a highest-valued successor of *current*

if *neighbor*.VALUE \leq *current*.VALUE **then return** *current*.STATE

current \leftarrow *neighbor*

Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state

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

Q1-Q2

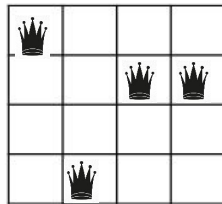
Q1-Q3

Q1-Q4

Q2-Q3

Q2-Q4

Q3-Q4

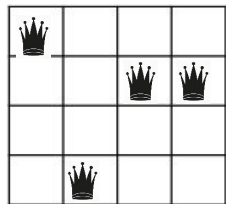


1	4	2	2	4
---	---	---	---	---

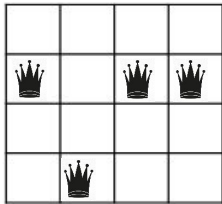
Note : Steps 3 & 4 in the above algorithm will be a part of variation of Hill climbing

Hill Climbing

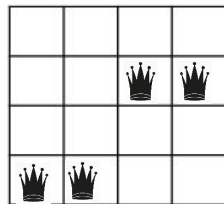
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state



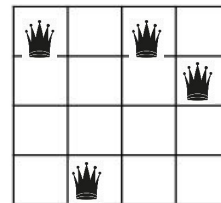
1	4	2	2
---	---	---	---



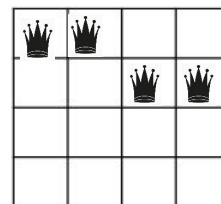
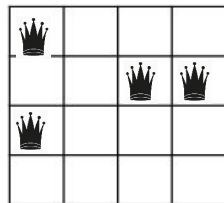
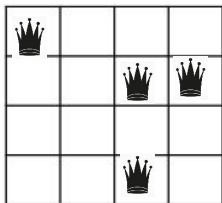
2	4	2	2
---	---	---	---



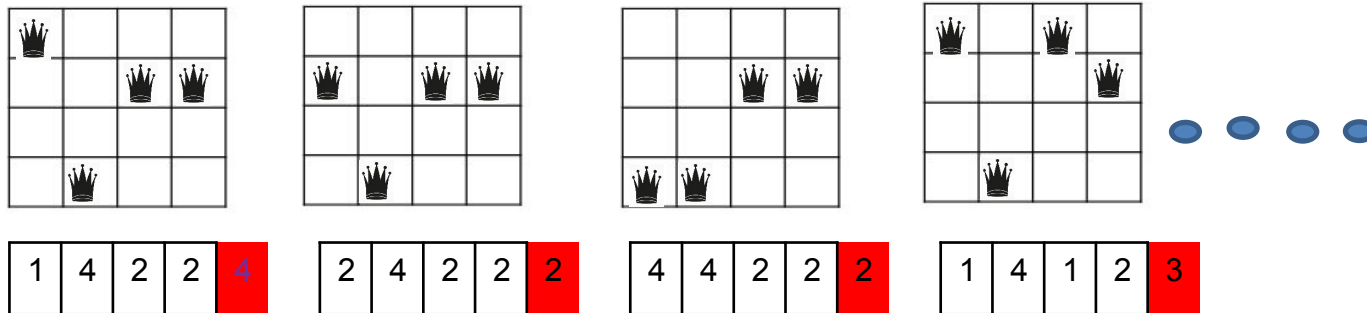
4	4	2	2
---	---	---	---



1	4	1	2
---	---	---	---



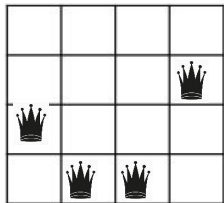
Hill Climbing



Local Maxima ☐ Random Restart

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 \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

loop do

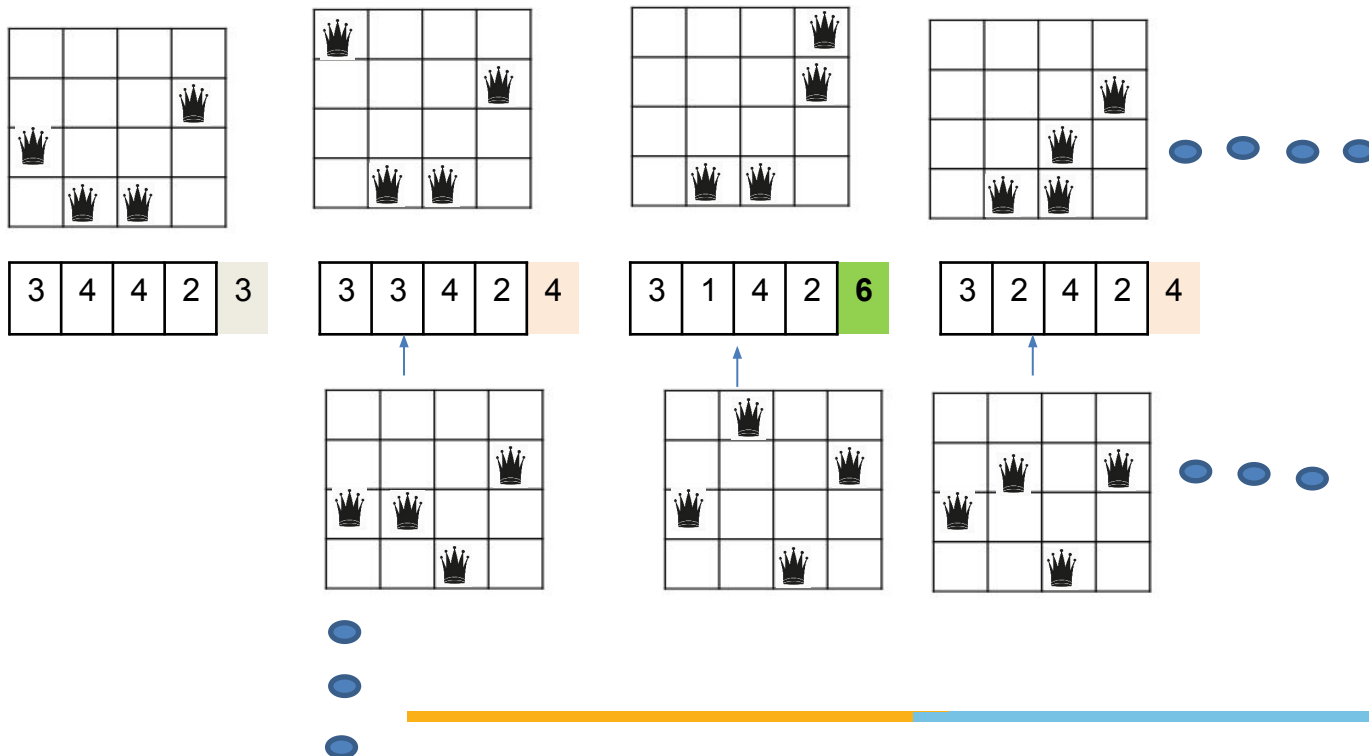
neighbor \leftarrow a highest-valued successor of *current*

if *neighbor*.VALUE \leq *current*.VALUE **then return** *current*.STATE

current \leftarrow *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

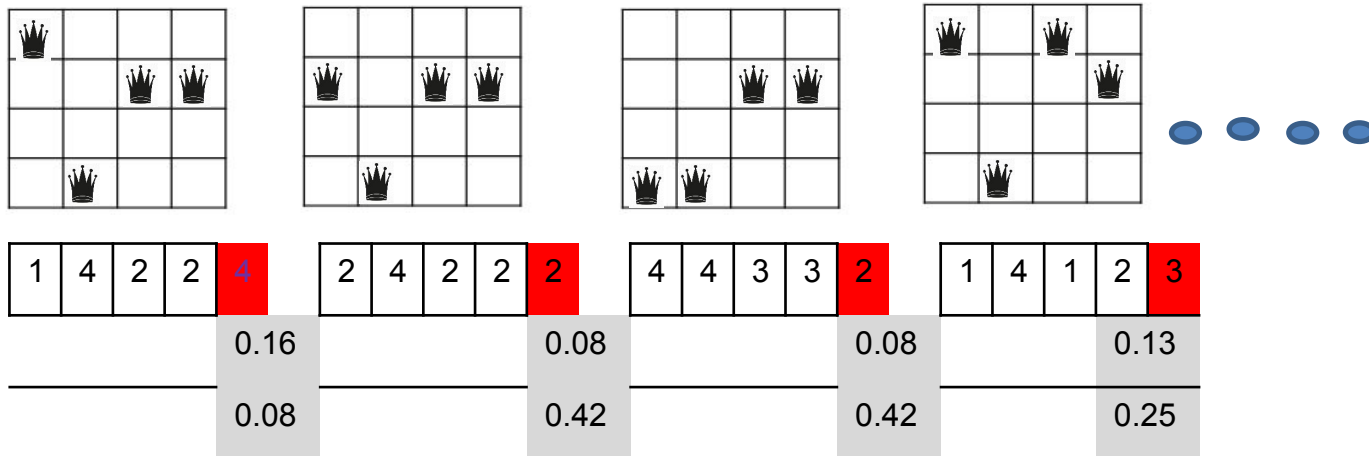


Stochastic Hill Climbing

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

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 = {4,2,2,3,3,2,1,3,2,1,3,2}

Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ *schedule*(t)

if $T = 0$ **then return** *current*

next \leftarrow a randomly selected successor of *current*

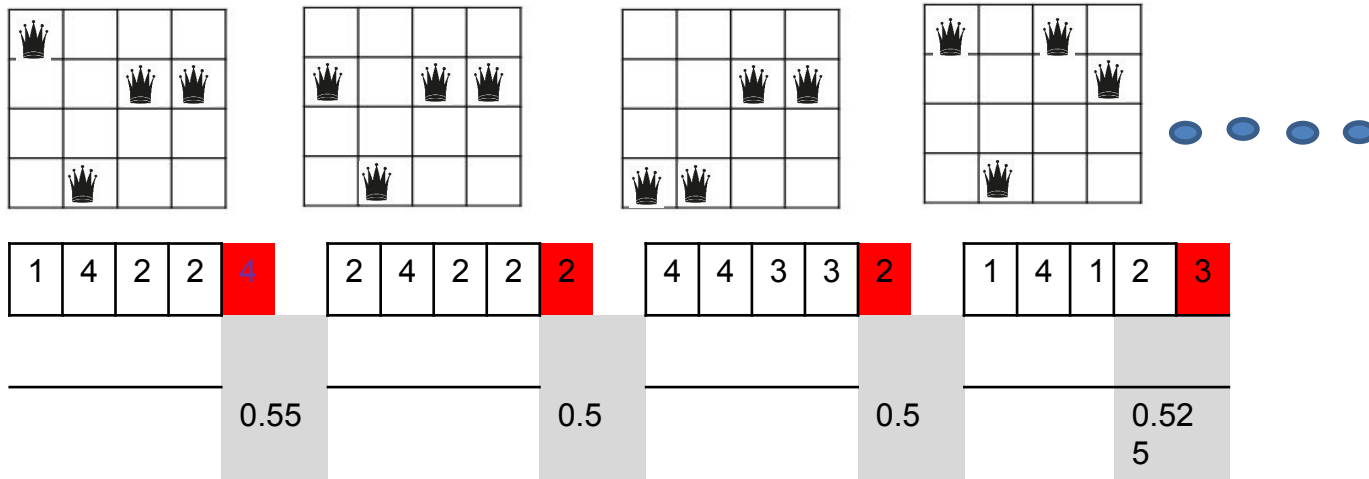
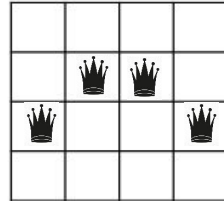
$\Delta E \leftarrow$ *next*.VALUE – *current*.VALUE

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Simulated Annealing

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



$12\ N = \{4, 2, 2, 3, 3, 2, 1, 3, 2, 1, 3, 2\}$

Init = 2

Simulated Annealing

Current Value = 4 (Local Maxima)

Global Maxima = 6

Next Value	ΔE	$\Delta E/t$			$\Delta E/t$		
2	2	0.1	1.12	0.47	0.4	1.49	0.40
3	1	0.05	1.05	0.49	0.2	1.22	0.45
5	-1	-0.05	0.95	0.51	-0.2	0.82	0.55

Simulated Annealing



Maximization problem design to achieve global minima

Set Temp to very high temp t

Set n as number of iteration to be performed at a particular t

L1: Randomly select a random neighbour

Calculate Energy barrier $E = f(N) - f(C)$

If $E > 0$ then its a good move

Move ahead for next tree search level

Else

Create a random number $r: [0-1]$

If $r < e^{-E/t}$

Choose this bad state & move downhill

Else

Go to L1.

If Goal is reached or {acceptable goal(set criteria to check)node is reached & t is small END}

Else

If no.of.neighbors explored has reached a threshold $\geq n$

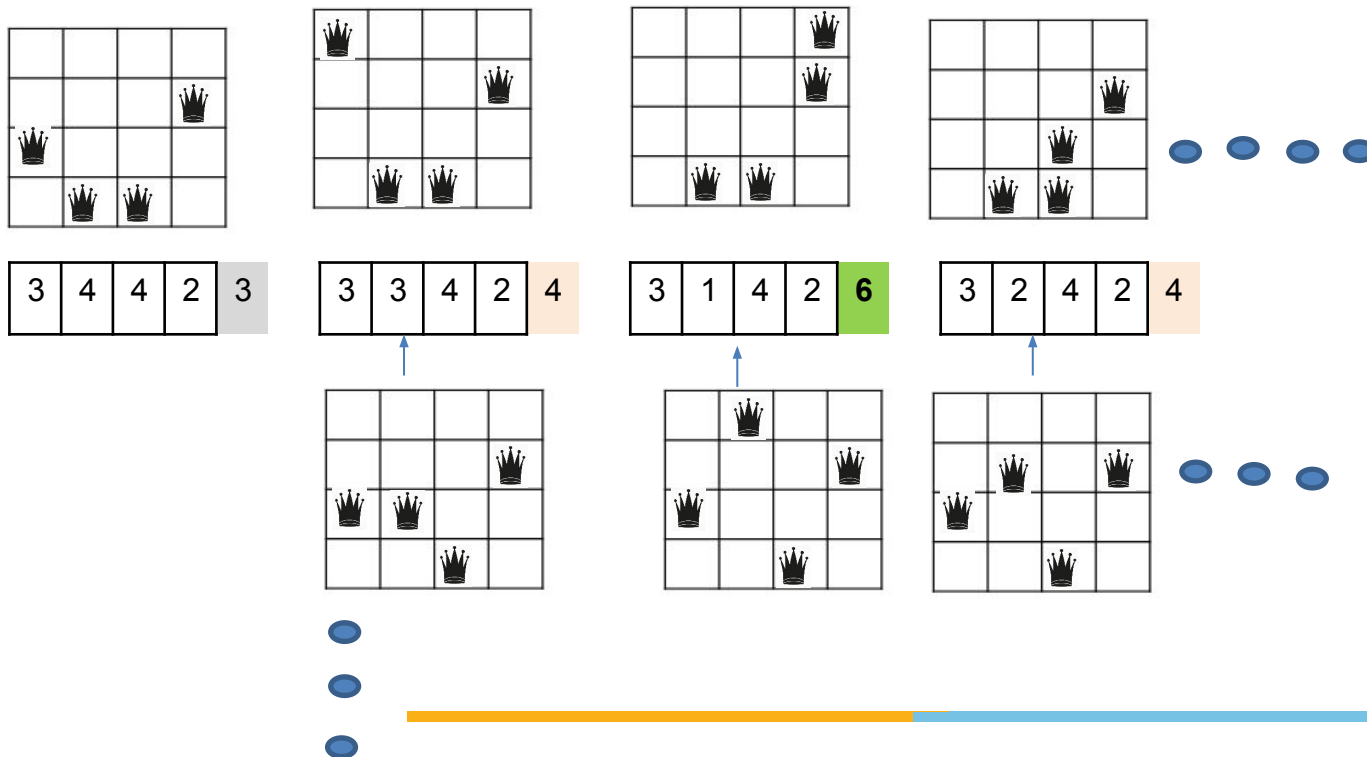
then Lower t and go to L1.

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 probability
6. Repeat from Step 2

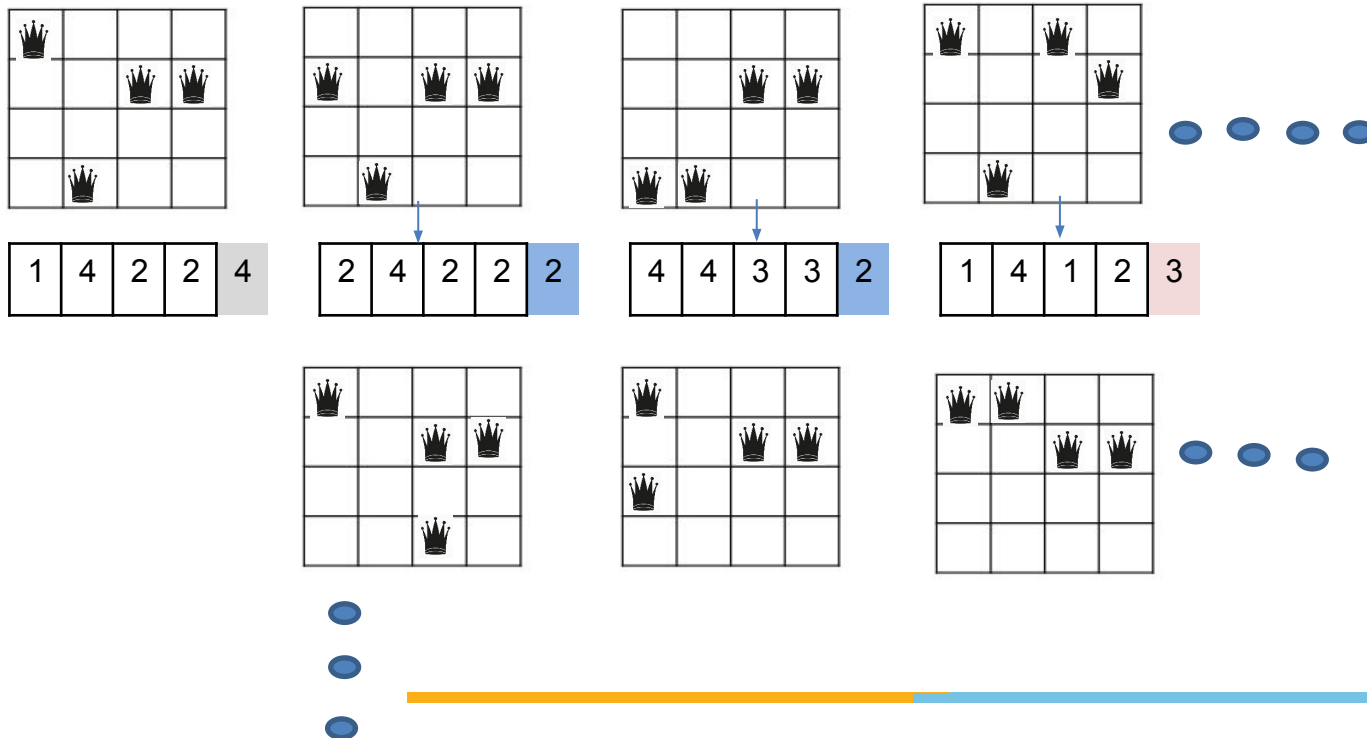


Stochastic Beam Search



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

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 \leftarrow empty set

for $i = 1$ to SIZE(*population*) **do**

$x \leftarrow$ RANDOM-SELECTION(*population*, FITNESS-FN)

$y \leftarrow$ RANDOM-SELECTION(*population*, FITNESS-FN)

child \leftarrow REPRODUCE(x, y)

if (small random probability) **then** *child* \leftarrow MUTATE(*child*)

add *child* to *new_population*

population \leftarrow *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

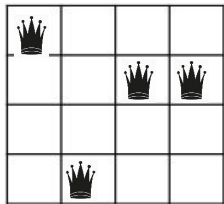
inputs: x, y , parent individuals

$n \leftarrow$ LENGTH(x); $c \leftarrow$ random number from 1 to n

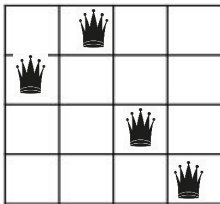
return APPEND(SUBSTRING($x, 1, c$), SUBSTRING($y, c + 1, n$))

Genetic Algorithm

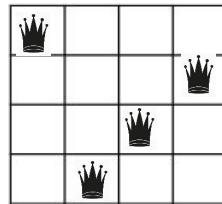
1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select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



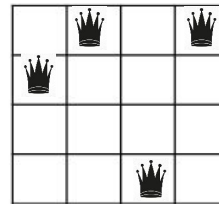
1	4	2	2
---	---	---	---



2	1	3	4
---	---	---	---



1	4	3	2
---	---	---	---

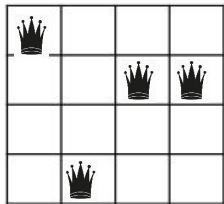


2	1	4	1
---	---	---	---

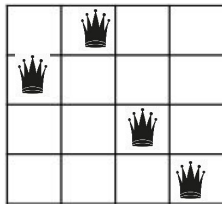
Genetic Algorithm



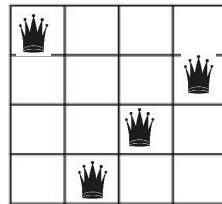
1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens □ Threshold = 6
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration 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



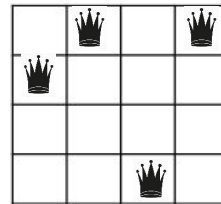
1	4	2	2	4
---	---	---	---	---



2	1	3	4	4
---	---	---	---	---



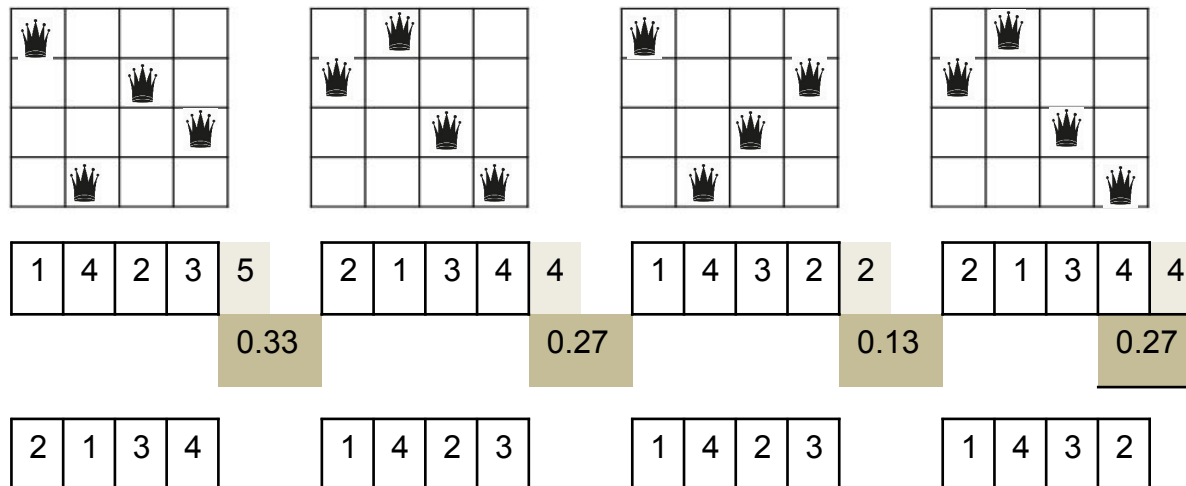
1	4	3	2	2
---	---	---	---	---



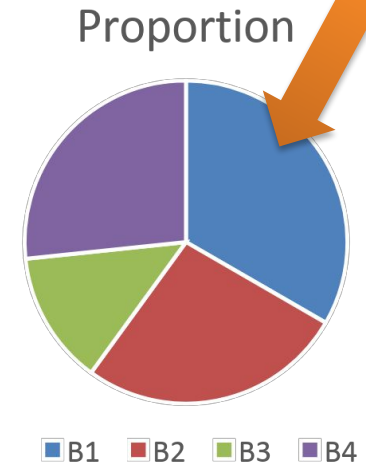
2	1	4	1	3
---	---	---	---	---

Genetic Algorithm – Example 1

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



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

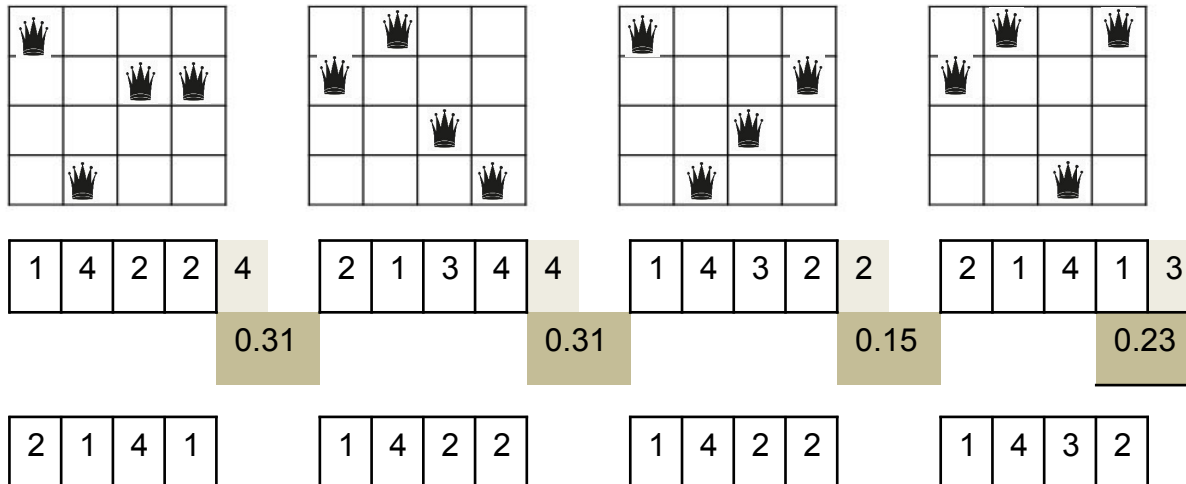


Genetic Algorithm –Example 2



Selection

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



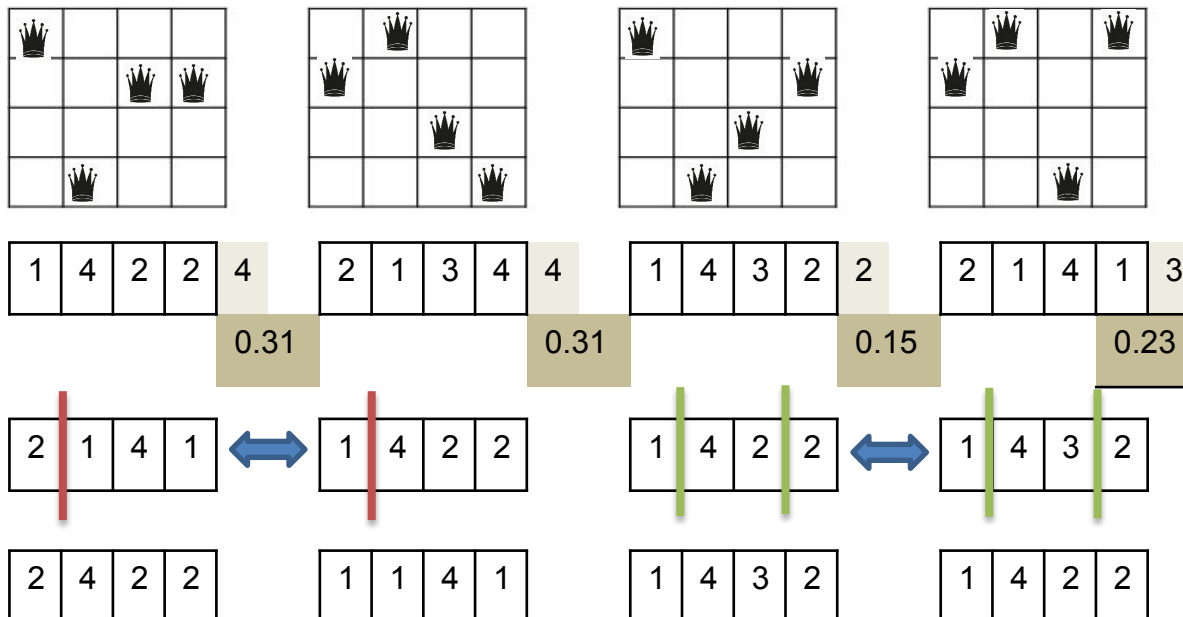
Sample winners of game -1 ,2,3,4 : B4, B1, B1, B3

Genetic Algorithm - Example 2



Crossover

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

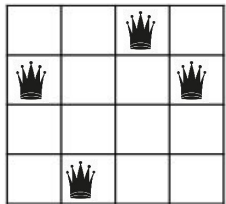


Genetic Algorithm - Example 2



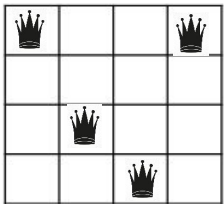
Mutation

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



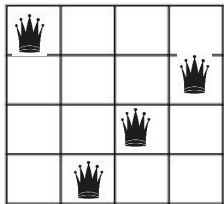
2	4	2	2
---	---	---	---

2	4	1	2
---	---	---	---



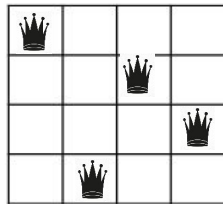
1	1	4	1
---	---	---	---

1	3	4	1
---	---	---	---



1	4	3	2
---	---	---	---

1	4	3	2
---	---	---	---

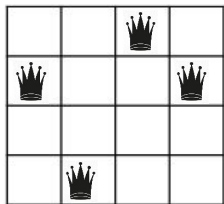


1	4	2	2
---	---	---	---

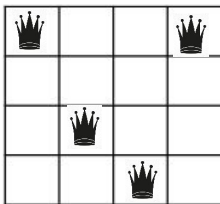
1	4	2	3
---	---	---	---

Genetic Algorithm

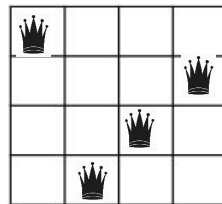
1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens □ Threshold = 6
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration 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



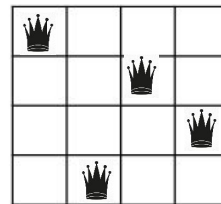
2	4	2	2
---	---	---	---



1	1	4	1
---	---	---	---



1	4	3	2
---	---	---	---



1	4	2	2
---	---	---	---

2	4	1	2	3
---	---	---	---	---

0.23

1	3	4	1	3
---	---	---	---	---

0.23

1	4	3	2	2
---	---	---	---	---

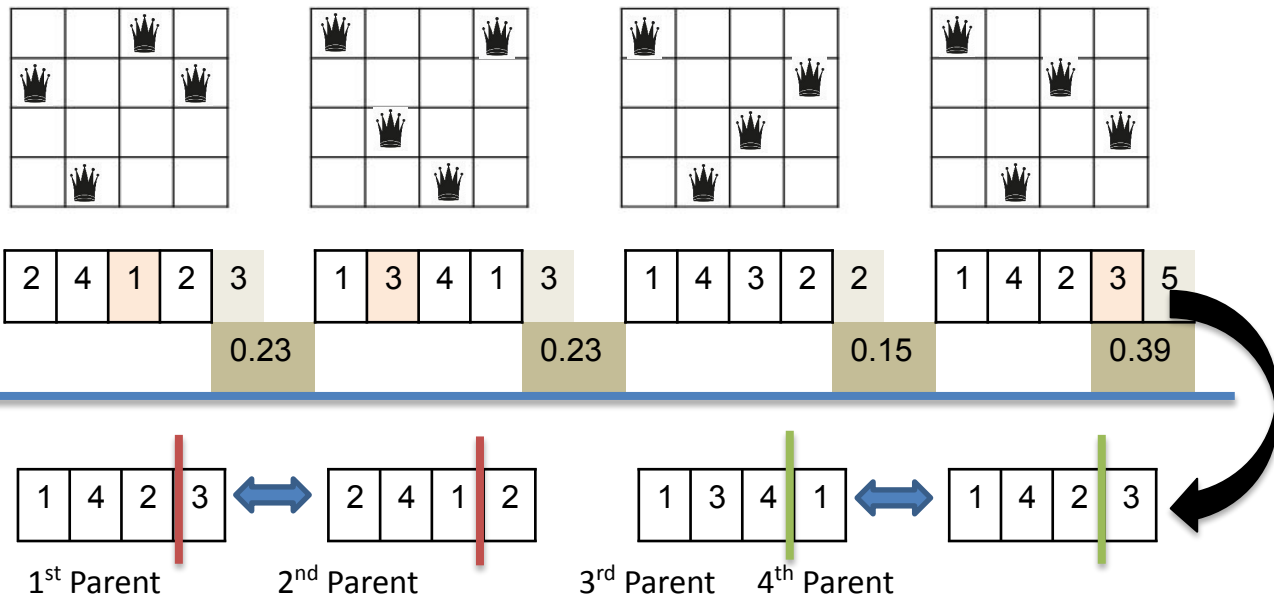
0.15

1	4	2	3	5
---	---	---	---	---

0.39

Genetic Algorithm

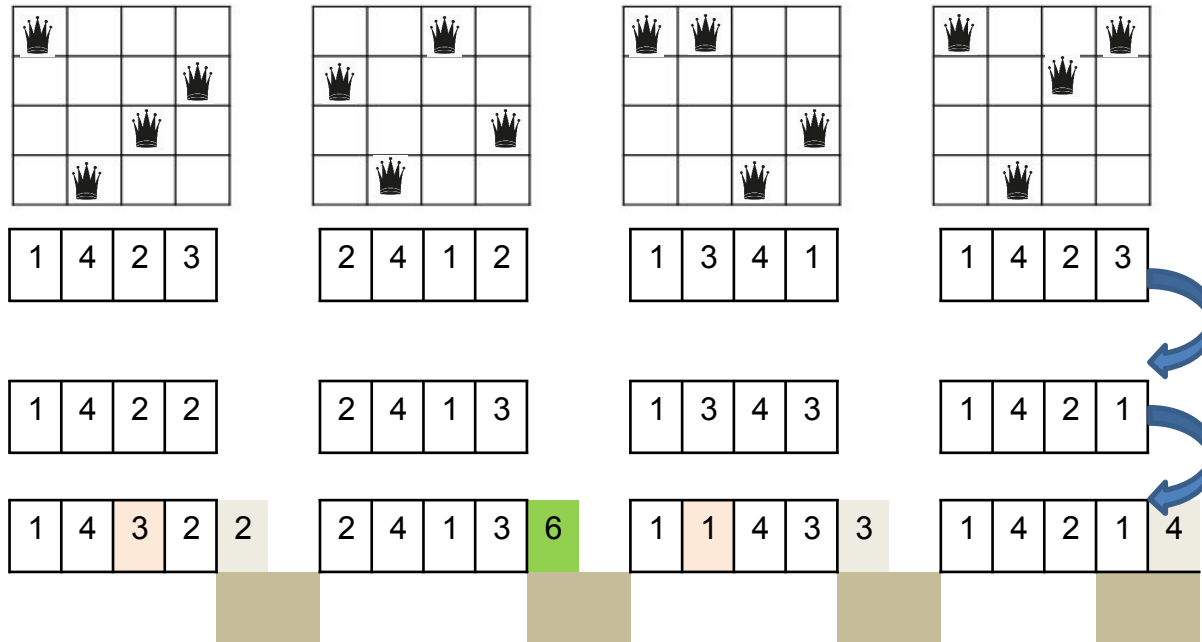
1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens □ Threshold = 6
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration 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



Other crossover and mutation operators are shared in separate document & uploaded in the elearn portal

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

Required Reading: AIMA - Chapter # 4.1, #4.2

Thank You for all your Attention

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