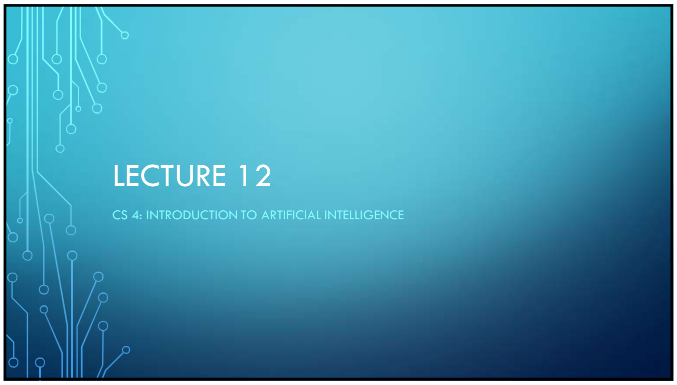


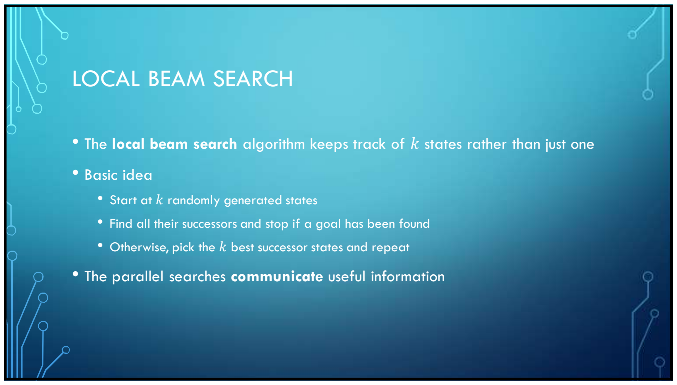
LECTURE 12

CS 4: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

1

Why limit ourselves to just one state?

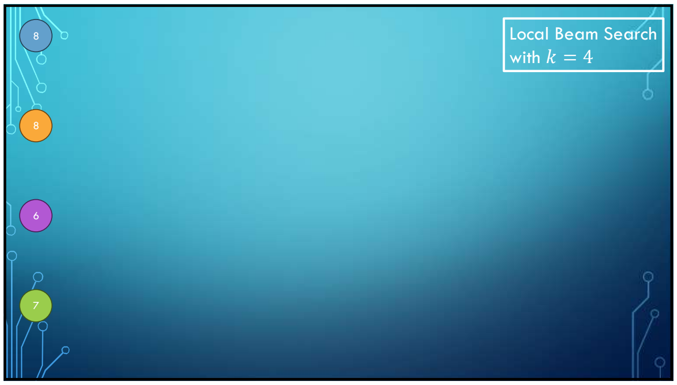
2



LOCAL BEAM SEARCH

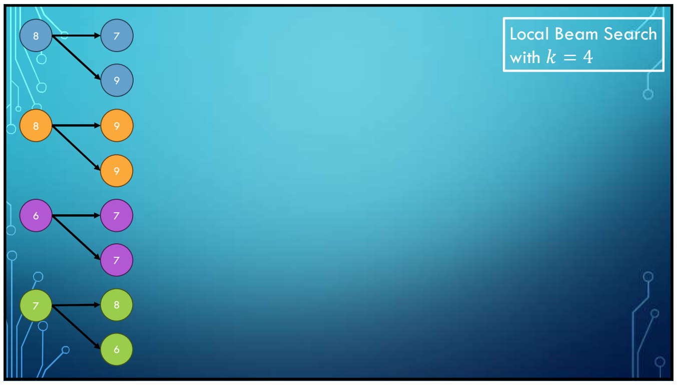
- The **local beam search** algorithm keeps track of k states rather than just one
- Basic idea
  - Start at k randomly generated states
  - Find all their successors and stop if a goal has been found
  - Otherwise, pick the k best successor states and repeat
- The parallel searches **communicate** useful information

3



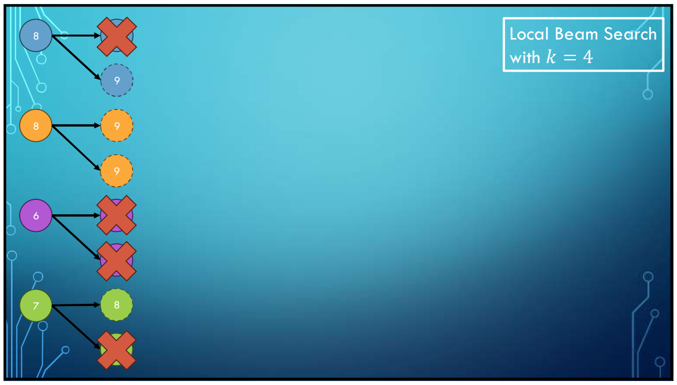
Local Beam Search with $k = 4$

4



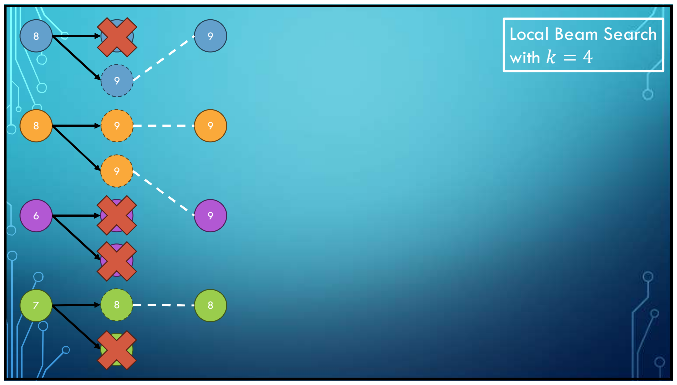
Local Beam Search with $k = 4$

5



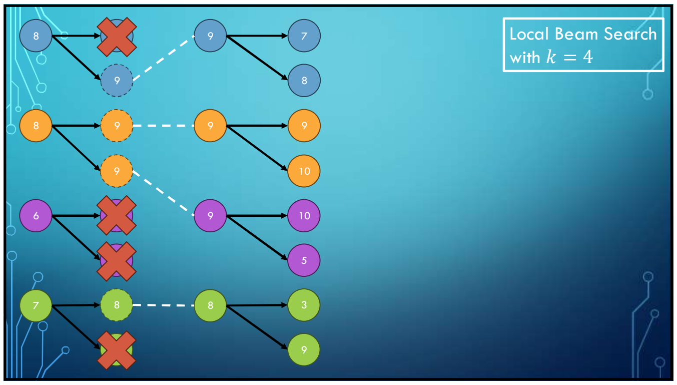
Local Beam Search with $k = 4$

6



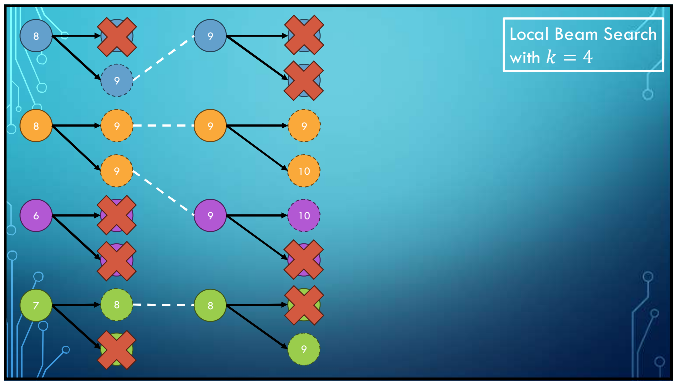
Local Beam Search with $k = 4$

7



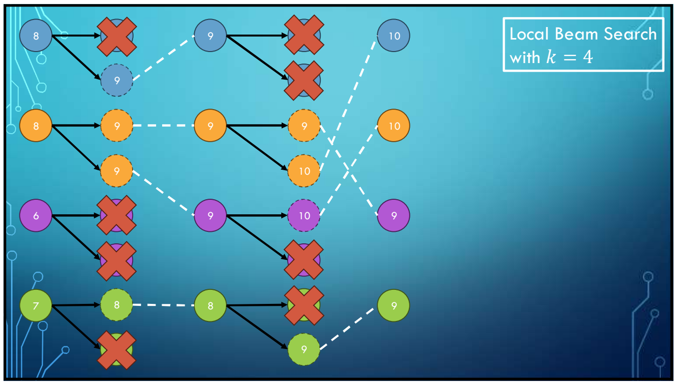
Local Beam Search with $k = 4$

8



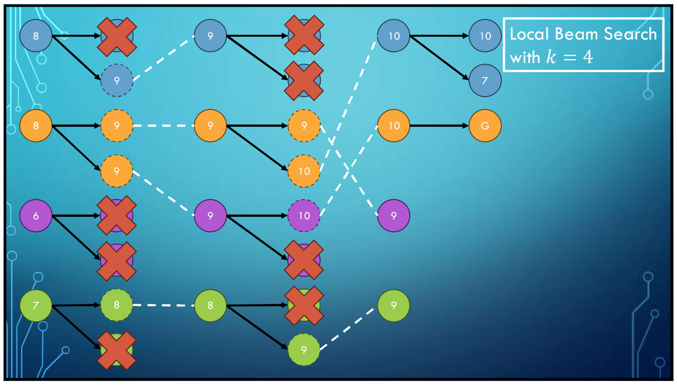
Local Beam Search with $k = 4$

9



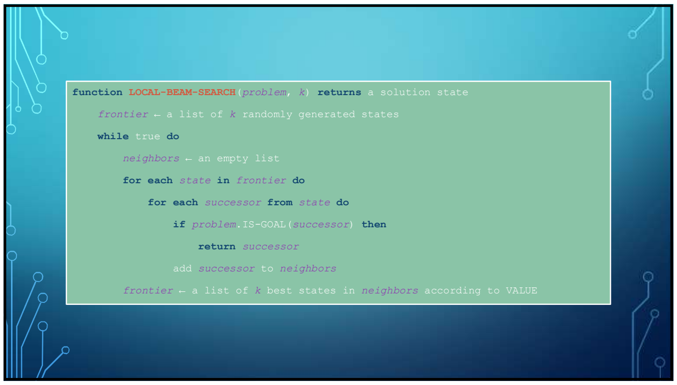
Local Beam Search with $k = 4$

10



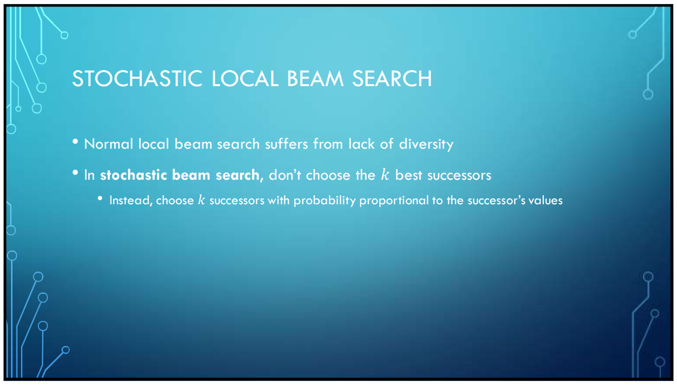
Local Beam Search with $k = 4$

11



```
function LOCAL-BEAM-SEARCH(problem, k) returns a solution state
    frontier ← a list of k randomly generated states
    while true do
        neighbors ← an empty list
        for each state in frontier do
            for each successor from state do
                if problem.IS-GOAL(successor) then
                    return successor
                add successor to neighbors
        frontier ← a list of k best states in neighbors according to VALUE
```

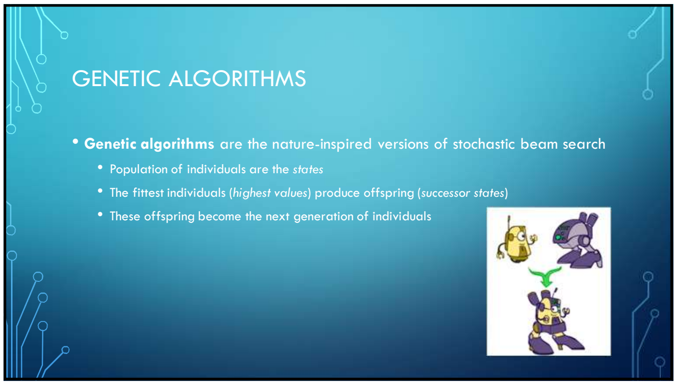
12



STOCHASTIC LOCAL BEAM SEARCH

- Normal local beam search suffers from lack of diversity
- In **stochastic beam search**, don't choose the k best successors
  - Instead, choose k successors with probability proportional to the successor's values

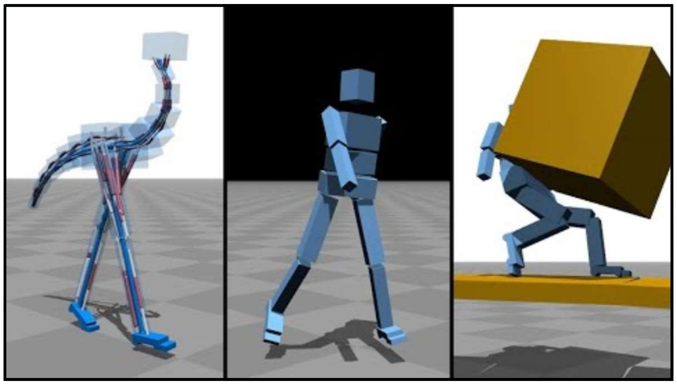
13



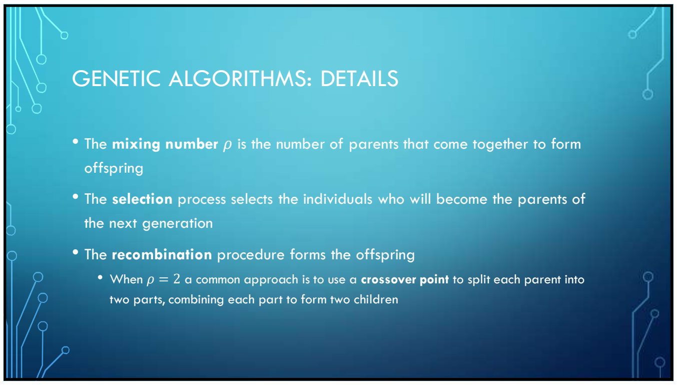
GENETIC ALGORITHMS

- **Genetic algorithms** are the nature-inspired versions of stochastic beam search
  - Population of individuals are the *states*
  - The fittest individuals (*highest values*) produce offspring (*successor states*)
  - These offspring become the next generation of individuals

14



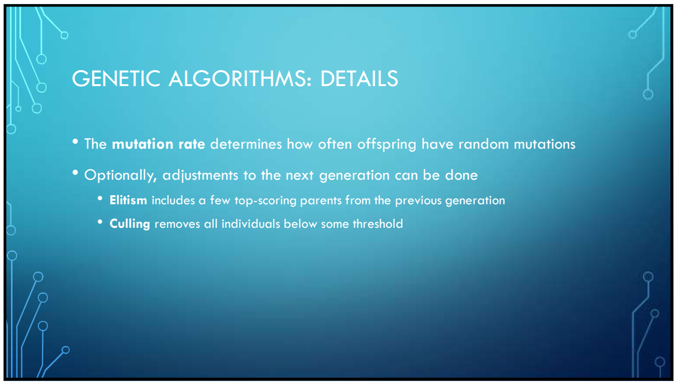
15



GENETIC ALGORITHMS: DETAILS

- The **mixing number** ρ is the number of parents that come together to form offspring
- The **selection** process selects the individuals who will become the parents of the next generation
- The **recombination** procedure forms the offspring
  - When $\rho = 2$ a common approach is to use a **crossover point** to split each parent into two parts, combining each part to form two children

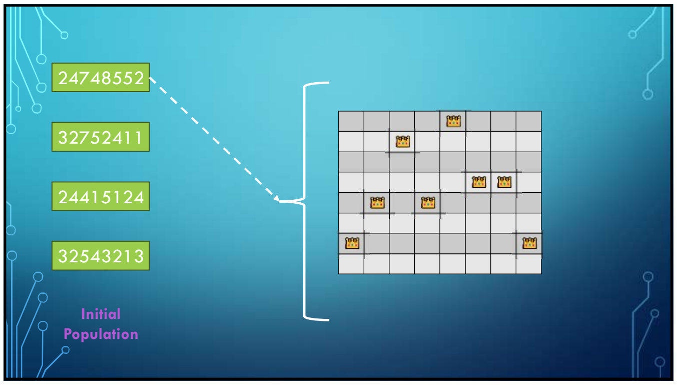
16



GENETIC ALGORITHMS: DETAILS

- The **mutation rate** determines how often offspring have random mutations
- Optionally, adjustments to the next generation can be done
  - **Elitism** includes a few top-scoring parents from the previous generation
  - **Culling** removes all individuals below some threshold

17



24748552

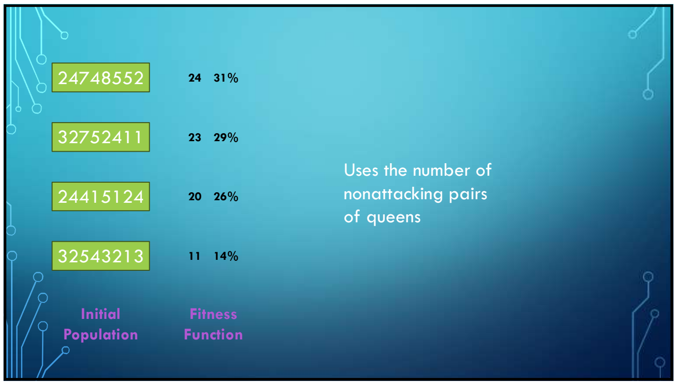
32752411

24415124

32543213

Initial Population

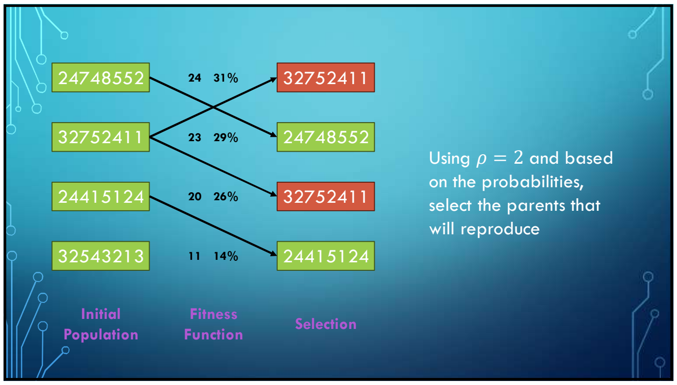
18



Initial Population	Fitness Function	
24748552	24	31%
32752411	23	29%
24415124	20	26%
32543213	11	14%

Uses the number of nonattacking pairs of queens

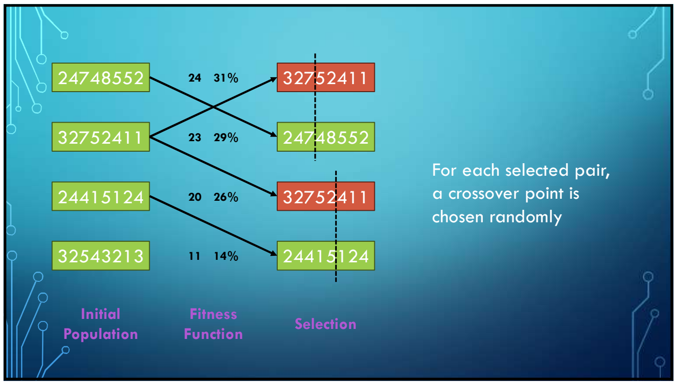
19



Initial Population	Fitness Function		Selection
24748552	24	31%	32752411
32752411	23	29%	24748552
24415124	20	26%	32752411
32543213	11	14%	24415124

Using $p = 2$ and based on the probabilities, select the parents that will reproduce

20



24748552 24 31% 32752411

32752411 23 29% 24748552

24415124 20 26% 32752411

32543213 11 14% 24415124

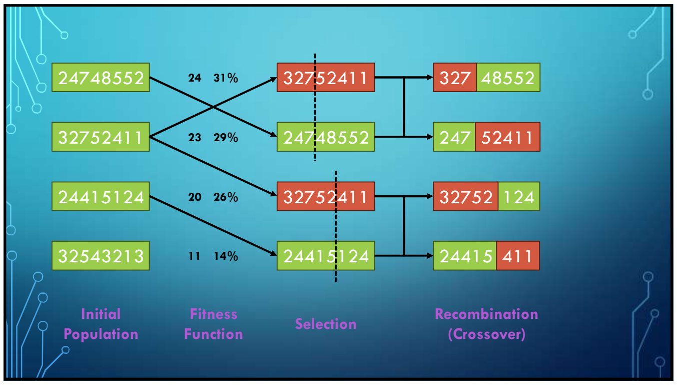
For each selected pair, a crossover point is chosen randomly

Initial Population

Fitness Function

Selection

21



24748552 24 31% 32752411 327 48552

32752411 23 29% 24748552 247 52411

24415124 20 26% 32752411 32752 124

32543213 11 14% 24415124 24415 411

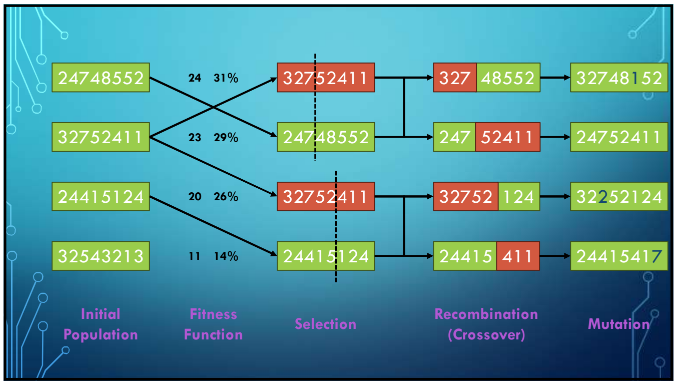
Initial Population

Fitness Function

Selection

Recombination (Crossover)

22



24748552 24 31% 32752411 327 48552 32748152

32752411 23 29% 24748552 247 52411 24752411

24415124 20 26% 32752411 32752 124 32252124

32543213 11 14% 24415124 24415 411 24415417

Initial Population

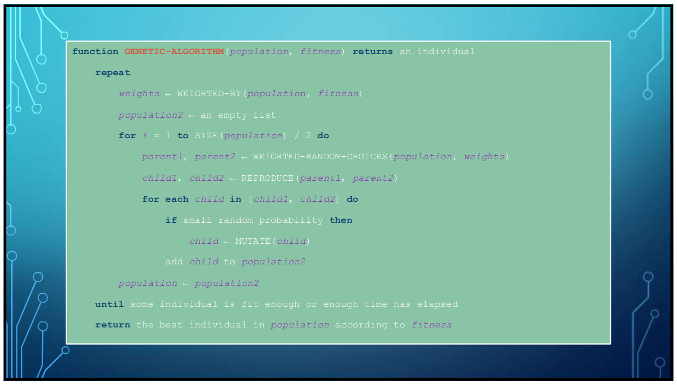
Fitness Function

Selection

Recombination (Crossover)

Mutation

23



```
function GENETIC-ALGORITHM(population, fitness) returns an individual
    repeat
        weights ← WEIGHTED-BY(population, fitness)
        population2 ← an empty list
        for i = 1 to SIZE(population) / 2 do
            parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights)
            child1, child2 ← REPRODUCE(parent1, parent2)
            for each child in [child1, child2] do
                if small random probability then
                    child ← MUTATE(child)
                add child to population2
        population ← population2
    until some individual is fit enough or enough time has elapsed
    return the best individual in population according to fitness
```

24