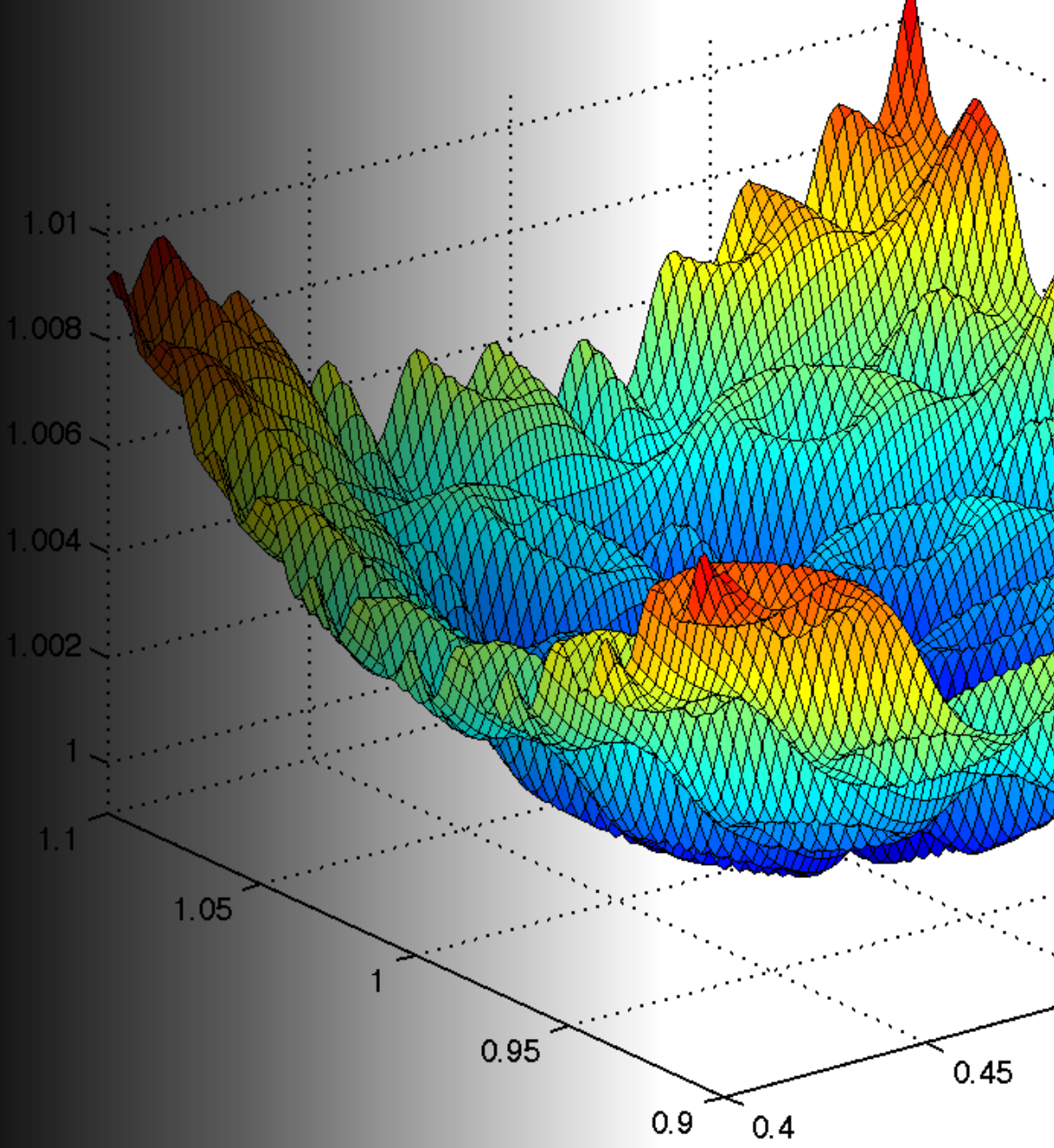
CS 5/7320
Artificial Intelligence

Local Search
AIMA Chapters 4.1 & 4.2



Slides by Michael Hahsler
based on slides by Svetlana Lazepnik
with figures from the AIMA textbook.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Recap: Uninformed Search/informed search

Tries to find the

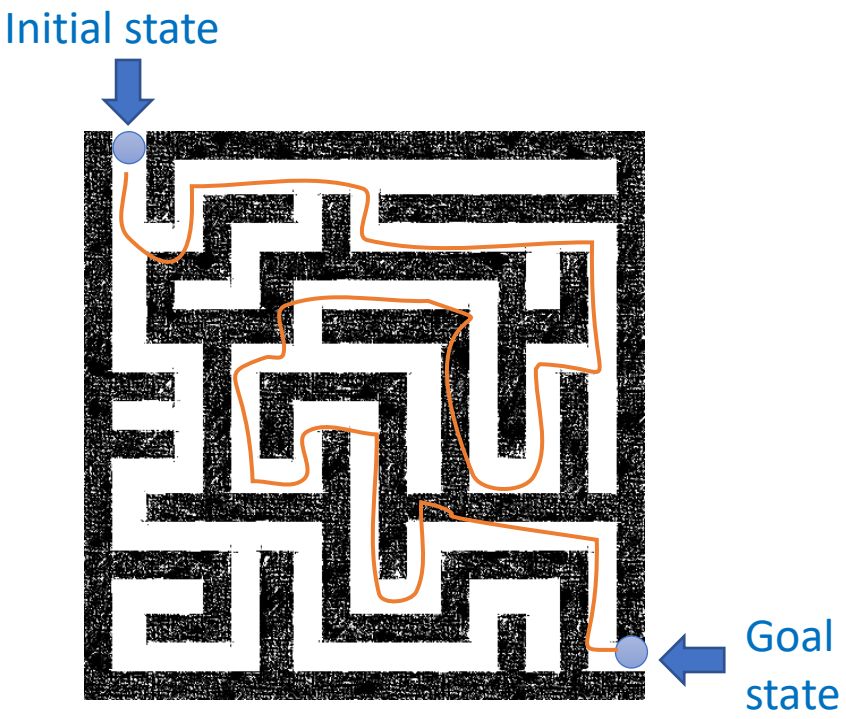
best path

from a

given initial state

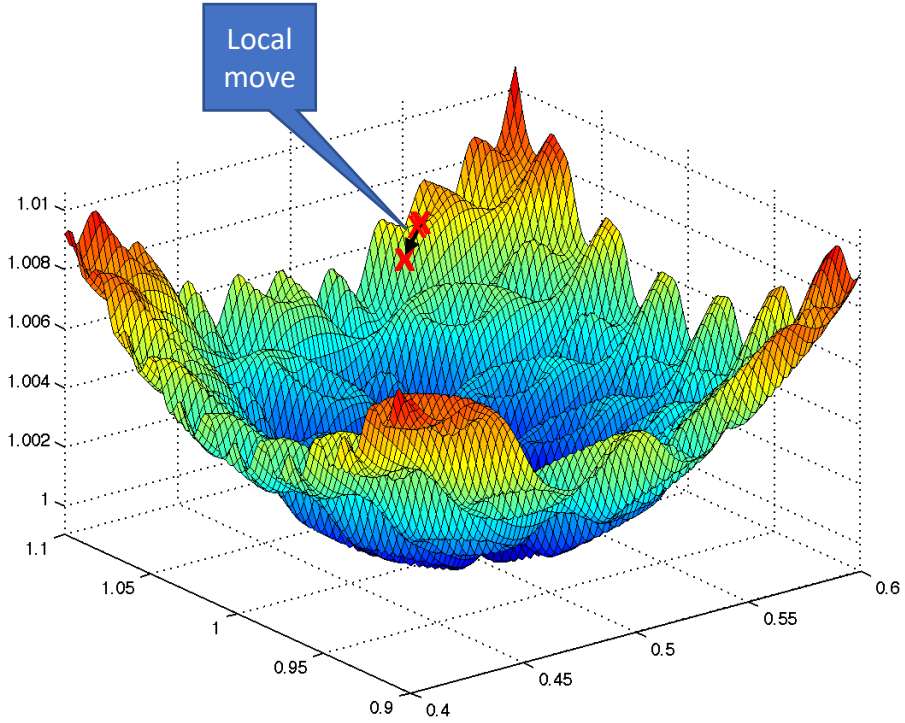
to a

given goal state.



- Typically searches a large portion of the search space (needs time and memory).
- Often comes with optimality guarantees.

Local search algorithms



- **Goal**: A fast and memory-efficient way to find a **good state.** That means to search only a small portion of the search space.
- We need an **objective function** over the states hat defines what "good" means → **optimization problem**.

Idea:

1. Formulate a solution as the state.
2. Improve the solution by moving to a neighboring state (i.e., search locally). This is fast and needs little memory (no search tree).

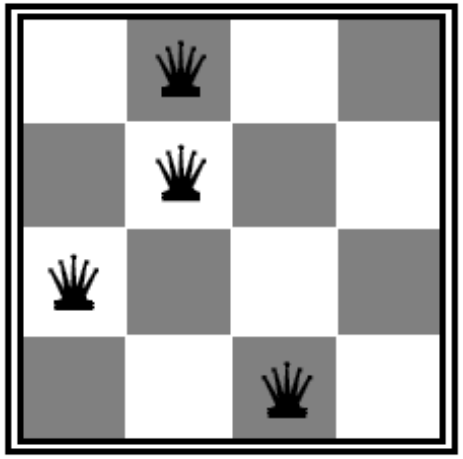
Local search algorithms

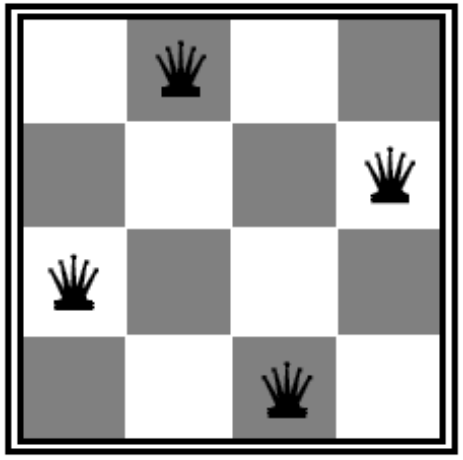
Difference to search from the previous chapter:

a) Often no explicit initial state.
b) Goal state is unknown and needs to be identified.
c) Path to goal and path cost are not important.
d) No search tree structure. Just stores the current state.

Use in AI

- **Utility-based agent**: Use utility as the objective function and always move to higher utility states. A greedy method used for complicated/large state spaces or online search.
- **Goal-based agent**: Identify a good goal state with a good objective function value before planning the path to that state.
- **General optimization**: Use for effective heuristic search in large or continuous spaces (with an infinite state space).

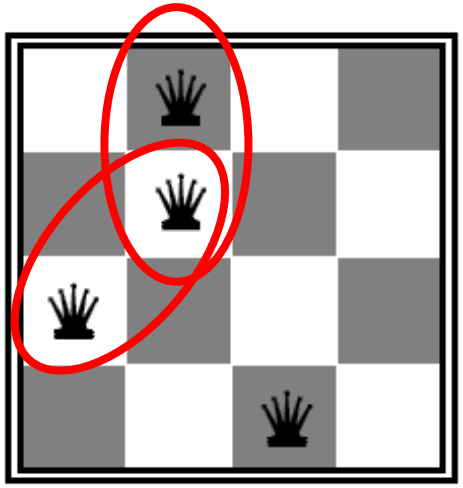


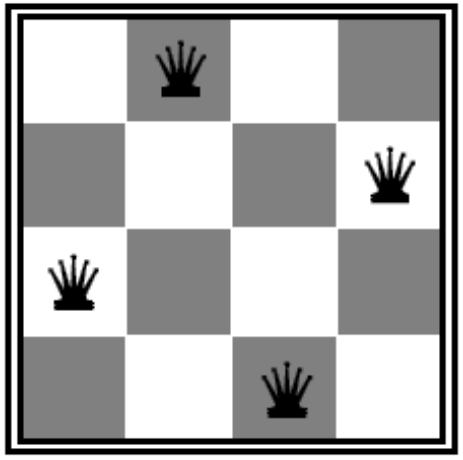


Example: *n*-queens problem

- **Goal**: Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.
- **State space:** All possible *n*-queen configurations. **How many are there?**
- What is a possible **objective function**?

2 conflicts





0 conflicts

Example: *n*-queens problem

- **Goal**: Put *n* queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

- **State space:** all possible *n*-queen configurations:
  4-queens problem: $\binom{16}{4} = 1820$

- What is a possible **objective function**?

Minimize the number of pairwise conflicts

Note: this can be seen as a heuristic used in informed search.

Example: Traveling salesman problem

- **Goal**: Find the shortest tour connecting a given set of cities
- **State space:** all possible tours (states are not individual cities!)
- **Objective function:** minimize the length of the tour

Note: We have solved a different problem with uninformed/informed search! Each city was defined as a state and the path was the solution.

Hill-Climbing Search aka Greedy Local Search

Idea: keep a single "current" state and try to find better neighboring states.

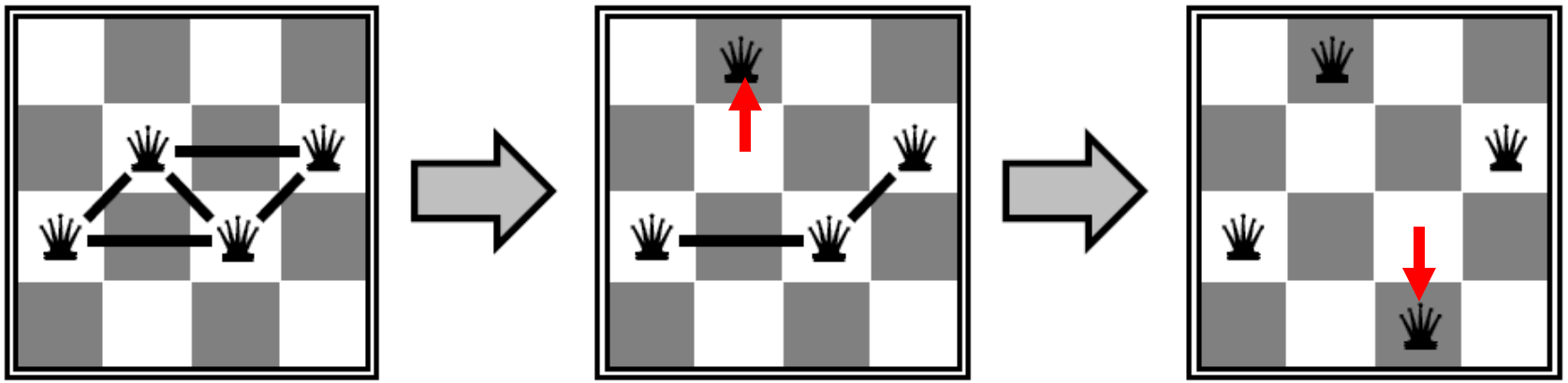
Example: *n*-queens problem

- **Goal:** Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal.
- **State space:** all possible *n*-queen configurations. We can restrict the state space: Only one queen per column.
- **Objective function:** minimize the number of pairwise conflicts.

State space is reduced from 1820 to $4^4 = 256$

What is a possible local improvement strategy?

- Move one queen within its column to reduce conflicts



Example: *n*-queens problem

- **Goal:** Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal.
- **State space:** all possible *n*-queen configurations. We can restrict the state space: Only one queen per column.
- **Objective function:** minimize the number of pairwise conflicts.

What is a possible local improvement strategy?

- Move one queen within its column to reduce conflicts

h = 17
best local improvement has h = 12

Note that there are many options, and we have to choose one!

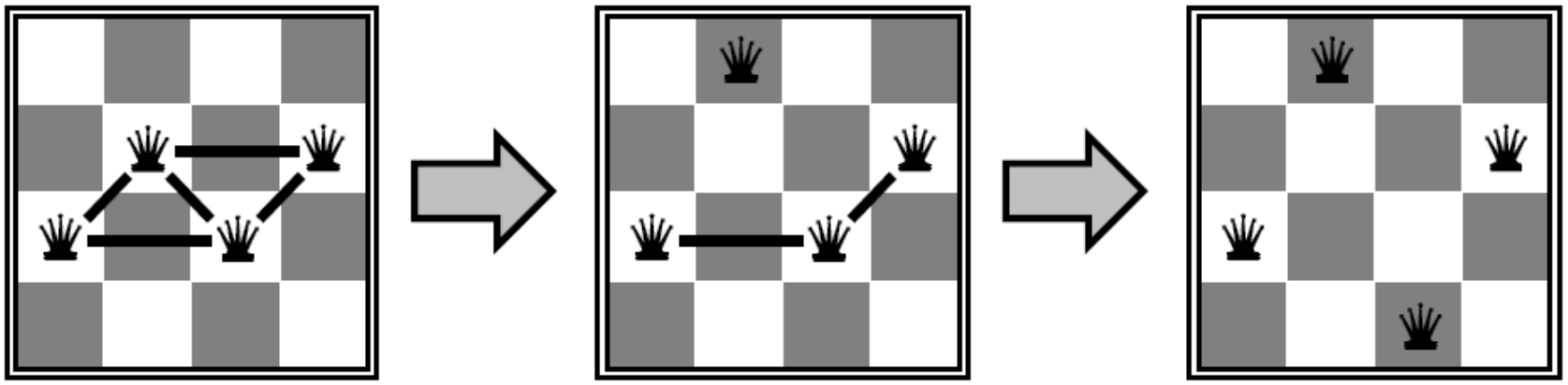
Example: *n*-queens problem

Optimization problem: find the best arrangement a

$$\text{argmin}_a \ \text{conflicts}(a)$$

s.t. a has one queen per column

This makes the problem easier.



h = 5 h = 2 h = 0

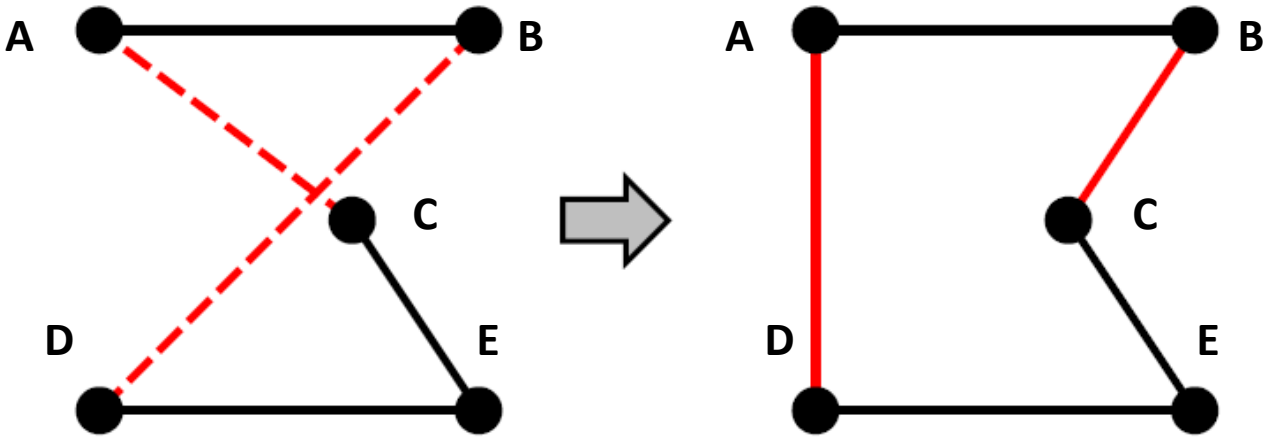
Example: Traveling Salesman Problem

- **Goal:** Find the shortest tour connecting n cities
- **State space:** all possible tours
- **Objective function:** length of tour

What's a possible local improvement strategy?

- Start with any complete tour, perform pairwise exchanges.

Permutation: **ABDEC** **ABCED**



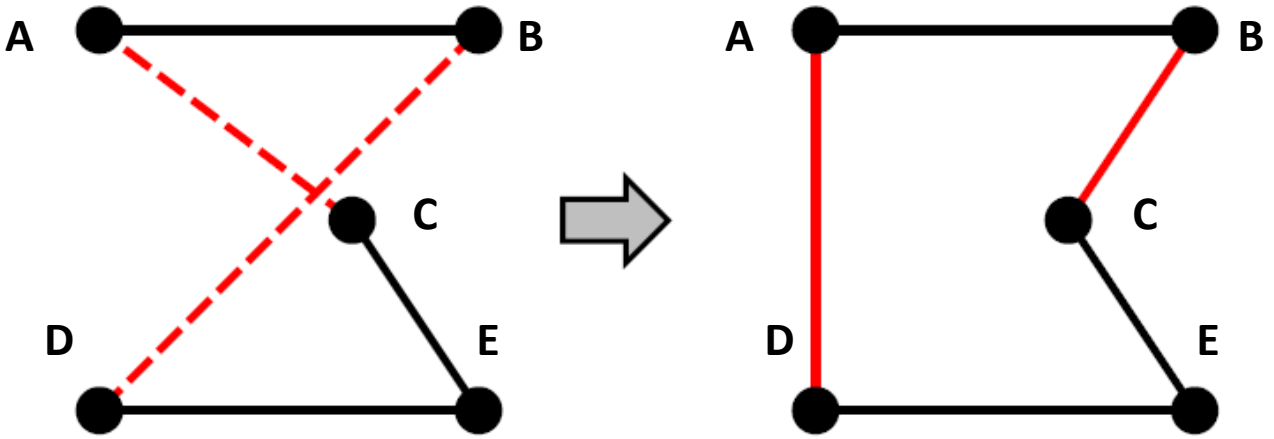
Example: Traveling Salesman Problem

Optimization problem: Find the best tour π

$$\text{argmin}_{\pi} \ \ \text{tourLength}(\pi)$$

s.t. π is a valid permutation (i.e., sub-tour elimination)

Permutation: **ABDEC** **ABCED**



Hill-climbing search (= Greedy local search)

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

Variants:

- **Steepest-ascend hill climbing**
  - Check all possible siccessors and choose the highest-valued successors.
- **Stochastic hill climbing**
  - choose randomly among all uphill moves, or
  - generate randomly one new successor at a time until a better one is found (= first-choice hill climbing – the most popular variant)

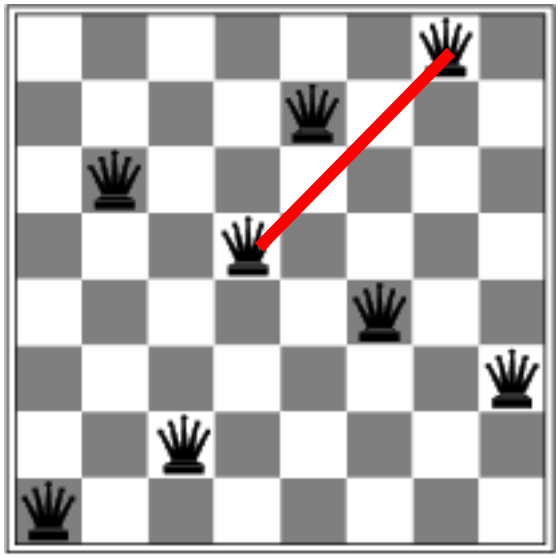
Random-restart hill climbing – to deal with local optima

Hill-climbing search

Hill-climbing search is similar to a greedy best-first search without backtracking.

Is it complete/optimal?

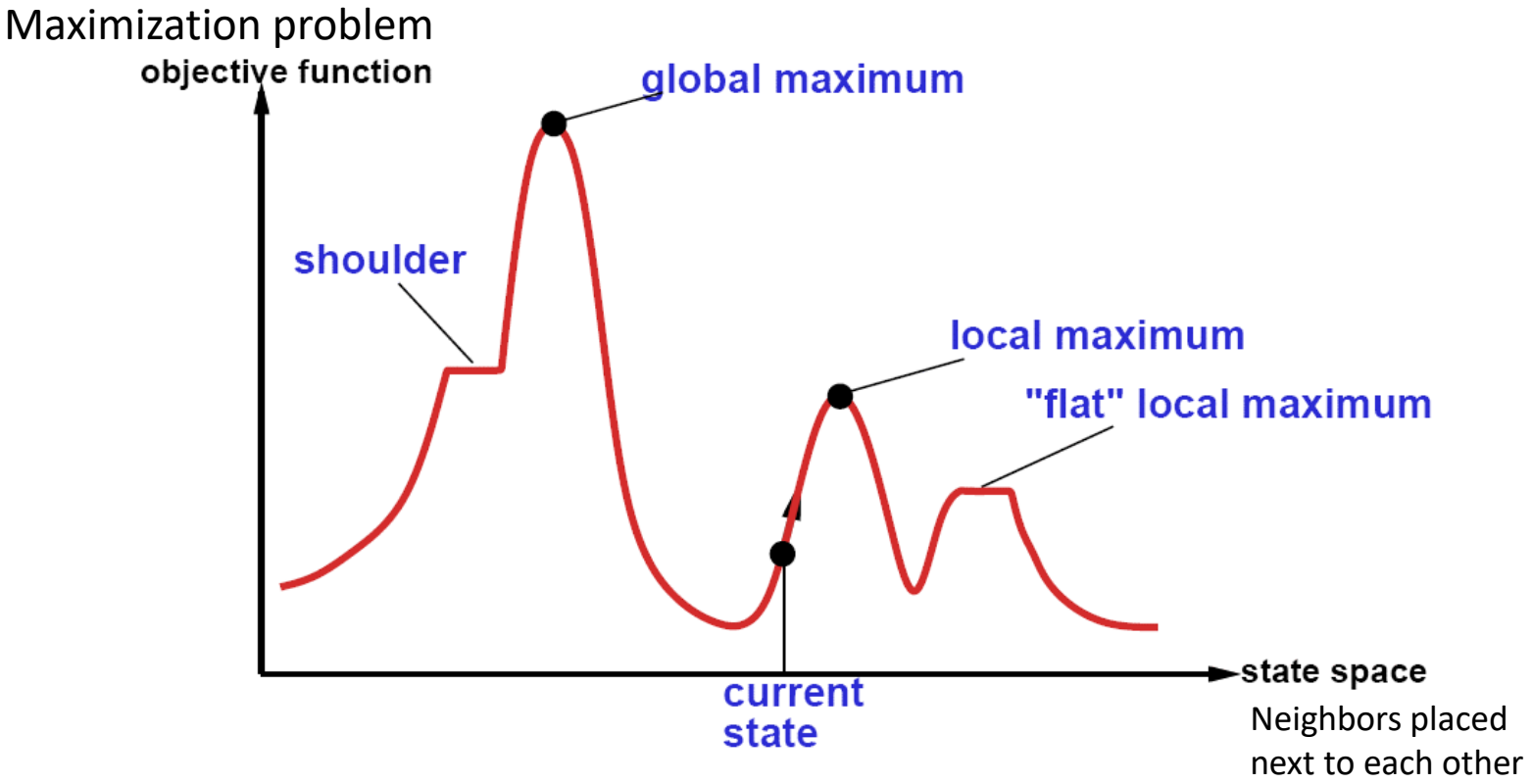
- No – can get stuck in local optima



Example: local optimum for the 8-queens problem. No single queen can be moved to improve the objective function.

$h = 1$

The state space “landscape”



How to escape local maxima?

→ Random restart hill-climbing can help.

What about “shoulders” (“ridges” in higher dimensional space)?

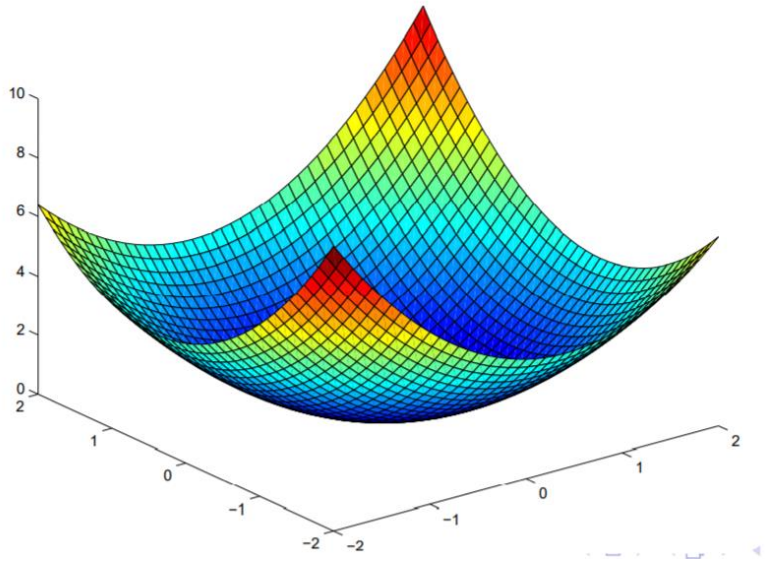
What about “plateaux”?

→ Allow sideways moves.

Non-convex/convex Optimization Problems

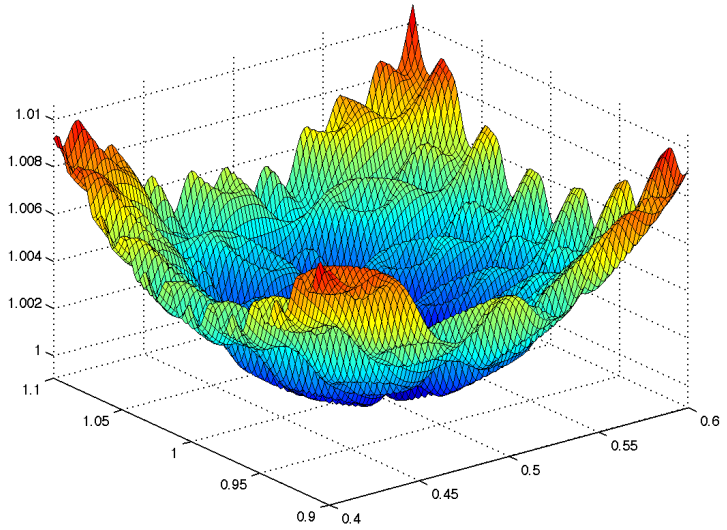
Minimization problem

Convex Problem



One global optimum + smooth function → easy

Non-convex Problem



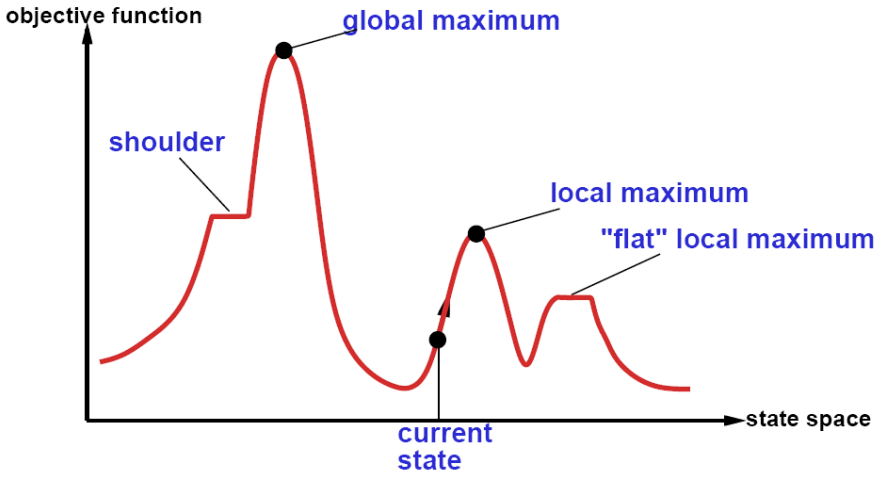
Many local optima → hard

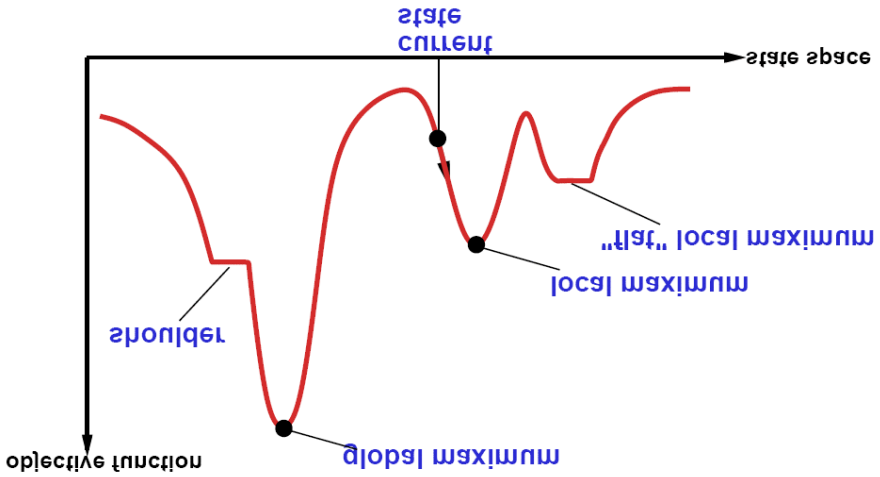
Many discrete optimization problems are like this.

A Note on Minimization vs. Maximization

- The name hill climbing implies **maximizing a function**.
- Optimizers like to state problems as **minimization problems** (e.g., gradient descent).
- Both types of problems are equivalent:

$$\max\big(f(x)\big) \quad \Leftrightarrow \quad \min\big(-f(x)\big)$$



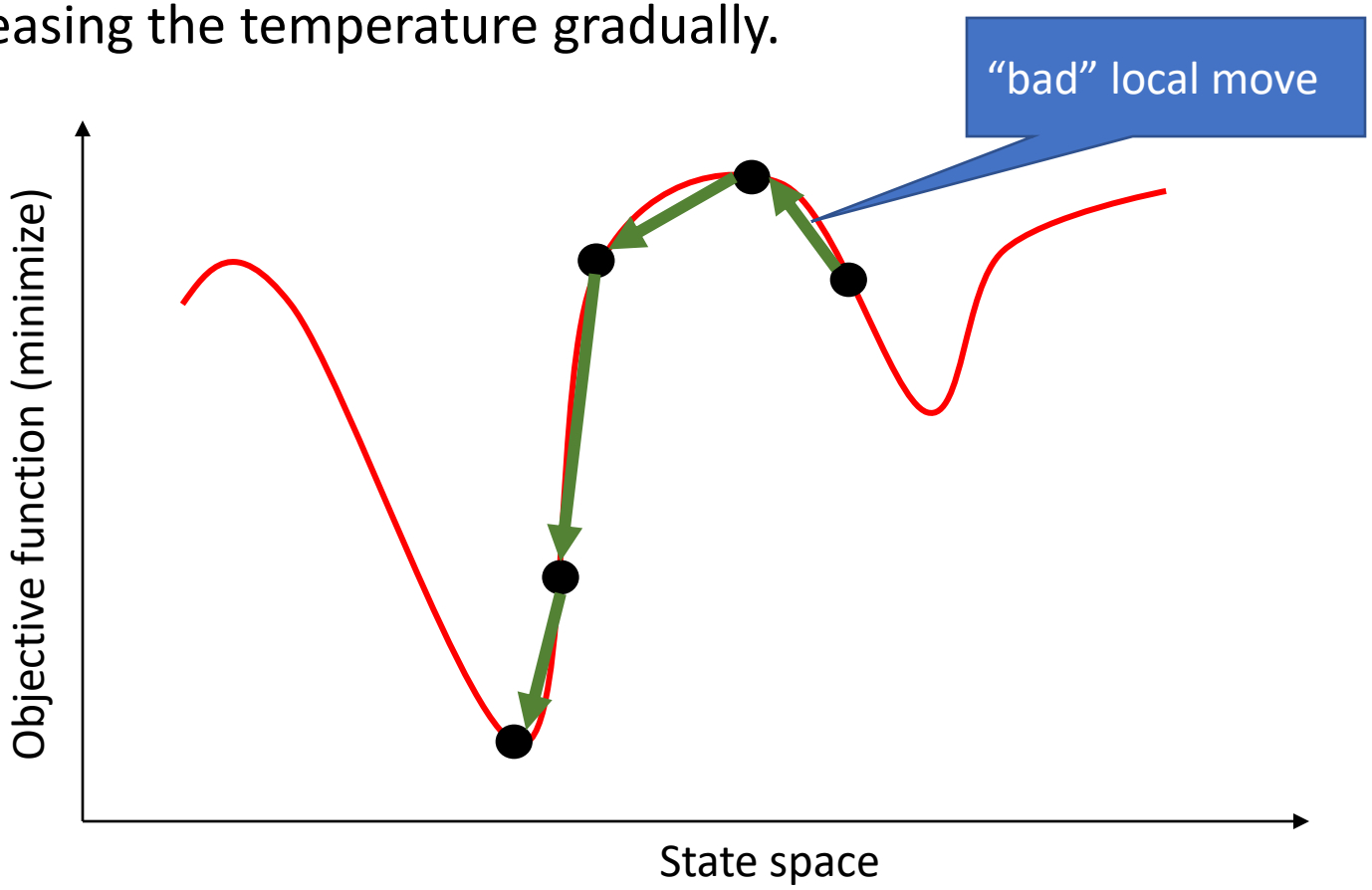


Simulated Annealing

Use heat to escape local optima…

Simulated annealing

- **Idea**: First-choice stochastic hill climbing + escape local minima by allowing some "bad" moves but gradually decrease their frequency.
- Inspired by the process of tempering or hardening metals by decreasing the temperature gradually.



Simulated annealing

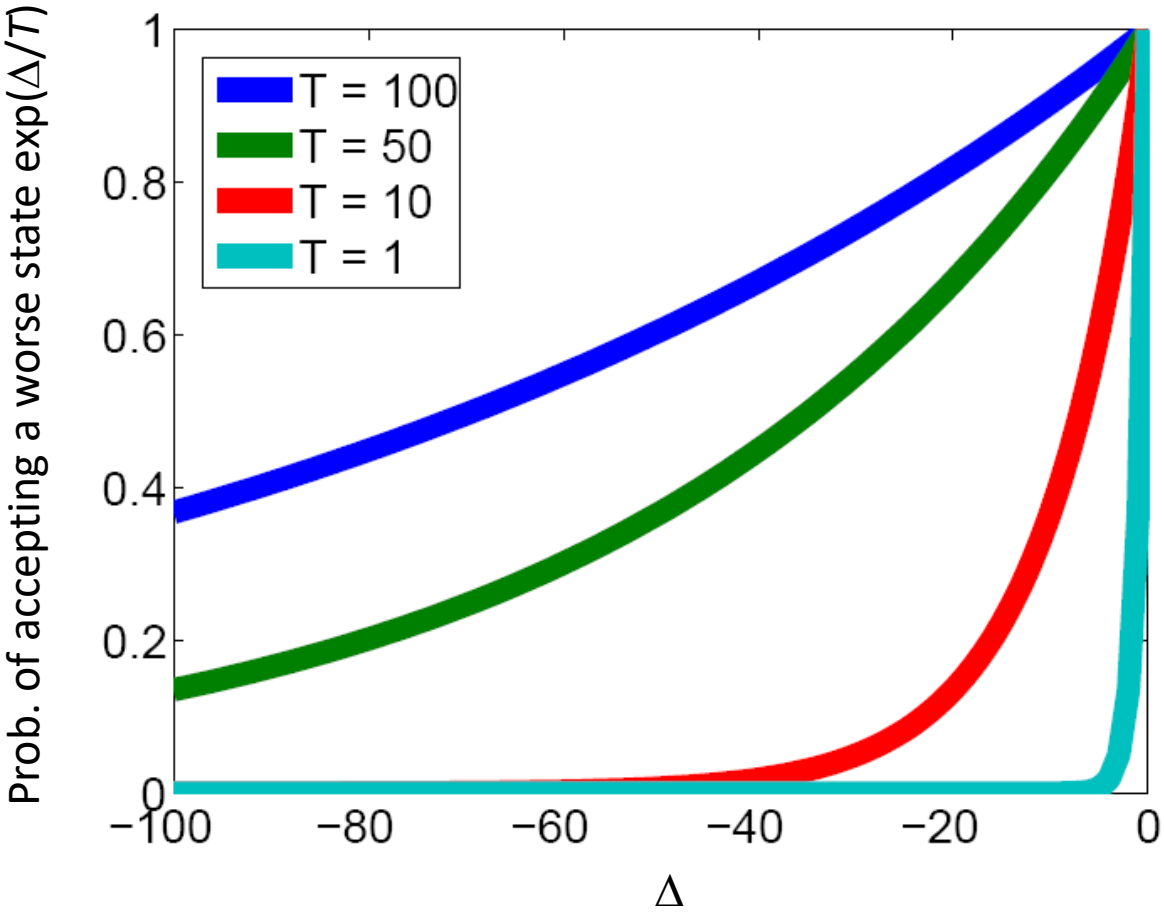
- **Idea**: First-choice stochastic hill climbing + escape local minima by allowing some "bad" moves but gradually decrease their frequency.
- Inspired by the process of tempering or hardening metals by decreasing the temperature gradually.
- The probability of accepting "bad" moves (Metropolis acceptance criterion) follows **an annealing schedule** that reduces the temperature T over time t.

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

Always do good moves

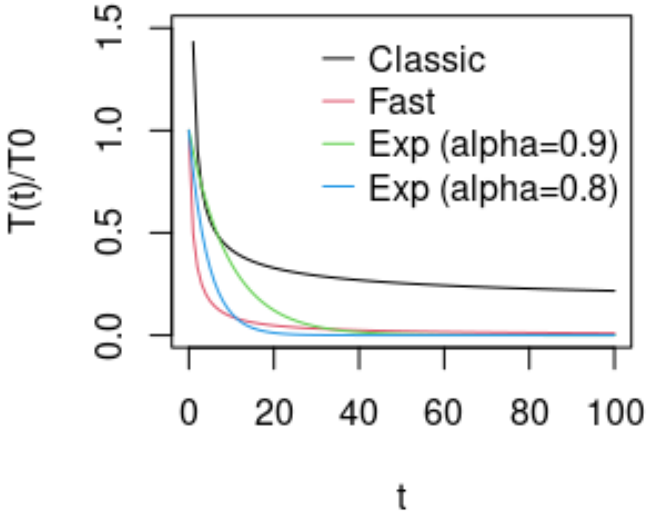
"bad" moves

Effect of temperature



The lower the temperature, the less likely the algorithm will accept a worse state.

Cooling Schedule



The cooling schedule is very important.
Popular schedules for the temperature at time t:

- **Classic simulated annealing:** $T_t = T_0 \frac{1}{\log(1+t)}$
- **Fast simulated annealing** (Szy and Hartley; 1987)

$$T_t = T_0 \frac{1}{1+t}$$

- **Exponential cooling** (Kirkpatrick, Gelatt and Vecchi; 1983)

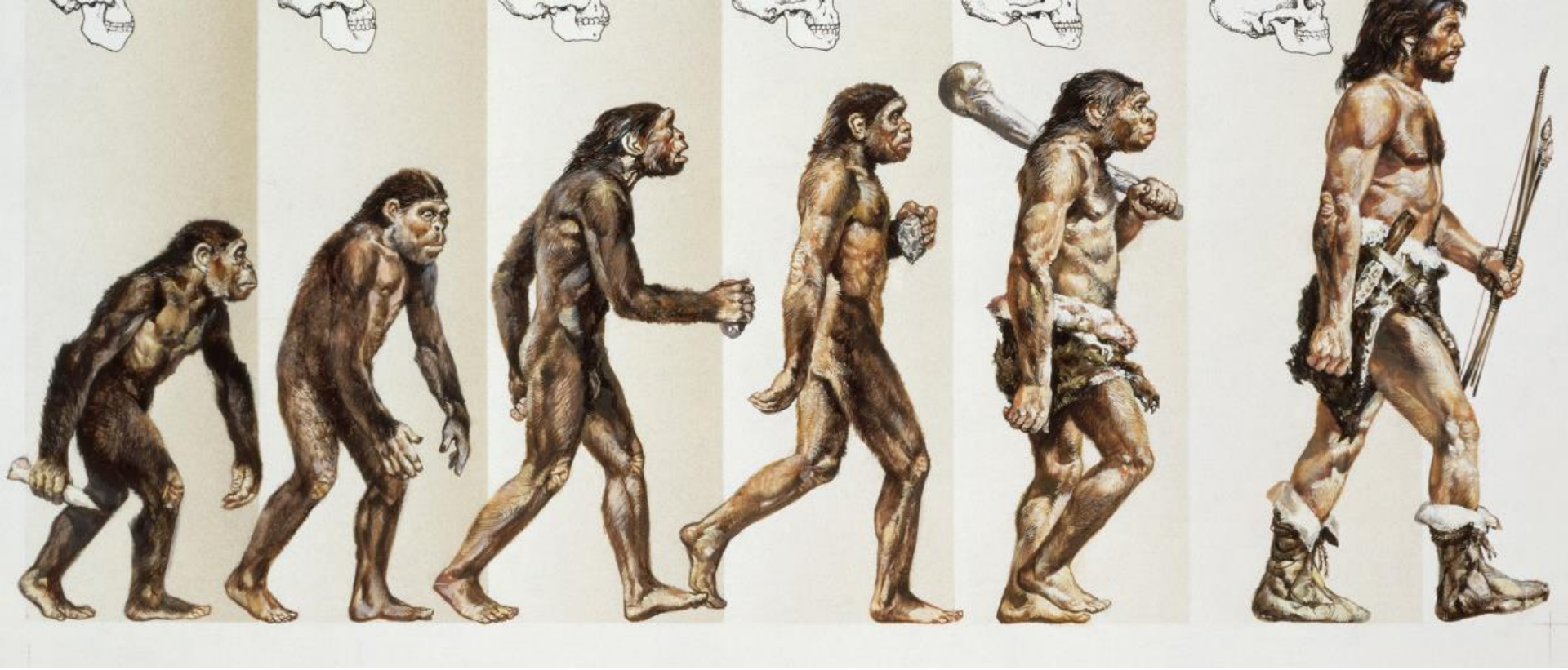
$$T_t = T_0 \alpha^t \quad \text{for} \quad 0.8 < \alpha < 1$$

Notes:

- The best schedule is typically determined by trial-and-error.
- Choose T_0 to provide a high probability that any move will be accepted at time $t = 0$.
- T_t will not be come 0 but very small. Stop when $T < \epsilon$ (ϵ is a very small constant).

Simulated annealing search

- **Guarantee:** If temperature decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching one.
- However:
  - This usually takes impractically long.
  - The more downhill steps you need to escape a local optimum, the less likely you are to make all of them in a row.
- **Markov Chain Monte Carlo (MCMC)** is a general family of randomized algorithms for exploring complicated state spaces.

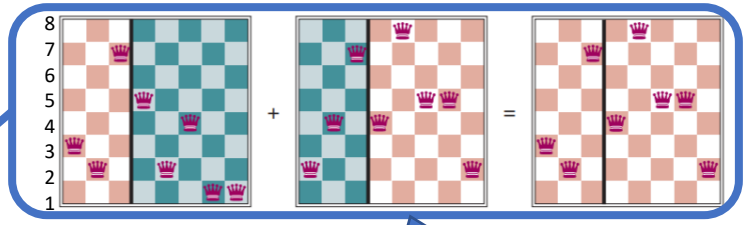


Evolutionary Algorithms

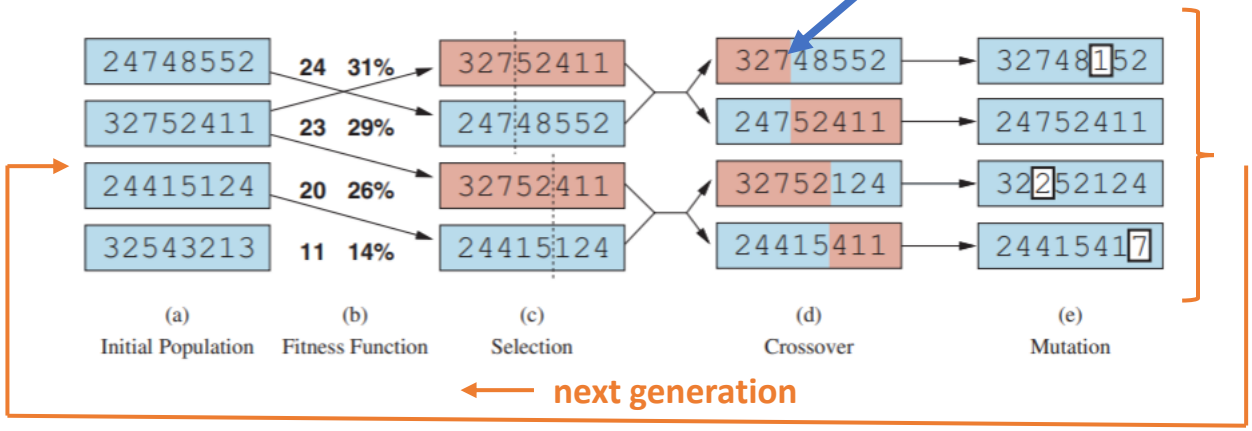
A Population-based Metaheuristics

Evolutionary algorithms / Genetic Algorithms

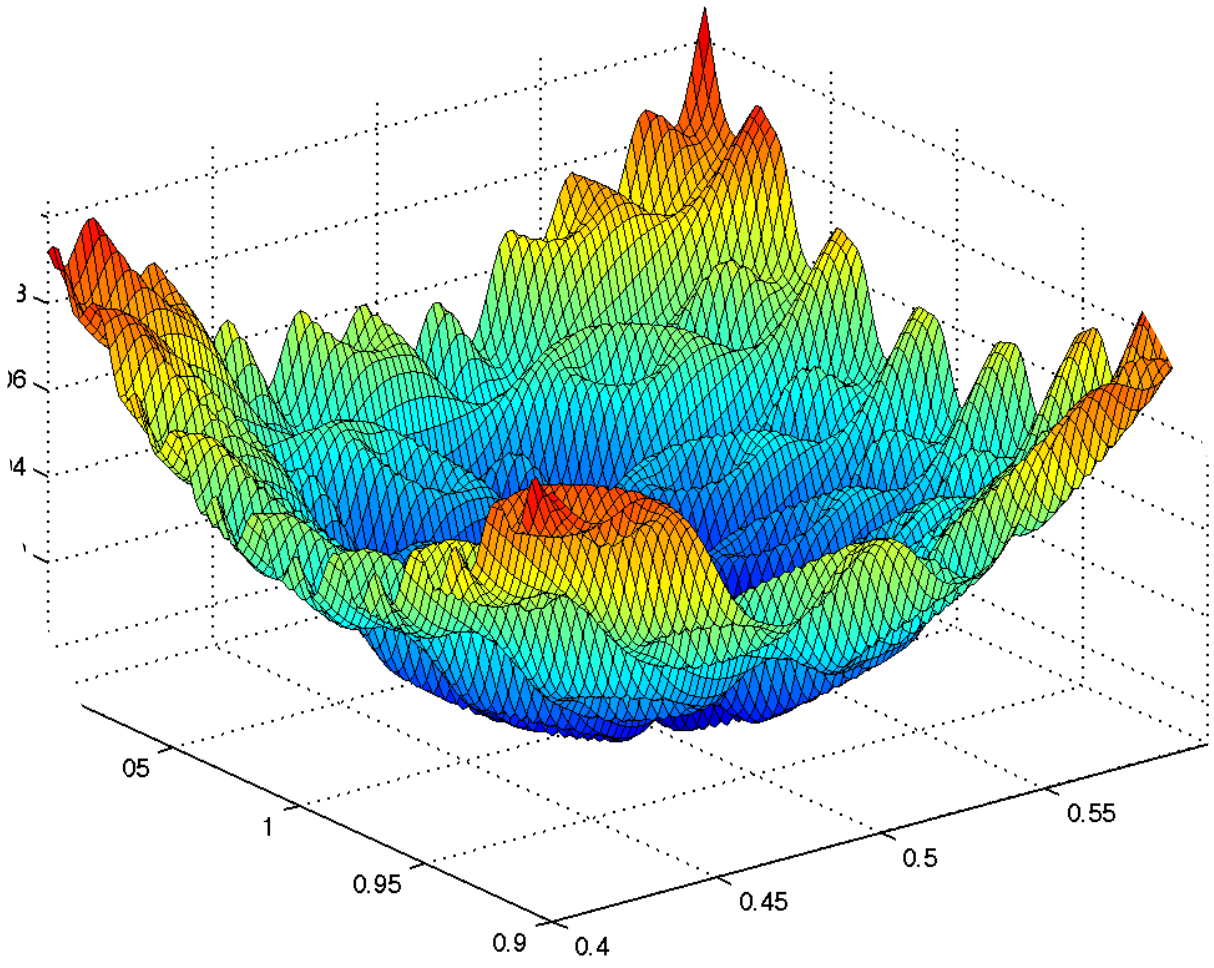
- A metaheuristic for **population-**based optimization.
- Uses mechanisms inspired by biological evolution (genetics):
  - Reproduction: Random selection with probability based on a **fitness** function.
  - Random recombination (crossover)
  - Random mutation
  - Repeated for many generations
- Example: 8-queens problem



Individual = state
Encoding: row of the queen in each column

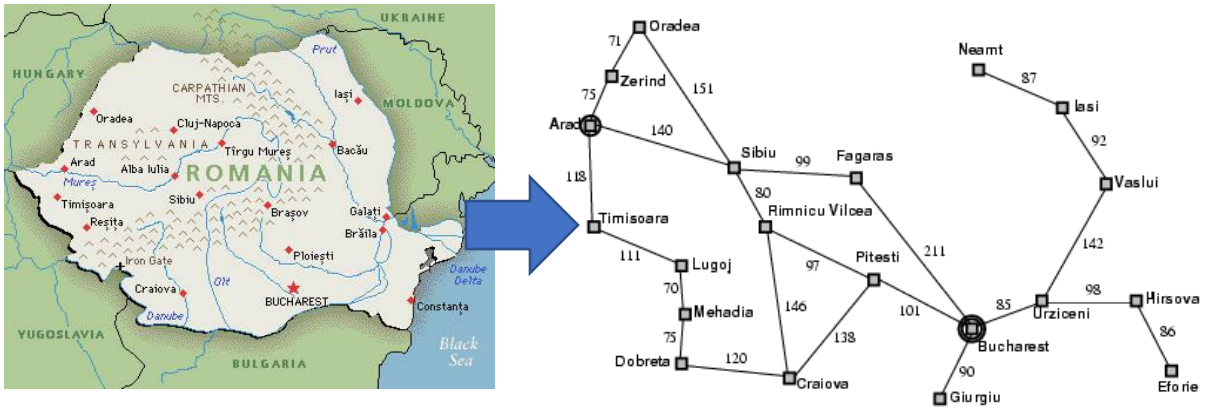


Search in Continuous Spaces



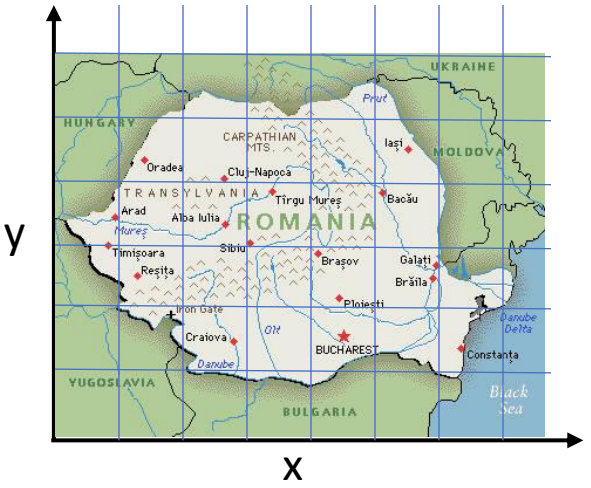
Discretization of the continuous space

- Use atomic states to create a graph



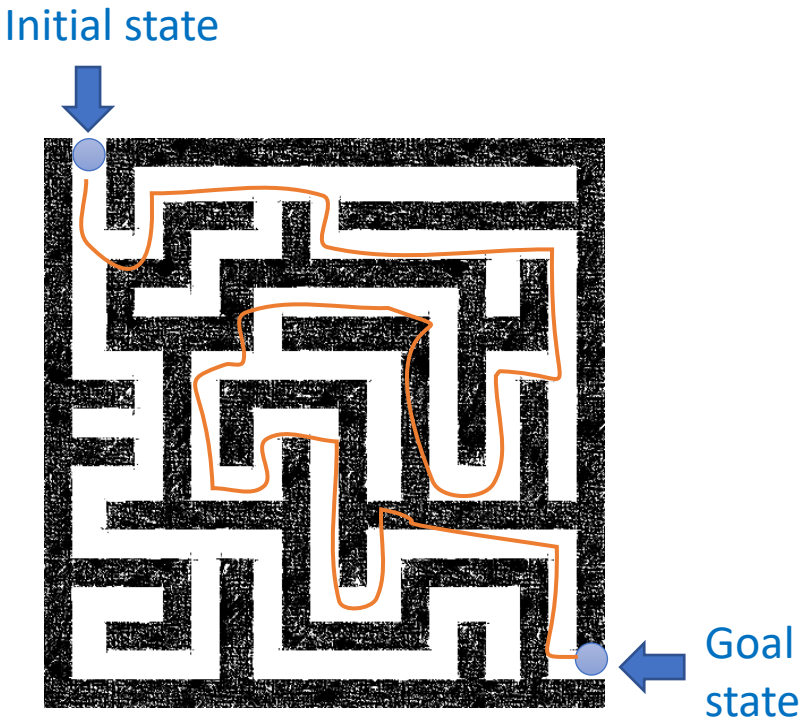
- Use a grid with spacing of size δ

  Note: You probably need a way finer grid!

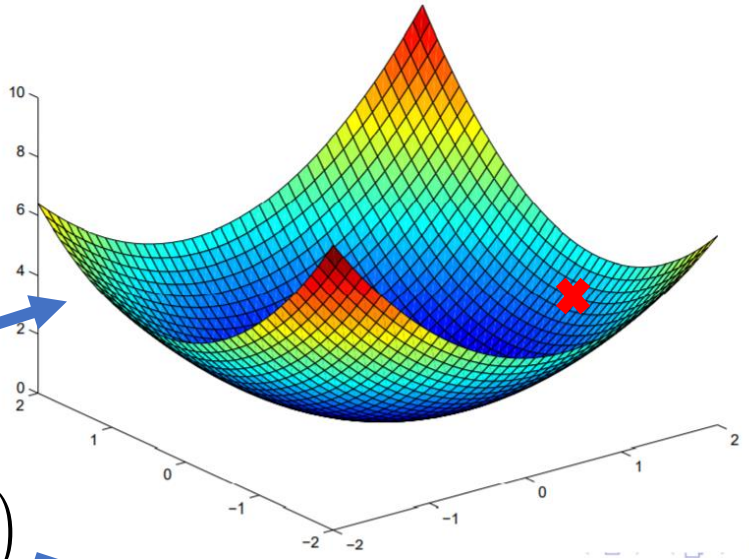


Discretization of the continuous space

How did we discretize this space?



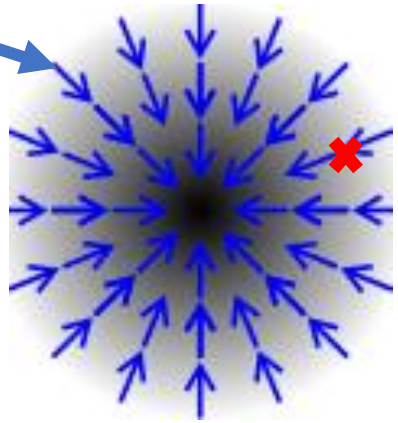
Search in continuous spaces: Gradient



Maximize $f(\boldsymbol{x}) = f(x_1, x_2, \ldots, x_k)$

Gradient at point $\boldsymbol{x}$: $\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_1}\right)$

Find maximum by solving: $\nabla f(\boldsymbol{x}) = 0$



- **Gradient descent (= Steepest-ascend hill climbing for minimization)** with step size α

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x})$$

- **Newton-Raphson method**
  uses the inverse of the Hessian matrix of the second derivative $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ for the step size α

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{H}_f^{-1}(\boldsymbol{x}) \nabla f(\boldsymbol{x})$$

May get stuck in a local optimum if the search space is non-convex! Use simulated annealing.

Search in continuous spaces: Empirical Gradient Methods

- What if the mathematical formulation of the objective function is not known?
- We may have objective values at fixed points, called the training data.
- In this case we can use **empirical gradient search.** This is related to steepest ascend hill climbing in the discretized state space.

→ We will talk more about search in continuous spaces with loss functions using gradient descend when we talk about **parameter learning for machine learning.**