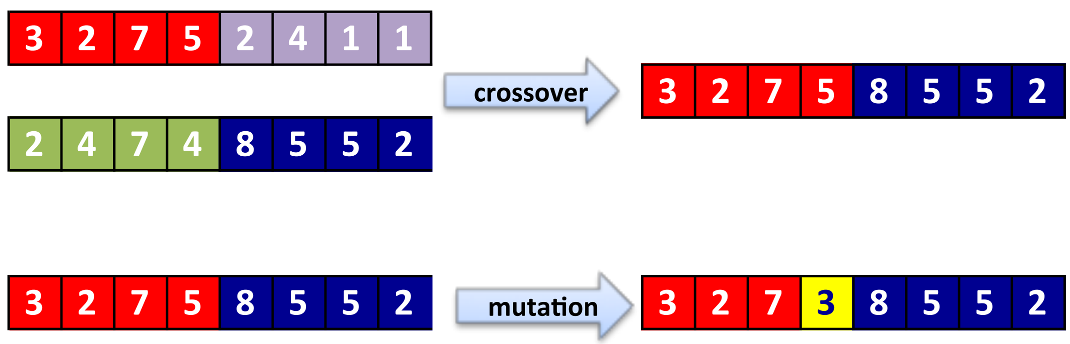
Artificial Intelligence

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.

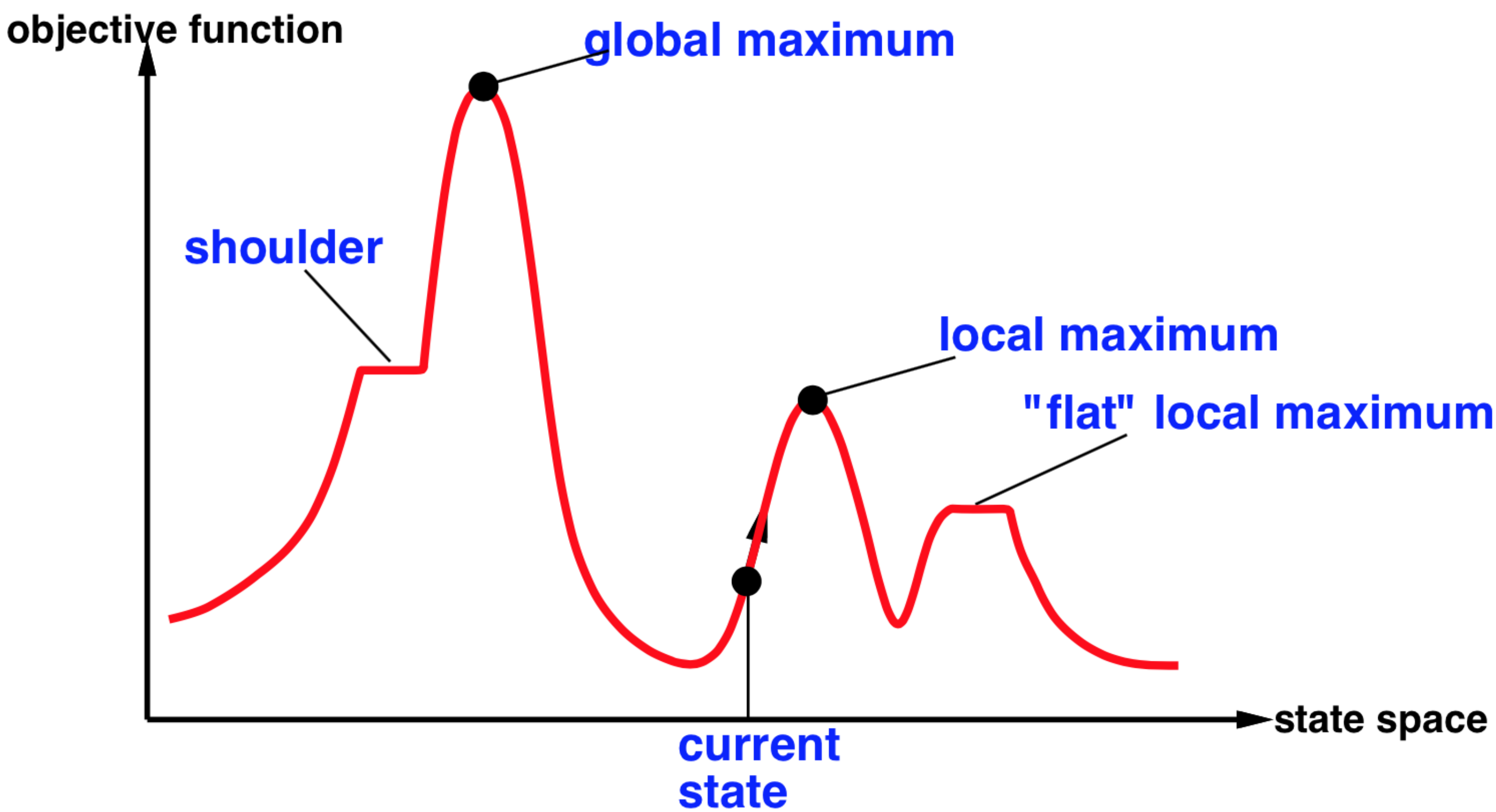
Local search

- **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

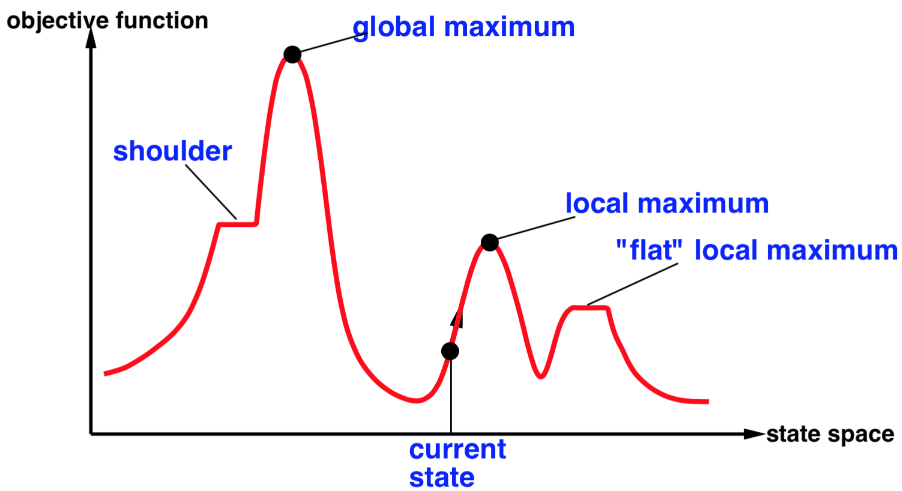
- **Local Search Algorithms:**
  - Hill climbing (steepest ascent/descent).
  - Simulated Annealing: inspired by statistical physics.
  - Local beam search.
  - Genetic algorithms: inspired by evolutionary biology.

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 in a plateau, 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 $\leq$ current.value:
            return **current.state**

        current = neighbor

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 $\leq$ current.value:
return **current.state**

current = neighbor

Question: How to change the algorithm to search for the global minimum instead?

Hill climbing

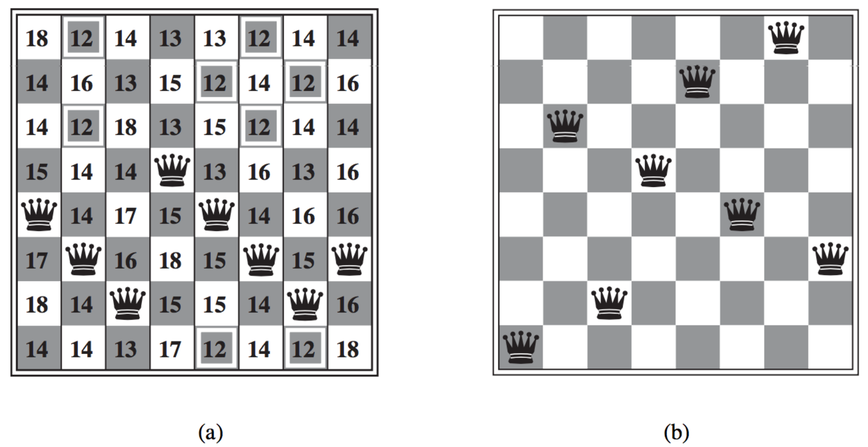


Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

- State space has $8^8 \approx 17$ million states.
- Each state has $8 \times 7 = 56$ successors.
- Heuristic function: #queens attacking each other.
- Global minimum $= 0$ (here steepest descent).
- It takes only 5 steps to go from (a) to (b) (h=17 to h=1), after that the algorithm get stuck in a local minima.

Hill climbing

Other variants of hill climbing include

- **Sideways moves** escape from plateaux, in the hope it is a shoulder. Has to be a limited number to avoid infinite loops.

- **Random-restart** hill climbing overcomes local maxima: keep trying! (either find a goal or get several possible solution and pick the max). Effective for 8-queen problem.

- **Stochastic** hill climbing chooses at random among the uphill moves. Slower than hill climbing but finds better solutions in some state landscapes.

Hill climbing

- **Hill climbing** effective in general but depends on shape of the landscape (few local maxima/minima and plateaux). Successful in many real-problems after a reasonable number of restarts. Also efficient, sometimes takes few steps.

- **Local beam search** maintains k states instead of one state. Same as random restart?

- Not really. Select the k best successor, and useful information is passed among the states about better areas for search.

- **Stochastic beam search** choose k successors are random.

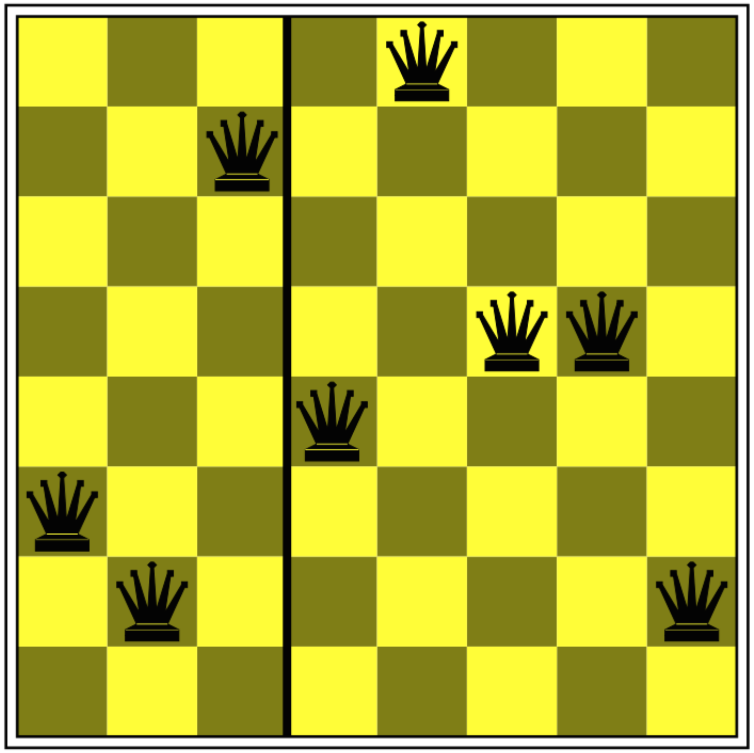
- Helps alleviate the problem of the the states agglomerating around the same part of the state space.

Genetic algorithms

- **Genetic algorithms (GA)** is a variant of stochastic beam search.
- But, 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

Fitness function

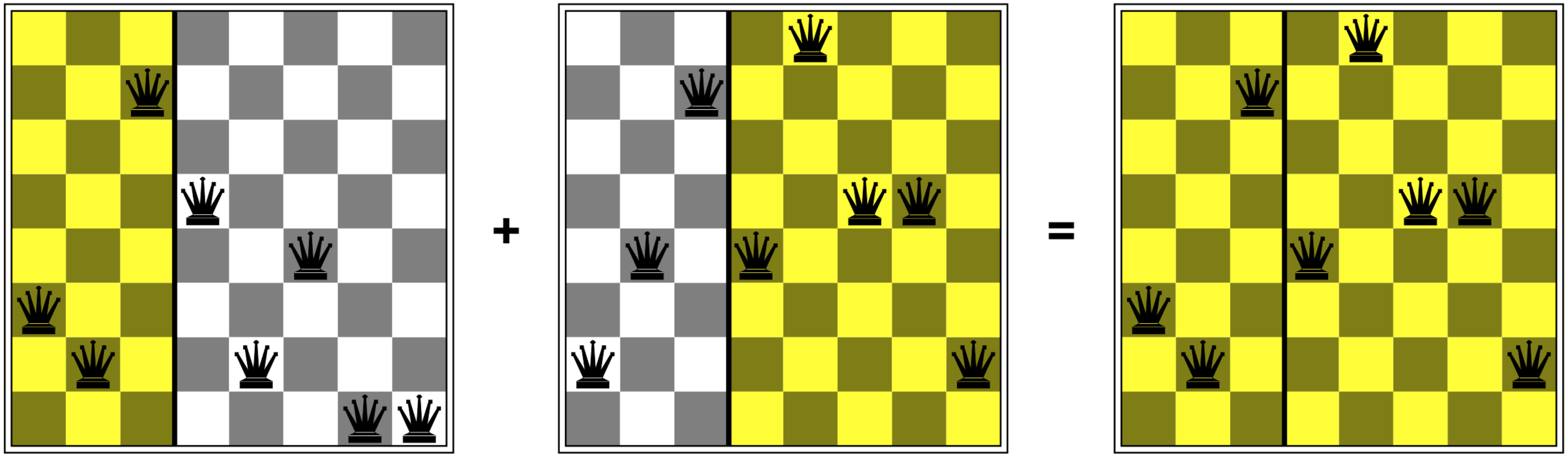
- The objective function is called **fitness function**: better states have high values of fitness function.
- Possible fitness function is the **number of non-attacking pairs of queens**.
- Fitness function of the solution: $7 + 6 + 5 + 4 + 3 + 2 + 1 = 28$.

Genetic algorithms

Genetic operators

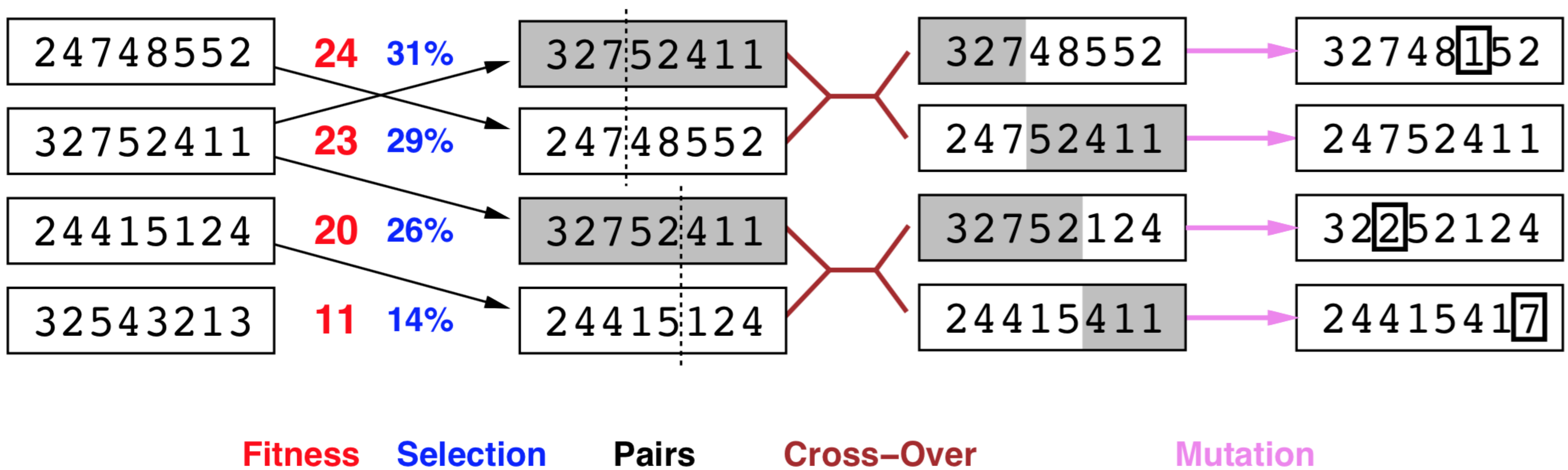
- Pairs of individuals are selected at random for **reproduction** w.r.t. some probabilities.
- A **crossover** point is chosen randomly in the string.
- **Offspring** are created by crossing the parents at the crossover point.
- Each element in the string is also subject to some **mutation** with a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

Genetic algorithms



Genetic algorithms

Generate successors from pairs of states.



- Probability to be selected is here proportional to the fitness function.
- The pairs are selected at random for reproduction (individual 2 selected twice, the 4 is dropped).
- Crossover points are picked randomly.

Genetic algorithms

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

    repeat

        initialize new-population with ∅

        for i=1 to size(population) do

            x = random-select(population,fitness-function)
            y = 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
```

Genetic algorithms

- Widely used in optimization problems: e.g., VLSI.
- Much more work to be done.

Credit

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

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