CSE422: Artificial intelligence

Genetic Algorithm

Asif Shahriar
Lecturer, CSE, BRACU

Genetic Algorithm – Borrowed from Nature

- In nature, evolution helps populations of organisms adapt to their environment over time
  - GA mimics this process to evolve better solutions to problems
- Each organism has a genome (a string of DNA) that encodes its characteristics
- Organisms that are better suited to the environment are more likely to survive and reproduce (survival of the fittest)
  - In problem solving, fitness = how good is a candidate solution
- Two organisms mix their genes to create offspring
- Genes can mutate randomly—this introduces new traits that might be beneficial
  - Can be a way to escape the local maxima/minima
- Over generations, the population becomes better adapted

0/1 Knapsack

Based on the information below try to maximize the value of the 0/1 knapsack problem when the size of your knapsack is **12Kg**

Item	Weight	Value
A	5 Kg	$12
B	3 Kg	$5
C	7 Kg	$10
D	2 Kg	$7

0/1 Knapsack

Step 1: Generate Initial Population (Encoding).

Randomly generate 4-bit strings (chromosome), where each bit represents whether the item is chosen (1) or not (0)

- 0 1 1 0 -> B and C chosen
- 1 0 0 0 -> A chosen
- 1 1 1 1 -> A, B, C, D chosen
- 0 0 1 1 -> C and D chosen

Step 2: Calculate fitness score.

- 0 1 1 0 -> 10 Kg - $15
- 1 0 0 0 -> 5 Kg - $12
- 1 1 1 1 -> 17 Kg - $34
- 0 0 1 1 -> 9 Kg - $17

0/1 Knapsack

Step 3: Select Parents (Natural selection).

$$p_i = f_i / \sum_i f_i \quad \text{for } i = 1,2,\ldots n$$

String No	Chromosome	Weight	Fitness Score (f_i)	p_i
1	0 1 1 0	10	15	**0.34**
2	1 0 0 0	5	12	**0.27**
3	1 1 1 1	17	**0**	**0.0**
4	0 0 1 1	9	17	**0.39**
SUM			44	1

0/1 Knapsack

Step 4: Crossover.

Let we chose parents (1, 4) and (2, 4)

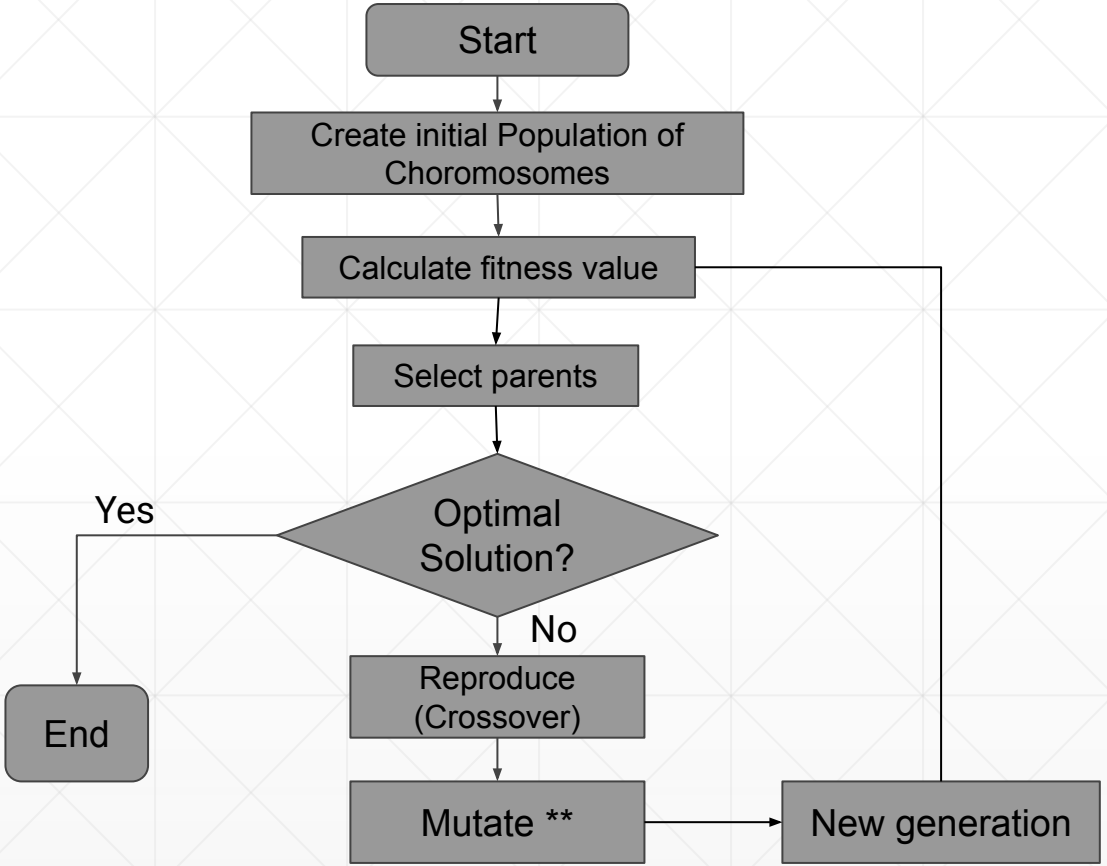
String No	Mating Pool	Crossover point	Offspring	Weights	Fitness Score (f_i)
1	0 | 1 1 0	0	0 0 1 1	9	**17**
4	0 | 0 1 1	0	0 1 1 0	10	**15**
2	1 0 0 | 0	2	1 0 0 1	7	**19**
4	0 0 1 | 1	2	0 0 1 0	7	**10**
SUM					61

0/1 Knapsack

Step 5: Mutation.

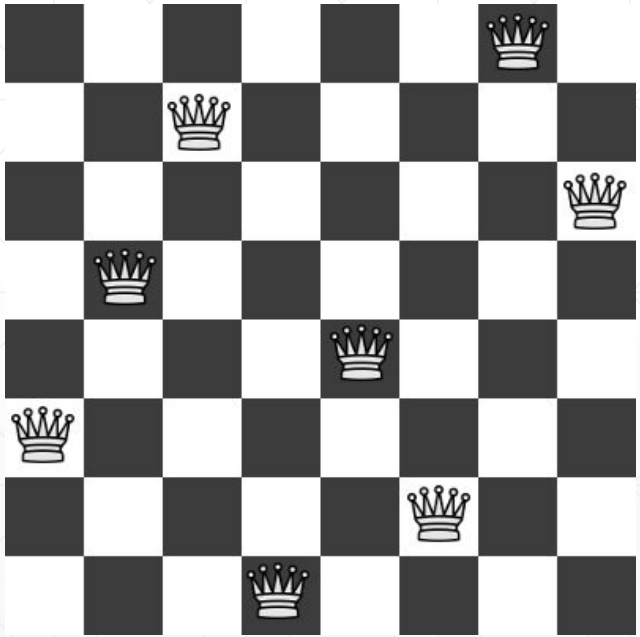
String No	Offspring	Mutation Point	Offspring (After mutation)	Weights	Fitness Score (f_i)
1	0 0 1 1	3	0 0 1 0	7	10
4	0 1 1 0	0	1 1 1 0	**17**	**0**
2	1 0 0 1	1	1 1 0 1	**10**	**24**
4	0 0 1 0	0	1 0 1 0	12	22
SUM					56

Genetic Algorithm: FLOWCHART



N Queens Problem

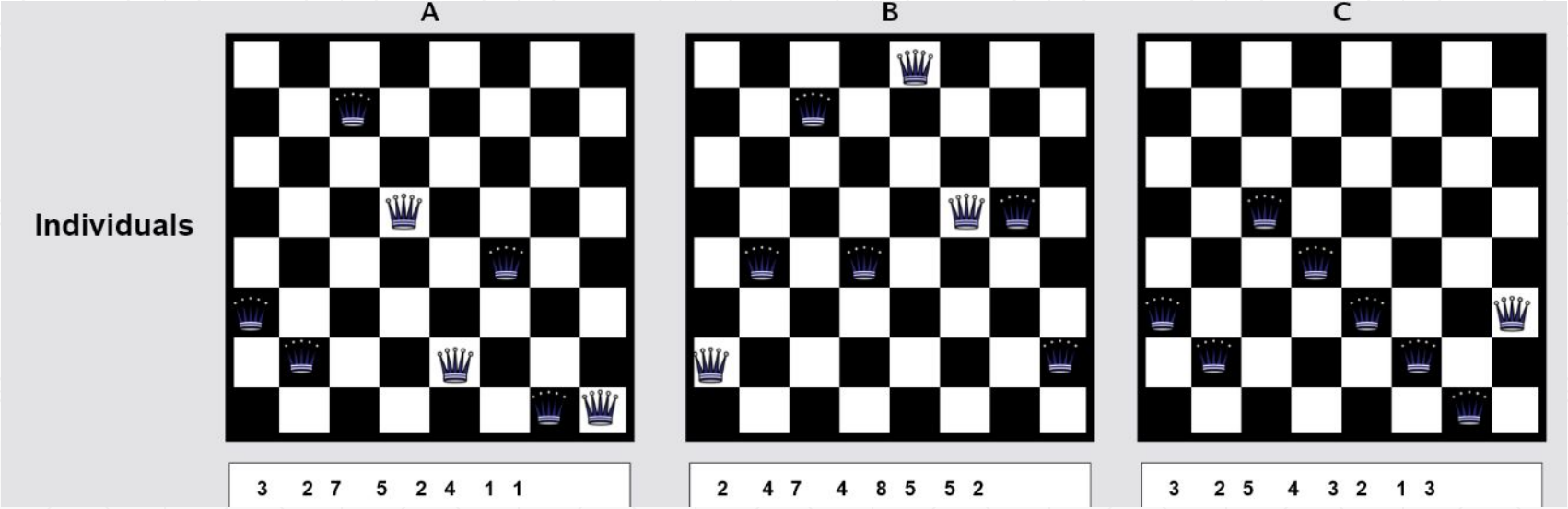
The goal is to place N queens on an NxN chessboard so that no two queens threaten or attack each other. To prevent the queens from attacking one another, no two queens should be in the same row, column, or diagonal.



8 Queens Problem

Step 1: Initial population (Encoding).

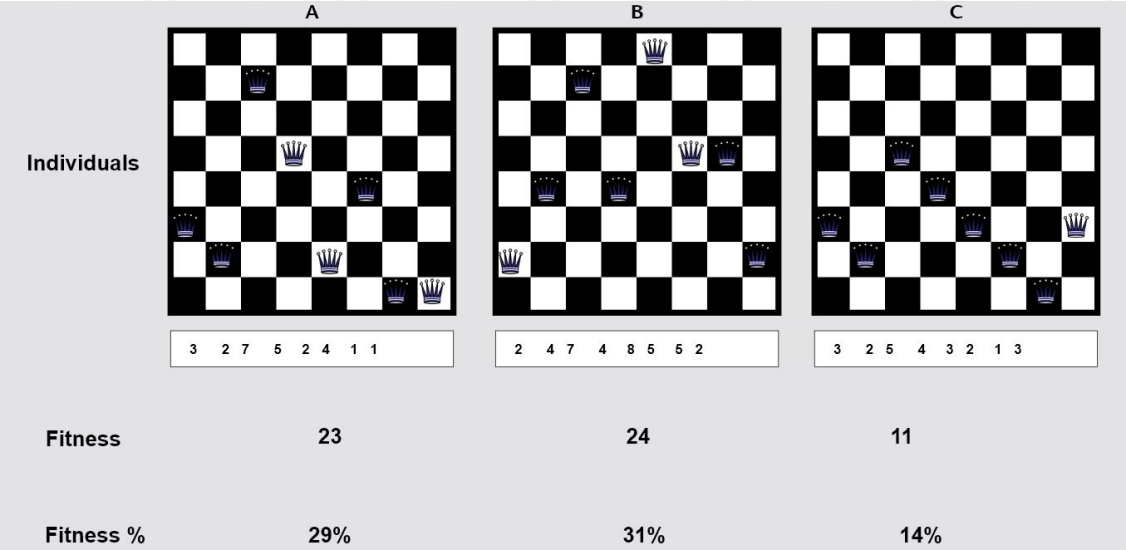
Chromosomes of 8 bits – i-th bit represents position of a queen in i-th column



8 Queens Problem

Step 2: Calculate fitness score.

Fitness function: **number of non-attacking pairs of queens**



8 Queens Problem

Step 3: Select parents.

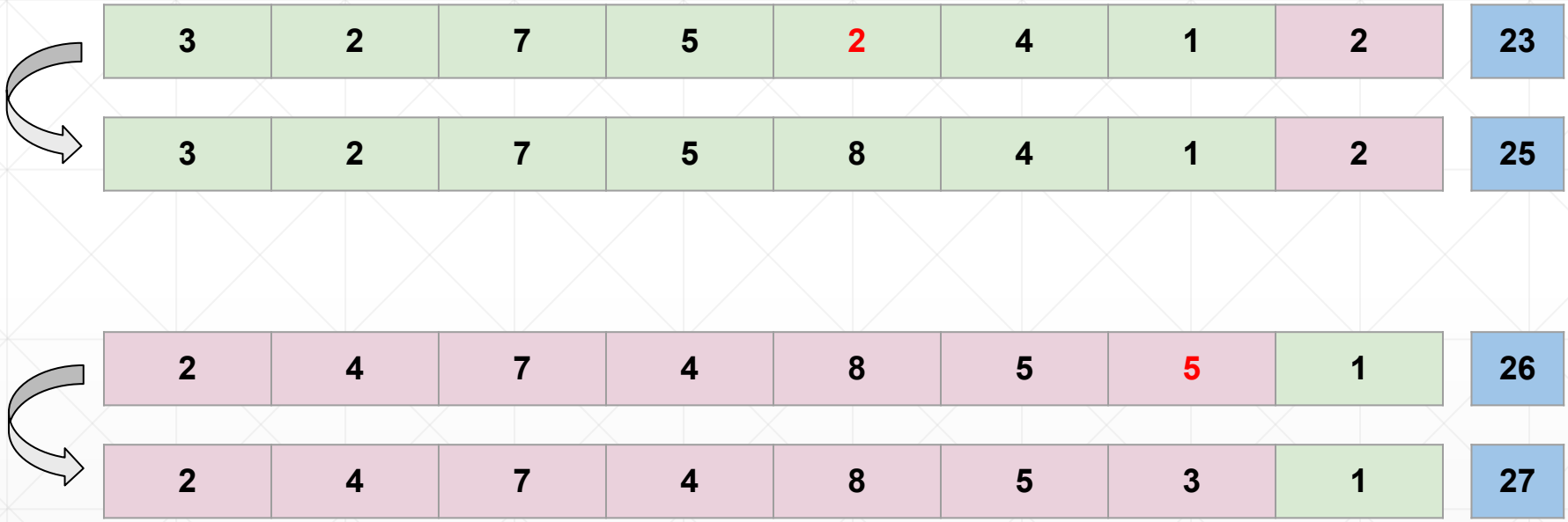
3	2	7	5	2	4	1	1
2	4	7	4	8	5	5	2

Step 4: Crossover.

3	2	7	5	2	4	1	2	23
2	4	7	4	8	5	5	1	26

8 Queens Problem

Step 5: Mutation.



8 Queens Problem

Step 6: New generation.

- Suppose we have: parents [p1, p2, p3, p4] and children [c1, c2, c3, c4]
- New generation size must be equal to previous generation
- Option 1: new generation = children
- Option 2: **elitism**
    - k-elitism: choose k best chromosomes from [parents, children], remaining chromosomes will be from children
    - Suppose in terms of fitness, c1 > p4 > p3 > c2 > c4 > p1 > p2 > c3
    - Using 2-elitism, new generation = [c1, p4, c2, c4]

Pseudocode

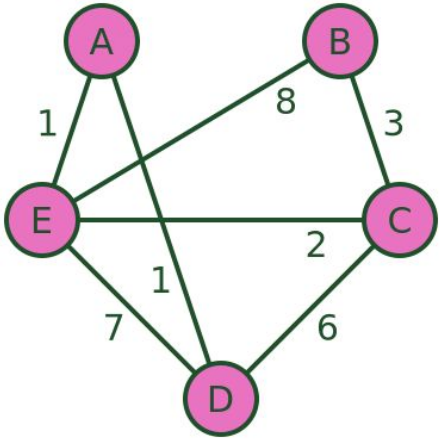
function GENETIC_ALGORITHM(*population,* FITNESS-FN) **return** an individual
    input: *population*, a set of individuals
        FITNESS-FN, a function which determines the quality of the individual
    repeat
        new_population ← empty set
        for i **from** 1 **to** SIZE(*population*) **do**
            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

Practice problem 1

You are to produce the following string **“I_LOVE_AI”** using the following characters “ABC…Z_” (27 in total). Use GA to produce this string.

- What will be the chromosome?
- What should be the fitness function?
- Work out all steps for 5 iterations.

Practice problem 2: TSP



5 cities are connected using bi-directional roads as shown in the graph. A salesman will start from a particular city, visit all cities exactly **ONCE**, and return to the starting city, **while covering the shortest path.**

Find out the tour as well as the cost using GA.

- What will be the chromosome?
- What should be the fitness function?
- Work out all steps for 5 iterations and report the best tour found.

Merits and Demerits of GA

Merits:

- GAs are less likely to get stuck in local optima because of crossover and mutation
- Diverse population encourages exploration across search space
- Flexible and versatile – can be applied to a wide range of problems
- Often finds a “good enough” solution quickly

Demerits:

- Not optimal: often finds “good enough” solutions but not necessarily best solution
- Slow: may take many generations to converge
- Needs careful selection of parameters (mutation rate, num of generations, etc)
- Poor fitness function makes GA ineffective