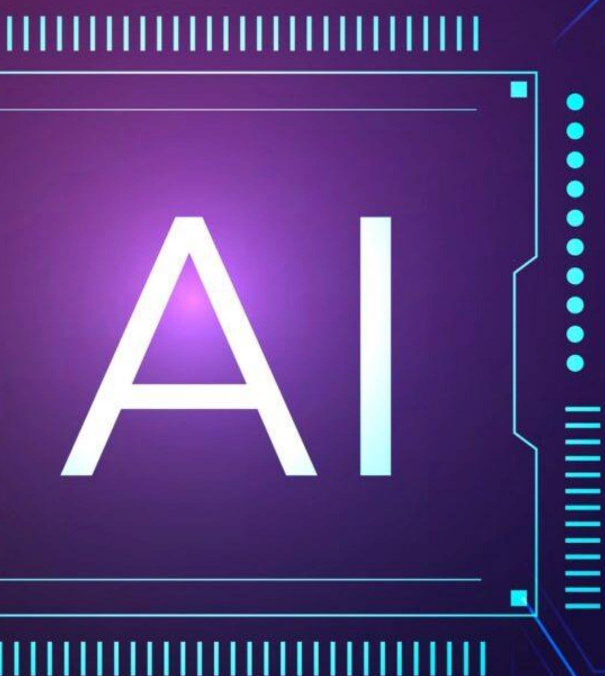


LOCAL SEARCH AND OPTIMIZATION PROBLEMS

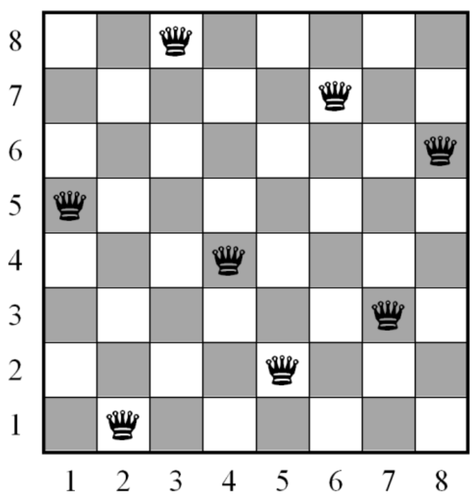
Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

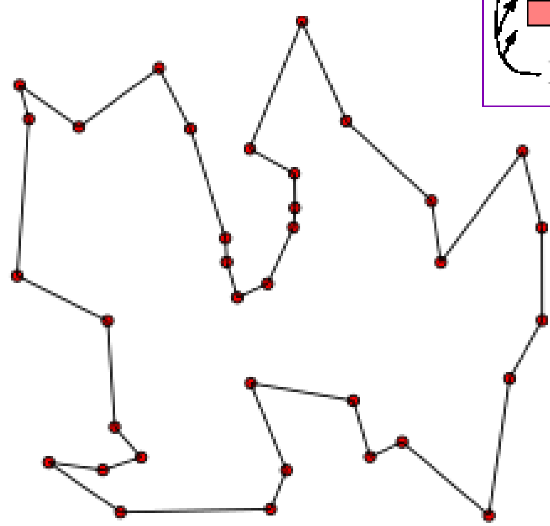
Outline

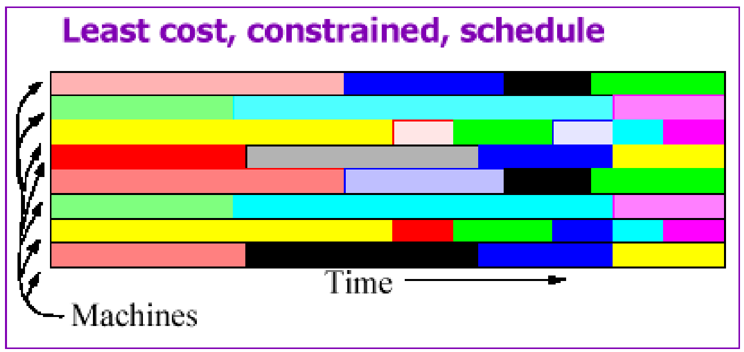
- Local search and optimization problems
- Local search algorithms
- Evolutionary algorithms

Paths are not always required

- There are applications that only the final state matters, not the path to get there.
  - Integrated-circuit design, factory floor layout, job shop scheduling, etc.







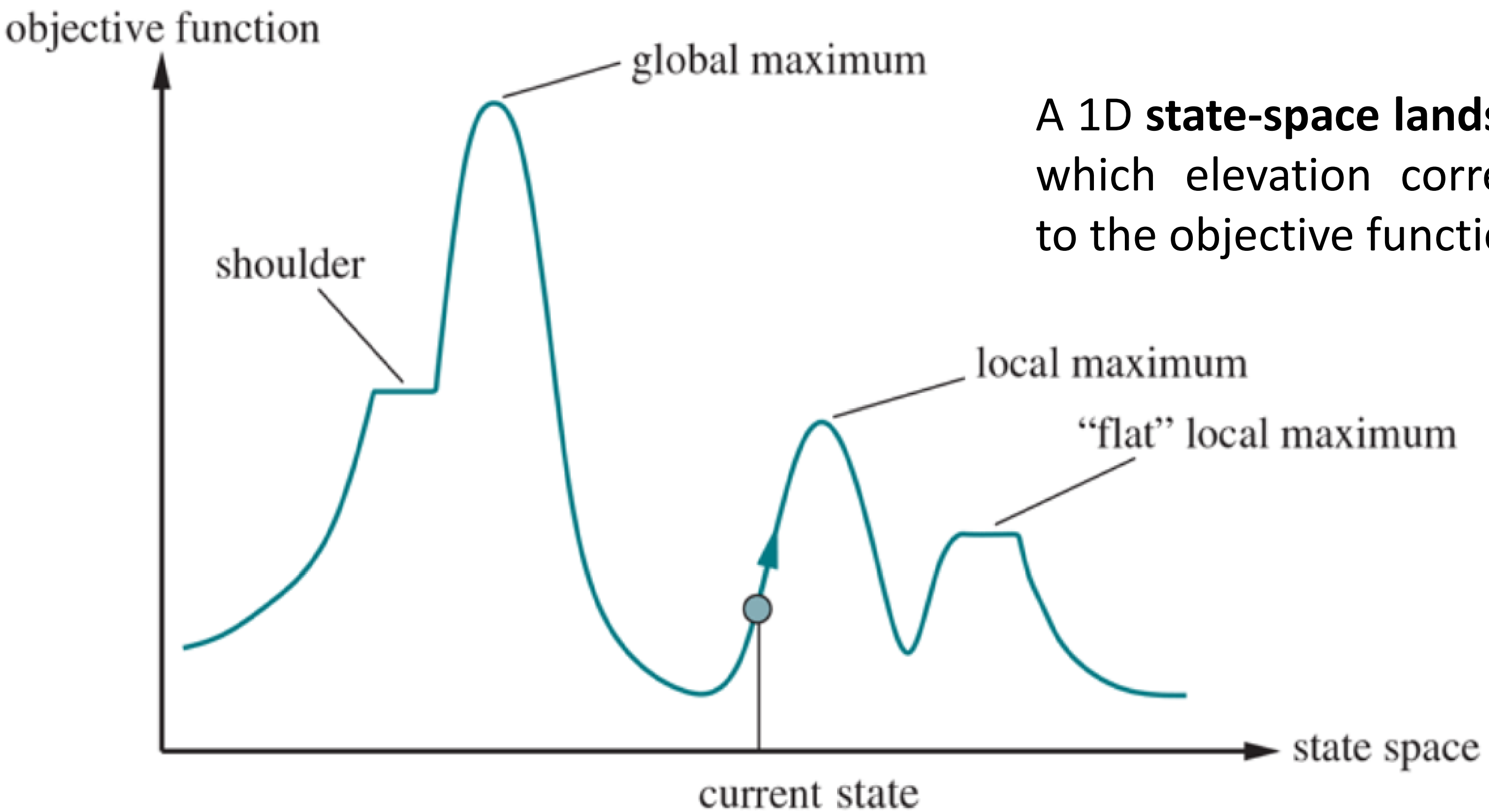
Local search algorithms

- These algorithms navigate from a start state to neighbors, without tracking the paths, nor the set of reached states.
- Not systematic
  - They might never explore a portion of the search space where a solution resides.

- Local search has two key advantages:
  - Use very little memory
  - Find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

Local search algorithms

- Local search can solve optimization problems, which finds the best state according to an objective function.



A 1D **state-space landscape** in which elevation corresponds to the objective function.

Hill climbing search

- The algorithm heads in the direction that gives the steepest ascent and terminates when it reaches a “peak”.
  - Peak = a state where no neighbor has a higher value.

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
    current ← problem.INITIAL
    while *true* **do**
        neighbor ← a highest-valued successor of *current*
        if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
        current ← *neighbor*

Hill-climbing tracks only one current state and, on each iteration, moves to the neighboring state with highest value.

Hill climbing for 8-queens problem

- Complete-state formulation: all queens on the board, one per column
- Successor function: move a queen to another square in the same column → each state has $8 \times 7 = 56$ successors
- $h(n) =$ the number of pairs of queens that are attacking each other → the global minimum has $h(n)$=0

The board shows the value of for each possible successor obtained by moving a queen within its column.

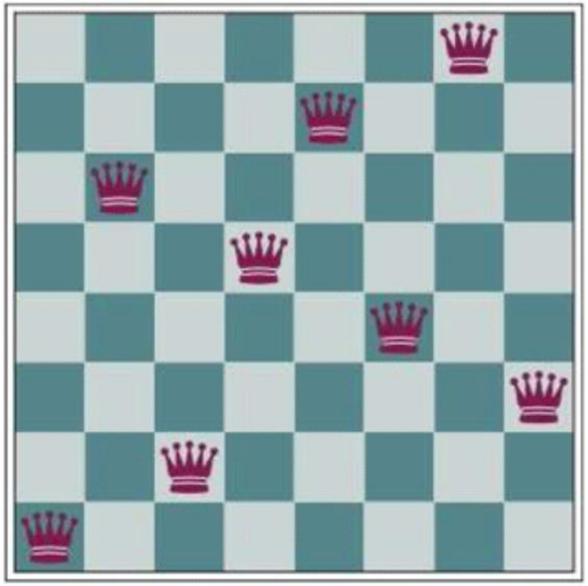
The current state n has $h(n) = 17$
$(c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8) = (4\ 3\ 2\ 5\ 4\ 3\ 2\ 3)$

There are 8 moves that are tied for best, with h = 12. Hill climbing algorithm will pick one of these.

Hill climbing: Suboptimality

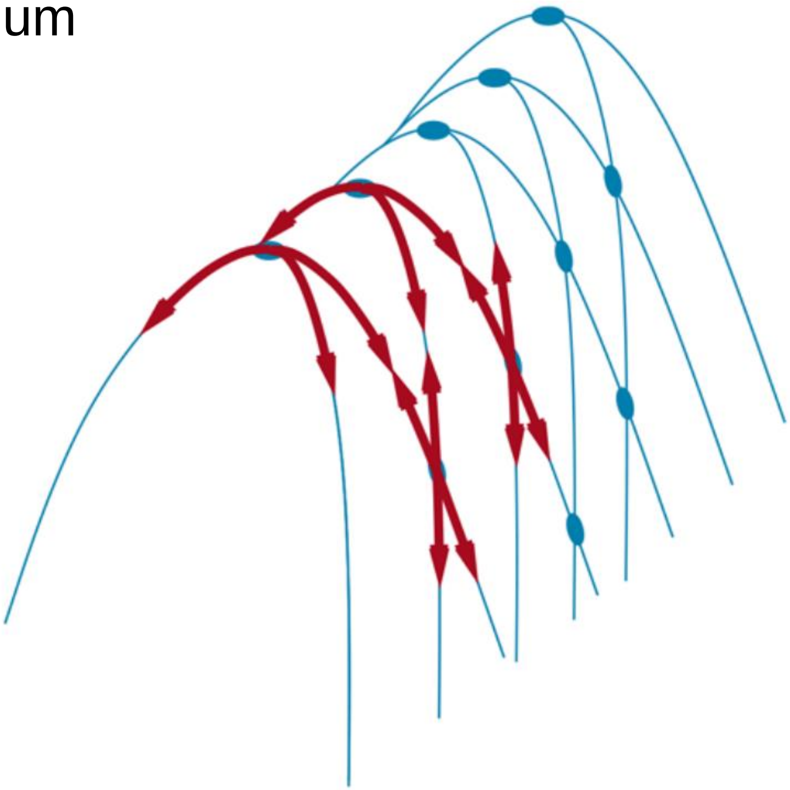
- Hill climbing is also called greedy local search.
  - It grabs a good neighbor state without thinking where to go next.
- It can easily improve a bad state and hence make rapid progress toward a solution.
  - 8-queens: 17 million states, 4 steps on average when succeeds and 3 when get stuck.
- Hill climbing can get stuck in local maxima, ridges, or plateaus

Local extrema and ridges



- Current state $(1\ 6\ 2\ 5\ 7\ 4\ 8\ 3)$ has $h(n) = 1$
- Every successor has a higher cost

  $\rightarrow$ local minimum



The grid of states (dark circles) is laid on a ridge rising from left to right, creating a sequence of local maxima

From each local maximum, all the available actions point downhill.

Local extrema and ridges

- Real-world problem or NP-hard problems typically have an exponential number of local maxima to get stuck on.

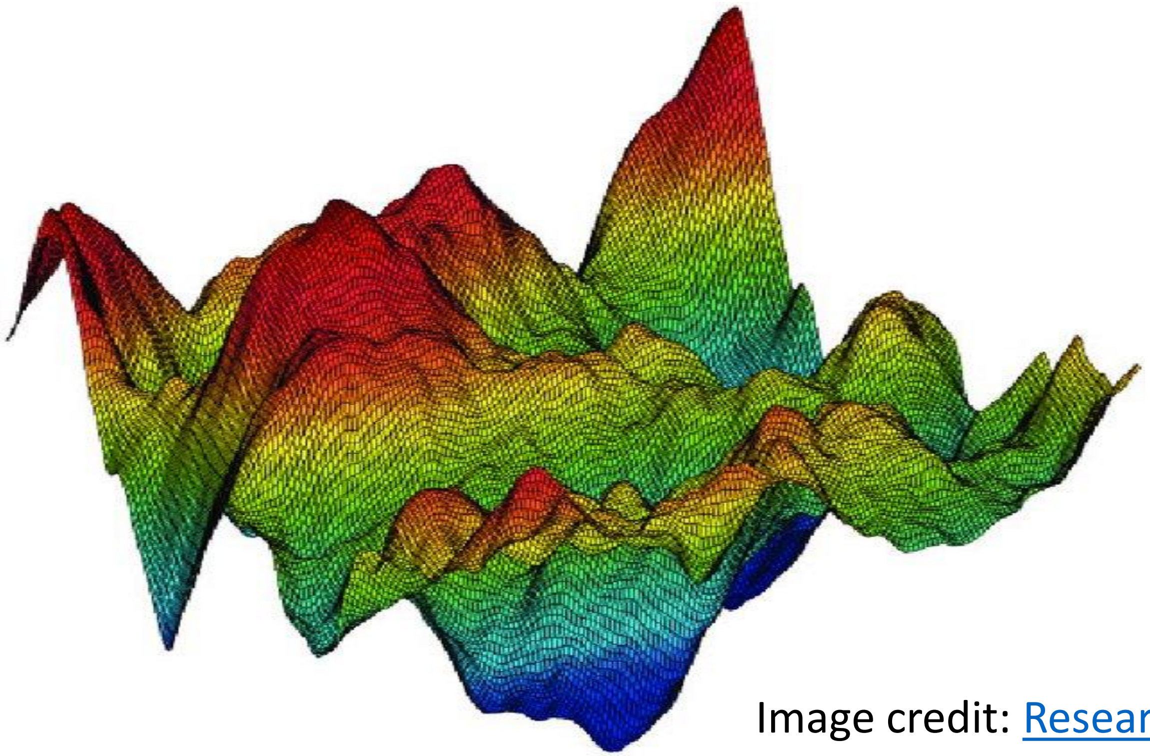


Image credit: [Research Gate]

Overcome the suboptimality

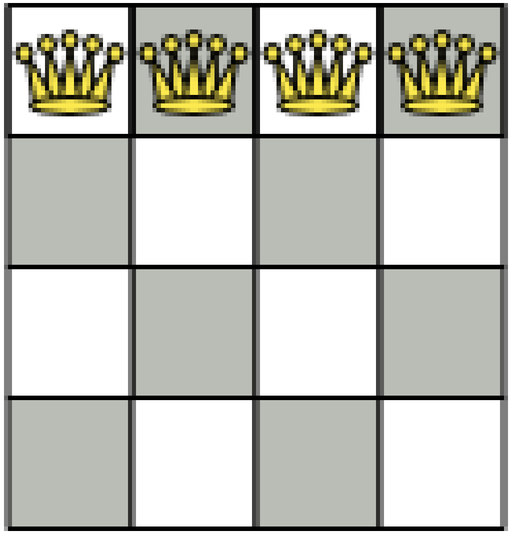
- A sideways move lets the agent keep going in the plateau.
  - It does not work on a flat local maximum.
- The number of consecutive sideways moves should be limited to avoid non-stop wander.
- This approach raises the percentage of problem instances solved by hill climbing.
  - E.g., for 8-queens problem: from 14% to 94%.
  - Success comes at a cost: roughly 21 steps for each successful instance and 64 for each failure

Overcome the suboptimality

- Stochastic hill climbing chooses at random from among the uphill moves.
  - The probability of selection can vary with the steepness of the move.
  - Slower convergence, yet better solutions in some state landscapes
- First-choice hill climbing generates successors randomly until obtaining one that is better than the current state.
  - Suitable when a state has many (e.g., thousands) of successors
- Random-restart hill climbing conducts several hill-climbing searches from random initial states, until a goal is found.
  - If each search has a probability p of success, the expected number of restarts required is $1/p$.

Quiz 01: 4-queens problem

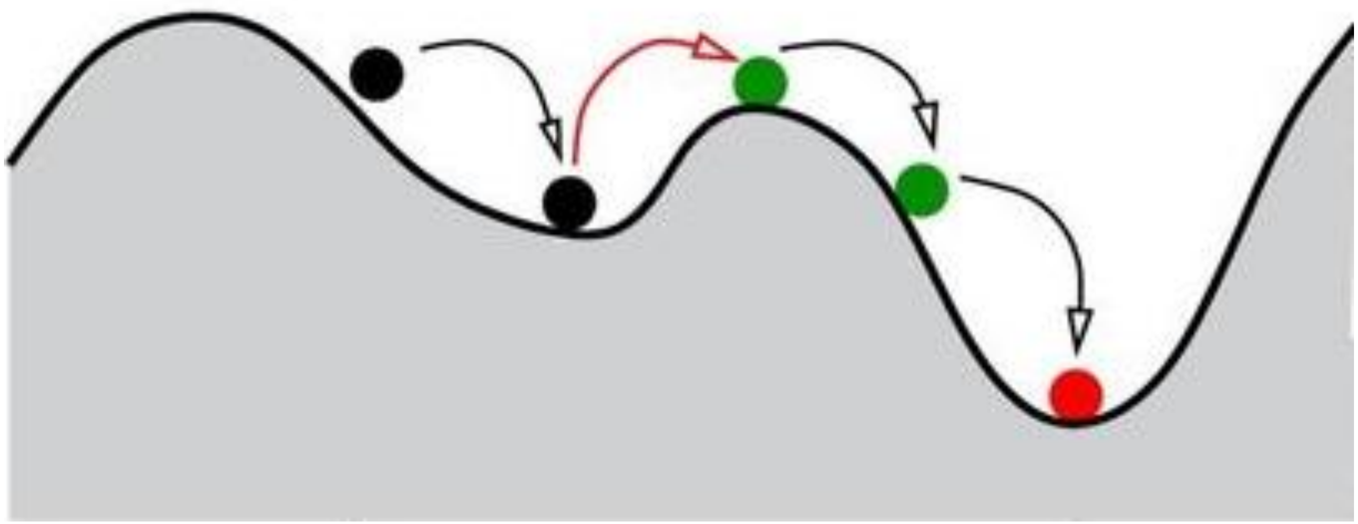
- Consider the following 4-queens problem



- Apply hill-climbing to find a solution, using the heuristic "*The number of pairs of queens attacking each other.*"

Simulated annealing

- Combine hill climbing with a random walk in some way that yields both efficiency and completeness

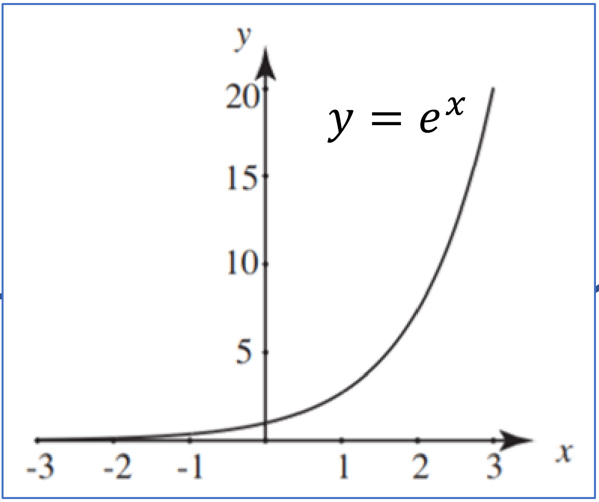


Shake hard (i.e., at a high temperature) and then gradually reduce the intensity of shaking (i.e., lower the temperature)

Simulated annealing

a mapping function from time to "temperature"

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
    current ← *problem*.INITIAL
    for t = 1 **to** ∞ **do**
        T ← *schedule*(t)
        if T = 0 **then return** *current*
        next ← a randomly selected successor of *current*
        ΔE ← VALUE(*next*) – VALUE(*current*)
        if $\Delta E > 0$ **then** *current* ← *next*
        else *current* ← *next* only with probability $e^{\Delta E/T}$

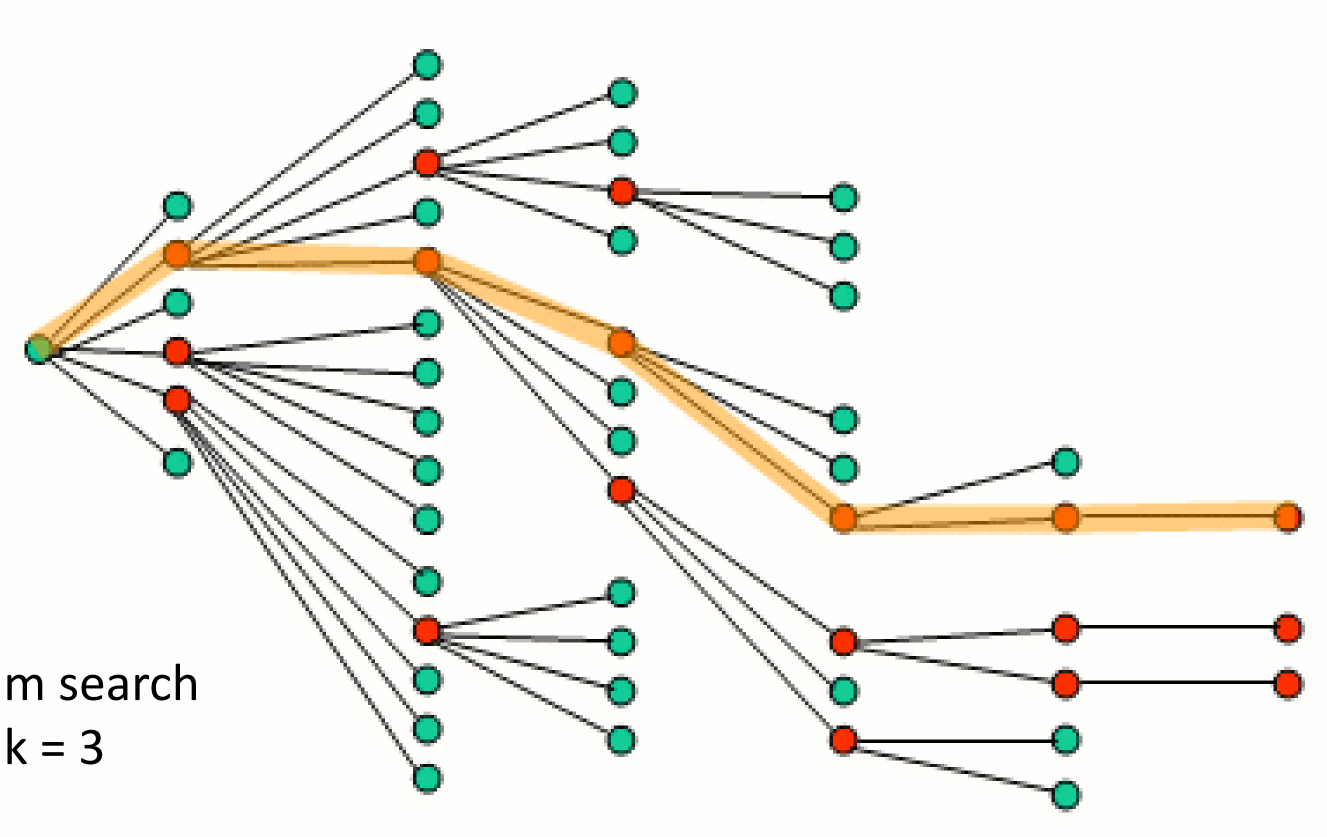


Local beam search

- Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations.

- The algorithm keeps track of k states rather than just one.
- It begins with randomly generated states.
- At each step, all the successors of all k states are generated
- If any one is a goal, the algorithm halts. Otherwise, it selects the best successors from the complete list and repeats.

Local beam search



Local beam search
with k = 3

• The algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made.

Local beam search

- Useful information is passed among the parallel search threads $\rightarrow$ major difference from random-restart search
- The algorithm possibly suffers from a lack of diversity among the k states.
  - The states can become clustered in a small region of the state space $\rightarrow$ an expensive version of hill climbing.
- Stochastic beam search can alleviate the above problem by randomly picking k successors following their values.

Evolutionary algorithms

- Variants of stochastic beam search, explicitly motivated by the metaphor of natural selection in biology

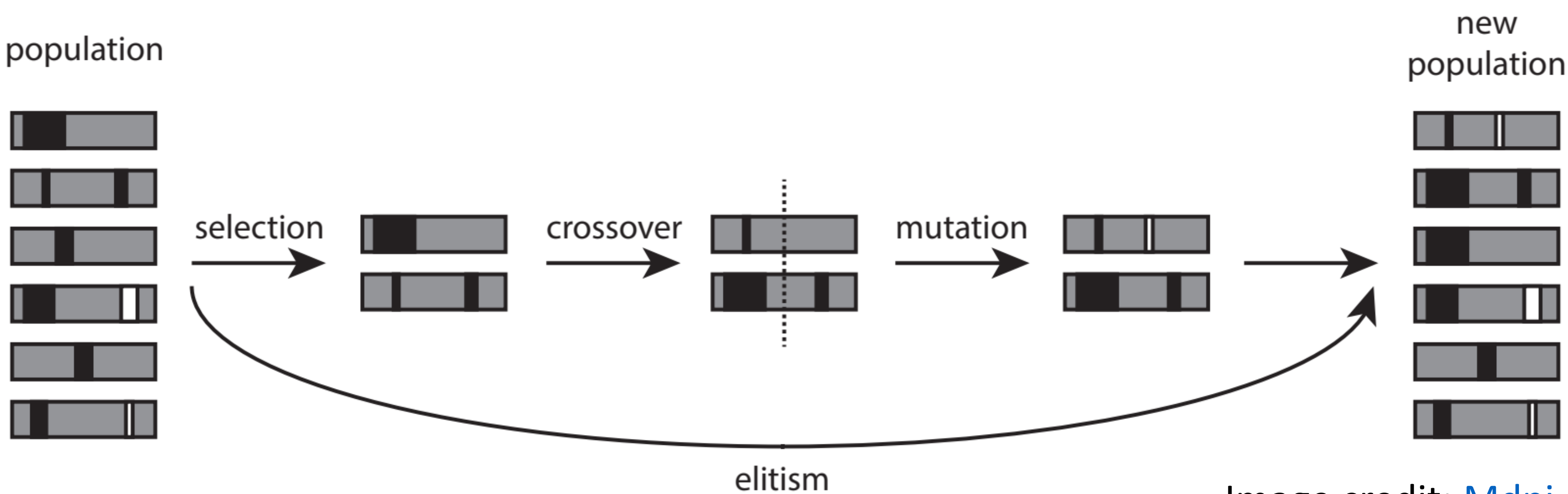
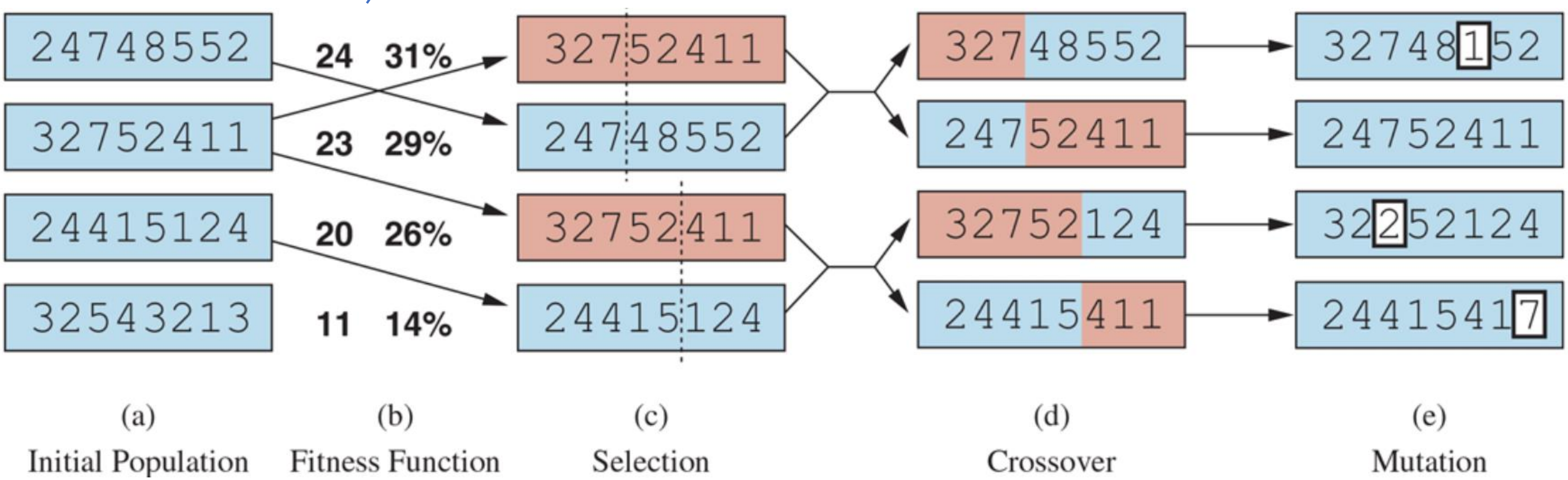


Image credit: [Mdpi]

There is a population of individuals (states). The fittest (highest value) individuals produce offspring (successor states) that populate the next generation.

Evolutionary algorithms

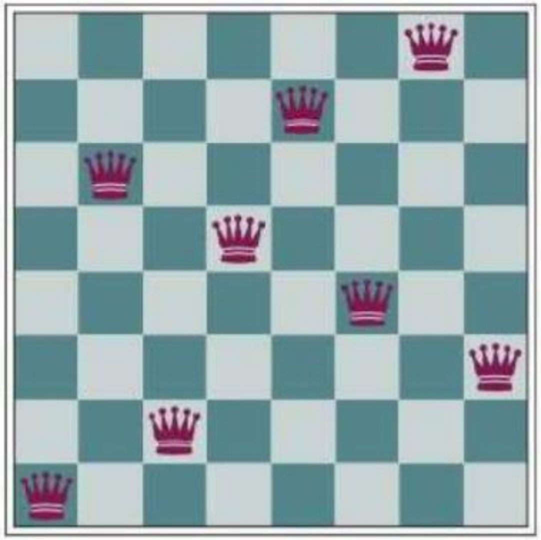
Fitness function = the number of nonattacking pairs of queens



A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Evolutionary algorithms

- There are endless forms of evolutionary algorithms.
- The representation of an individual
  - Genetic algorithms: a string over a finite alphabet.
  - Genetic programming: a computer program. Evolution strategies: a sequence of real numbers.



- Digit representation

$$\langle 1\ 6\ 2\ 5\ 7\ 4\ 8\ 3 \rangle$$

- Binary representation

$$\langle 000\ 101\ 001\ 100\ 110\ 011\ 111\ 010 \rangle$$

Evolutionary algorithms

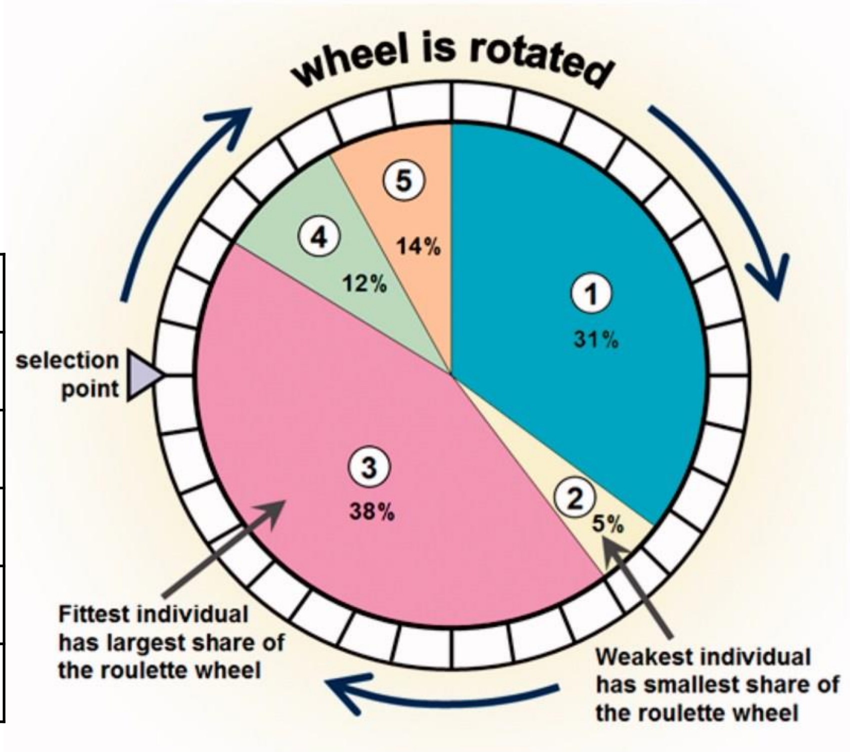
- The size of the population
  - A population is a set of k randomly generated states to begin with.
- Fitness function: an objective function that rates each state
  - The higher values, the better states
- The mixing number ρ: number of parents that come together to form offsprings
  - $\rho = 1$: stochastic beam search. $\rho = 2$: most common.

Evolutionary algorithms

- The selection process: choose individuals as parents of the next generation
  - Individuals can be chosen with probabilities proportional to their fitness score.

The Roulette wheel method

Individual	Fitness percentage (%)
1	31
2	5
3	38
4	12
5	14



Evolutionary algorithms

- The recombination procedure to form children
  - It randomly selects a crossover point to split each of the parent strings and recombines the parts to form two children.

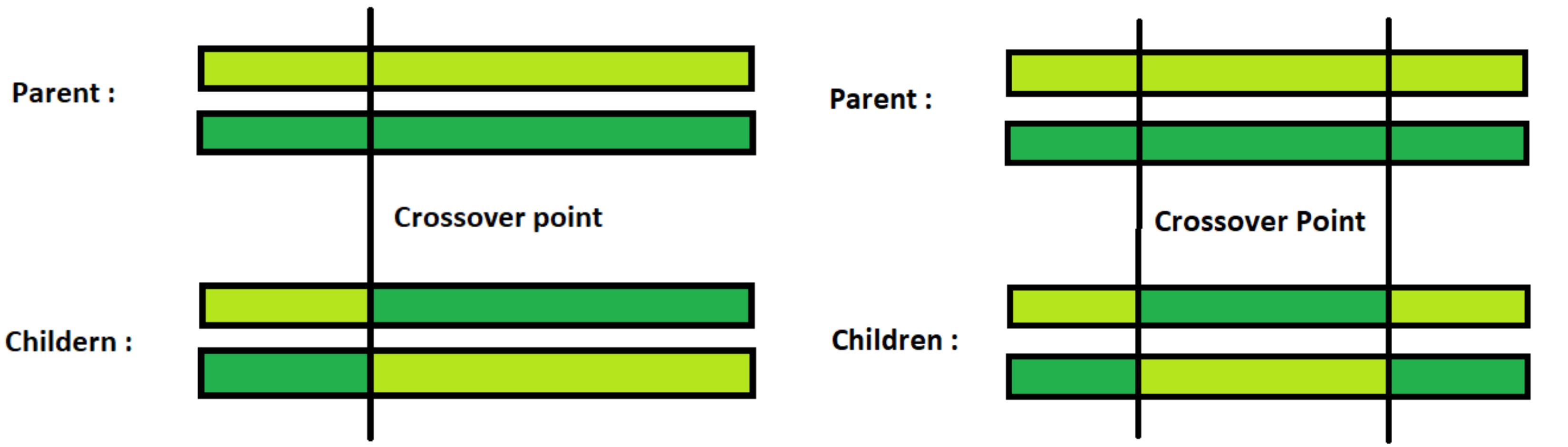
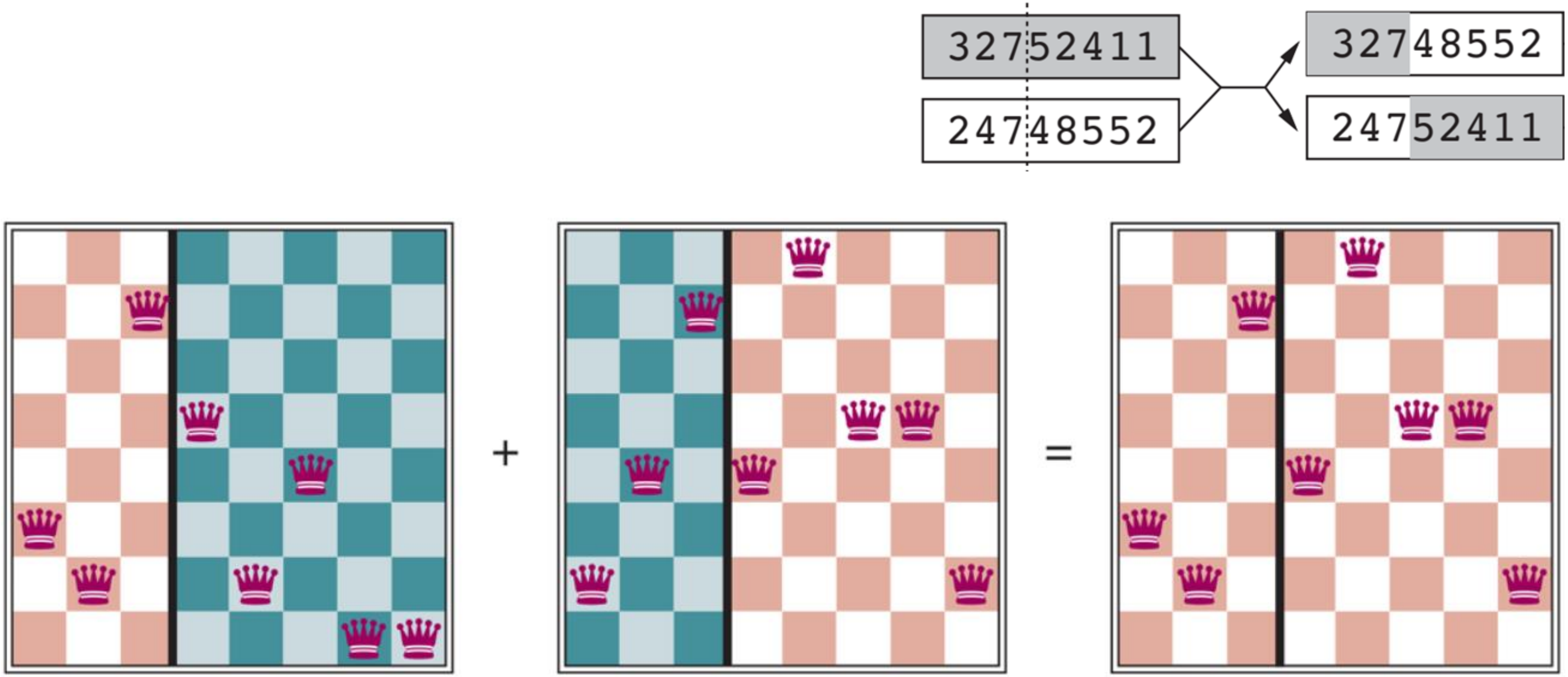


Image credit: Geeksforgeeks

- Crossover frequently takes large steps in the state space early in the search process.

Recombination step: An example



The 8-queens states corresponding to the two parents (left and middle) and the first offspring (right). The green columns are lost in the crossover step and the red columns are retained.

Evolutionary algorithms

- The mutation rate: determine how often offspring have random mutations to their representation.
  - Every bit in the individual's composition is flipped with probability equal to the mutation rate.
- The makeup of the next generation: only the newly formed offspring, or a few top-scoring parents also included.
  - Elitism practice: guarantee that overall fitness will never decrease over time
  - Culling practice: All individuals below a given threshold are discarded, which can lead to a speedup (Baum et al., 1995).

Quiz 02: Calculate fitness scores

- Consider the 4-queens problem, in which each state has 4 queens, one per column, on the board. The state can be represented in genetic algorithm as a sequence of 4 digits, each of which denotes the position of a queen in its own column (from 1 to 4).

- $\boldsymbol{Fit}(\boldsymbol{n})$ = *the number of non-attacking pairs of queens*
- Let the current generation includes 4 states:

S1 = 2341;   S2 = 2132;   S3 = 1232;   S4 = 4321.

- Calculate the value of $\boldsymbol{Fit}(\boldsymbol{n})$ for the given states and the probability that each of them will be chosen in the “selection” step.

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
     weights ← WEIGHTED-BY(population, fitness)
     population2 ← empty set
    for i = 1 to SIZE(population) do
       parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights,2)
       child ← REPRODUCE(parent1 , parent2)
       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
```

```
function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c+1, n))
```

An evaluation of Genetic algorithms

- Crossover gives better random exploration than local search.
- Rely on very little domain knowledge

- Large number of “tunable” parameters
  - Difficult to replicate performance from one problem to another
- Lack of good empirical studies comparing to simpler methods
  - Useful on some (small?) sets of problem, yet no convincing evidence that GAs are better than hill-climbing w/random restarts in general.


...the end.