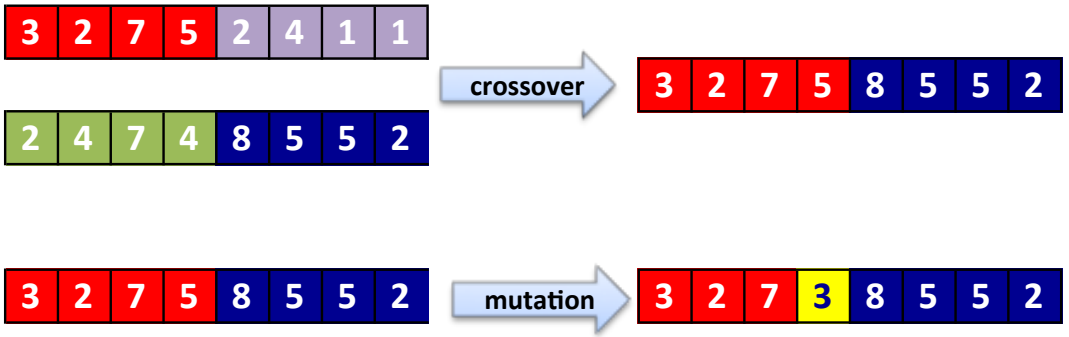
Artificial Intelligence

Search Agents

Local search



Local search

- Search algorithms seen so far are designed to **explore search spaces systematically**.
- Problems: observable, deterministic, known environments where the solution is a sequence of actions.
- Real-World problems are more complex.
- When a goal is found, the path to that goal constitutes a solution to the problem. But, depending on the applications, the path may or may not matter.
- If the path does not matter/systematic search is not possible, then consider another class of algorithms.

Local search

- In such cases, we can use iterative improvement algorithms, **Local search**.
- Also useful in pure **optimization problems** where the goal is to find the best state according to an **optimization function**.
- **Examples:**
  - Integrated circuit design, telecommunications network optimization, etc.
  - N-puzzle or 8-queen: what matters is the final configuration of the puzzle, not the intermediary steps to reach it.

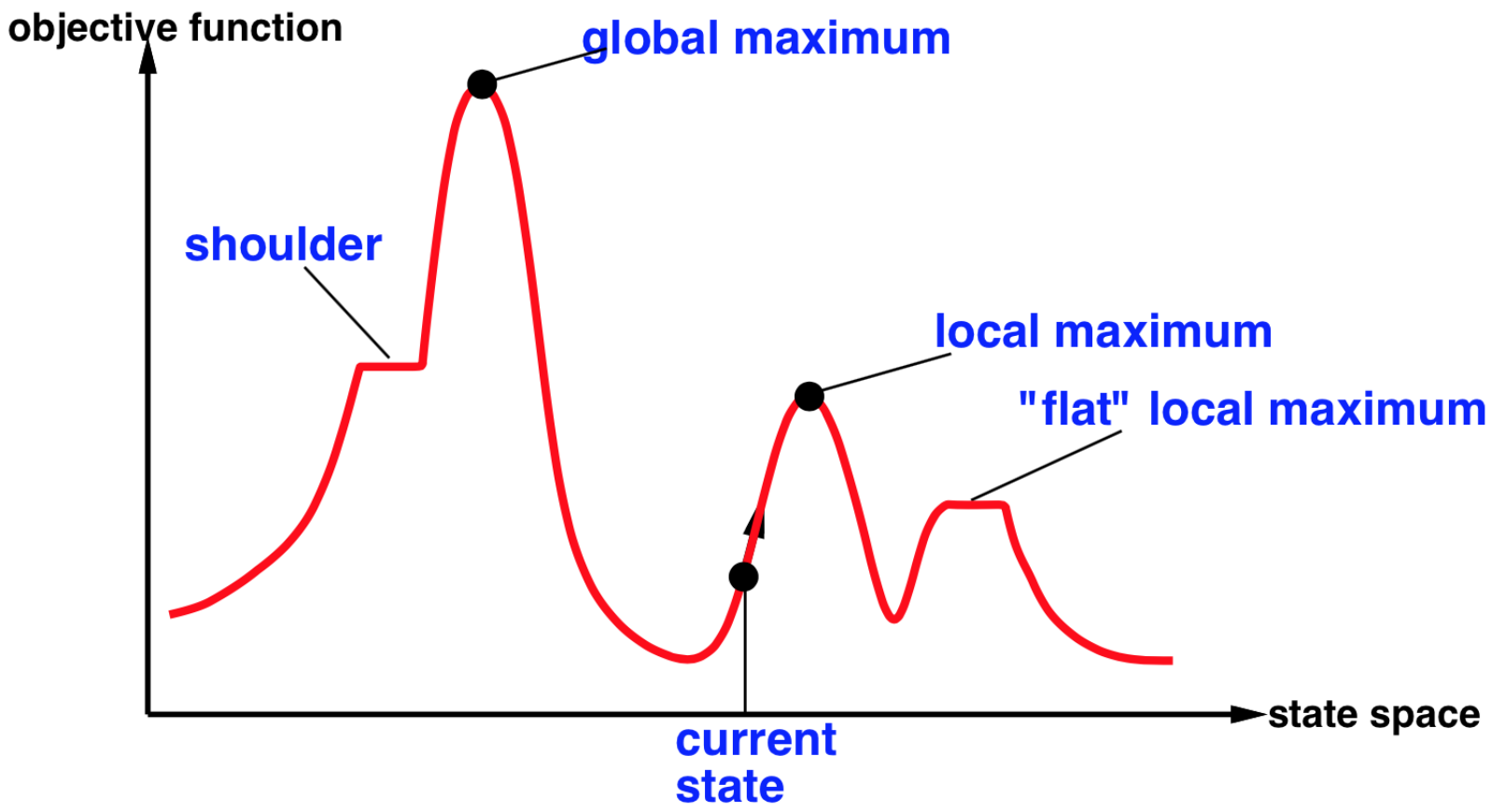
Local search

- **Idea**: keep a single "current" state, and try to improve it.
- Move only to neighbors of that node.
- **Advantages**:
  1. No need to maintain a search tree.
  2. Use very little memory.
  3. Can often find good enough solutions in continuous or large state spaces.

Local search

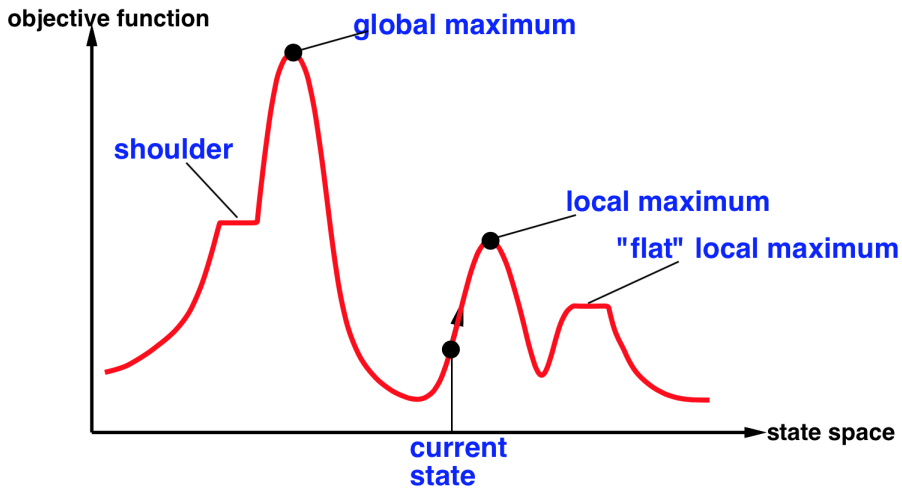
- **Local Search Algorithms:**
  - Hill climbing (steepest ascent/descent).
  - Simulated Annealing: inspired by statistical physics.
  - Local beam search.
  - Genetic algorithms: inspired by evolutionary biology.

Local search



State space landscape

Hill climbing



- Also called **greedy local search**.
- Looks only to immediate good neighbors and not beyond.
- Search moves uphill: moves in the direction of increasing elevation/value to find the top of the mountain.
- Terminates when it reaches a **pick**.
- Can terminate with a local maximum, global maximum or can get stuck and no progress is possible.
- A **node is a state and a value**.

Hill climbing

```
function HILL-CLIMBING(initialState)
    returns State that is a local maximum

    initialize current with initialState

    loop do
        neighbor = a highest-valued successor of current

        if neighbor.value ≤ current.value:
            return current.state

        current = neighbor
```

Hill climbing

Other variants of hill climbing include

- **Sideways moves** escape from plateaux where best successor has same value as the current state.
- **Random-restart** hill climbing overcomes local maxima: keep trying! (either find a goal or get several possible solution and pick the max).
- **Stochastic** hill climbing chooses at random among the uphill moves.

Hill climbing

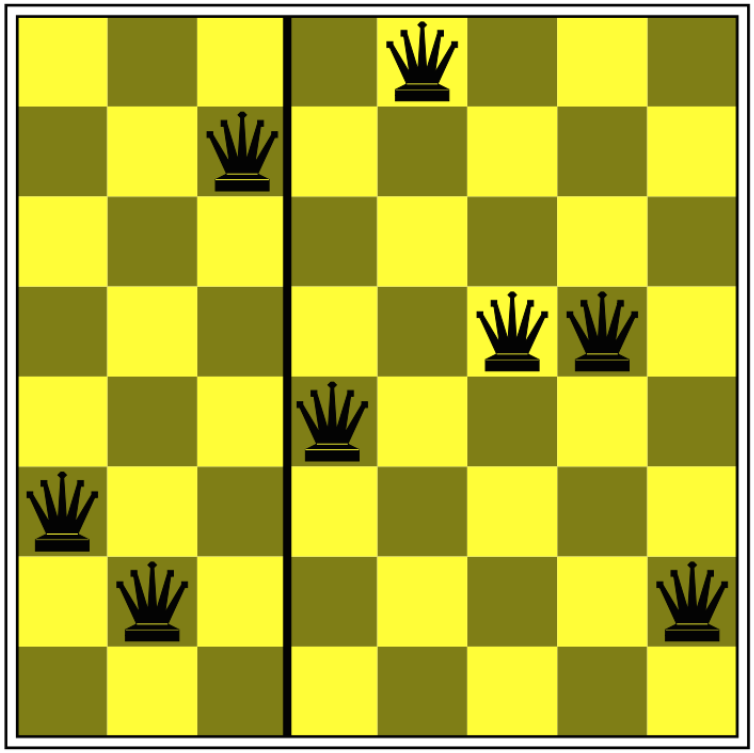
- **Hill climbing** effective in general but depends on shape of the landscape.
- Successful in many real-problems after a reasonable number of restarts.
- **Local beam search** maintains k states instead of one state.
- Select the k best successor, and useful information is passed among the states.
- **Stochastic beam search** choose k successors are random.
- Helps alleviate the problem of the the states agglomerating around the same part of the state space.

Genetic algorithms

- **Genetic algorithms (GA)** is a variant of stochastic beam search.
- Successor states are generated by combining two parents rather by modifying a single state.
- The process is inspired by **natural selection**.
- Starts with k **randomly generated states**, called **population**. Each state is an **individual**.
- An individual is usually represented by a **string** of 0's and 1's, or digits, a finite set.
- The objective function is called **fitness function**: better states have high values of fitness function.

Genetic algorithms

- In the 8-queen problem, an individual can be represented by a **string** digits 1 to 8, that represents the position of the 8 queens in the 8 columns.



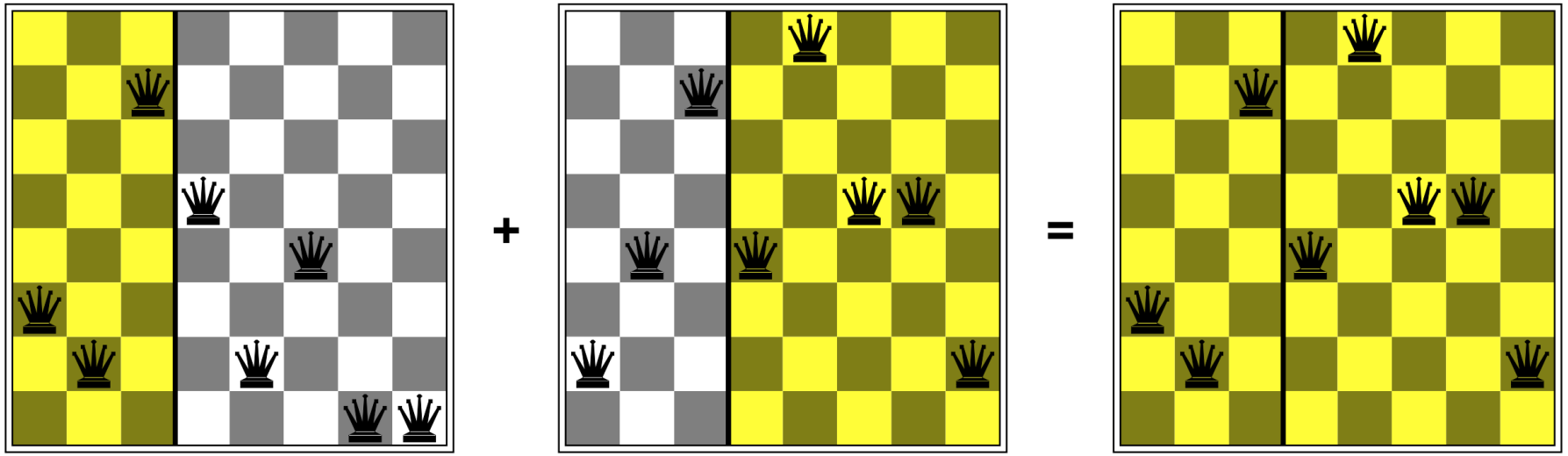
Genetic algorithms

- The objective function is called **fitness function**: better states have high values of fitness function.
- Possible fitness function is the **number of non-attacking pairs of queens**.
- Fitness function of the solution: 28.

Genetic algorithms

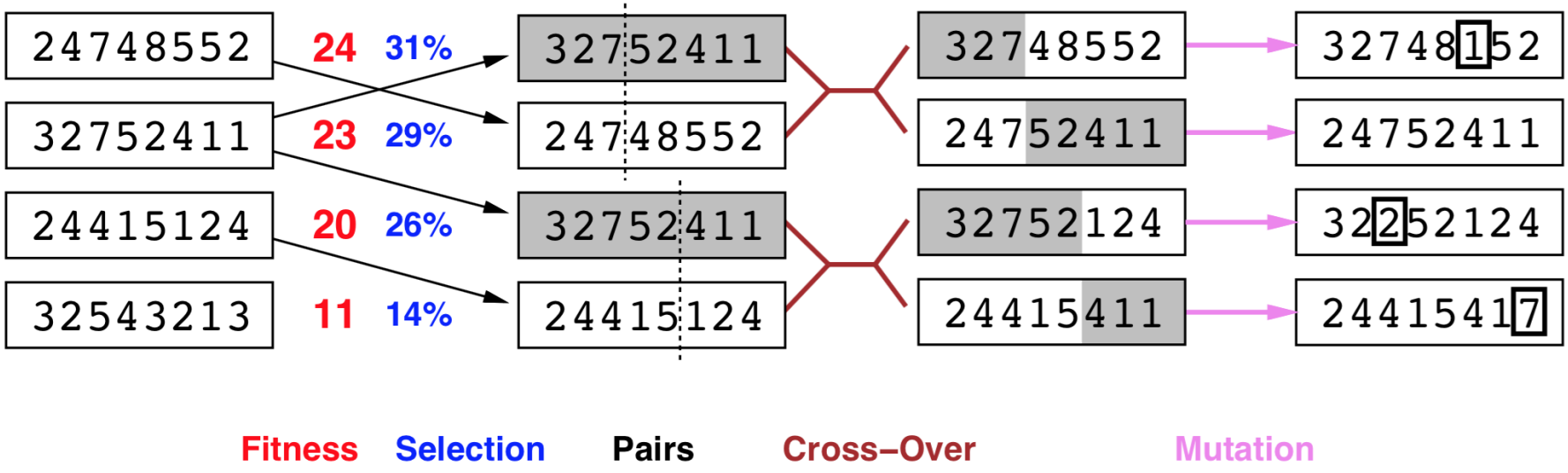
- Pairs of individuals are selected at random for **reproduction** w.r.t. some probabilities.
- A **crossover** point is chosen randomly in the string.
- **Offspring** are created by crossing the parents at the crossover point.
- Each element in the string is also subject to some **mutation** with a small probability.

Genetic algorithms



Genetic algorithms

Generate successors from pairs of states.



Genetic algorithms

function GENETIC-ALGORITHM(population, fitness-function)
returns an individual

repeat

initialize new-population **with** $\emptyset$

for i=1 to size(population) **do**

x = random-select(population,fitness-function)
x = random-select(population,fitness-function)
child = cross-over(x,y)
mutate (child) with a small random probability
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 w.r.t. fitness-function

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

http://aima.cs.berkeley.edu/