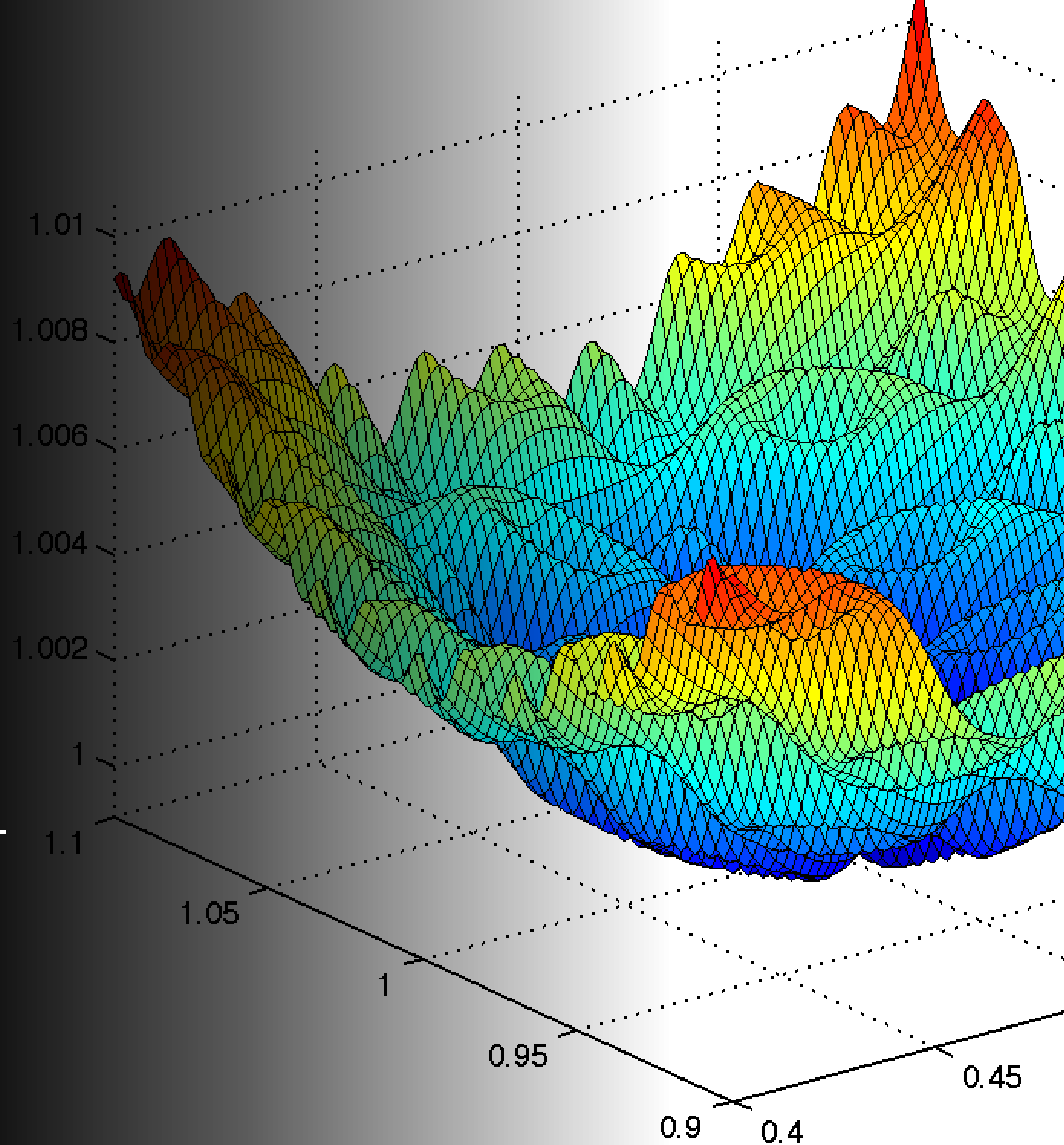
CS 5/7320 Artificial Intelligence

Local Search

AIMA Chapters 4.1 & 4.2

Slides by Michael Hahsler
based on slides by Svetlana Lazepnik



Recap: Uninformed Search/informed search

Tries to find the

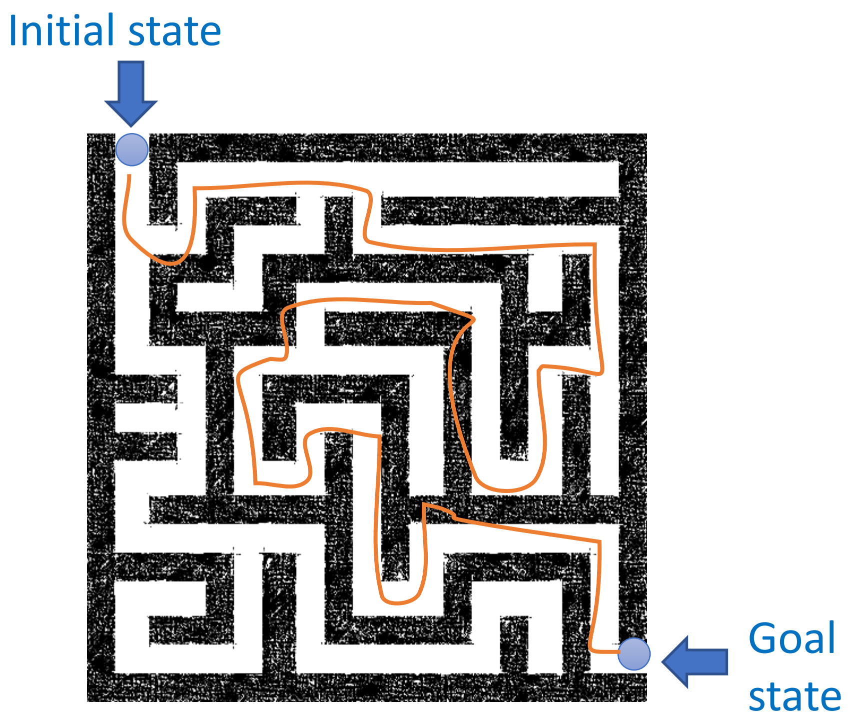
best path

from a

given initial state

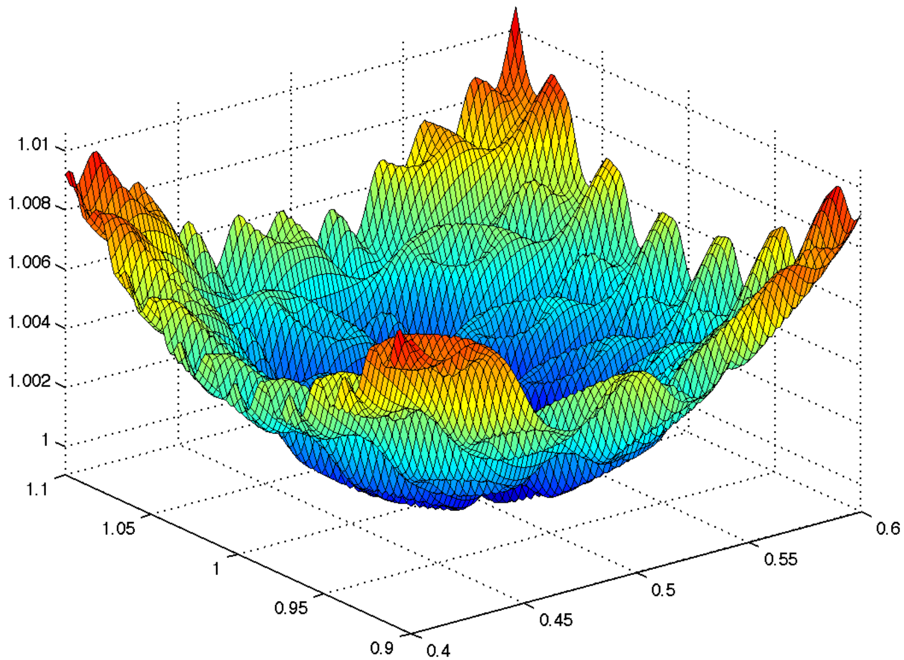
to a

given goal state.



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

Local search algorithms

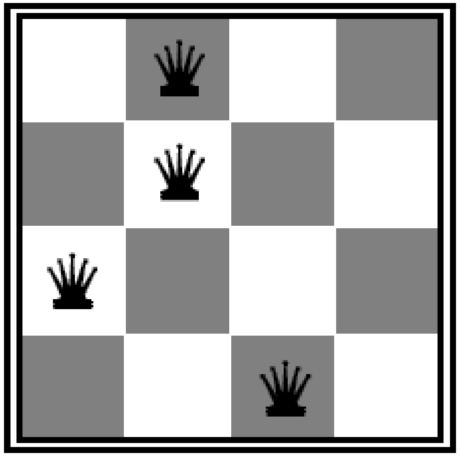


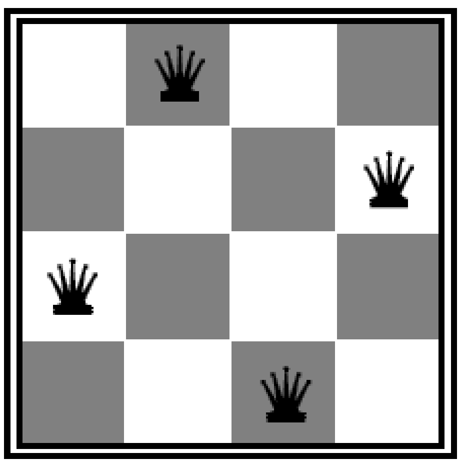
- **Goal**: Search only a small portion of the search space to identify a **good state.**
  a) Often no initial state
  b) Path to solution and path cost are often not important
- We need an **objective function** over the states hat defines what “good” means → **optimization problem**.
- **Idea**: search neighboring states is fast and needs little memory.

Example applications in AI:

- Use for effective search in continuous space (with an infinite state space).
- Identify a good goal state (objective function might be utility).
- Each state might encode a complete plan (a solution).

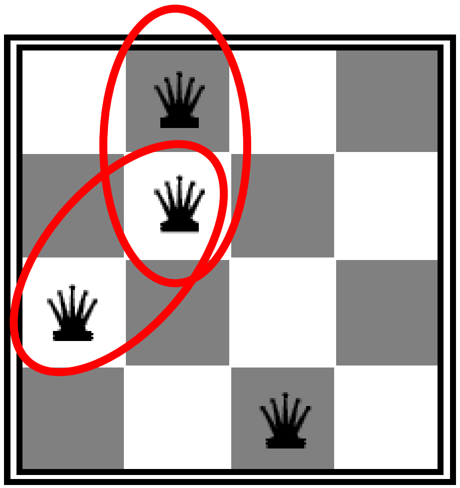
Example: *n*-queens problem

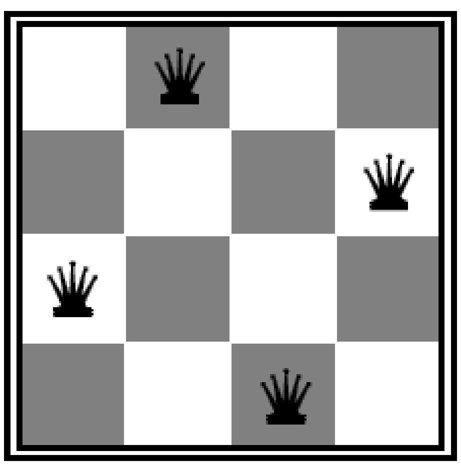




- **Goal**: Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- **State space:** all possible n-queen configurations. **How many are there?**
- What is the **objective function**?

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

  $$\mathbf{16 \times 15 \times 14 \times 13 = 43,680}$$

- What is the **objective function**?

  Number of pairwise conflicts

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:** length of 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
a.k.a Greedy Local Search

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

MotorCycleUSA.com

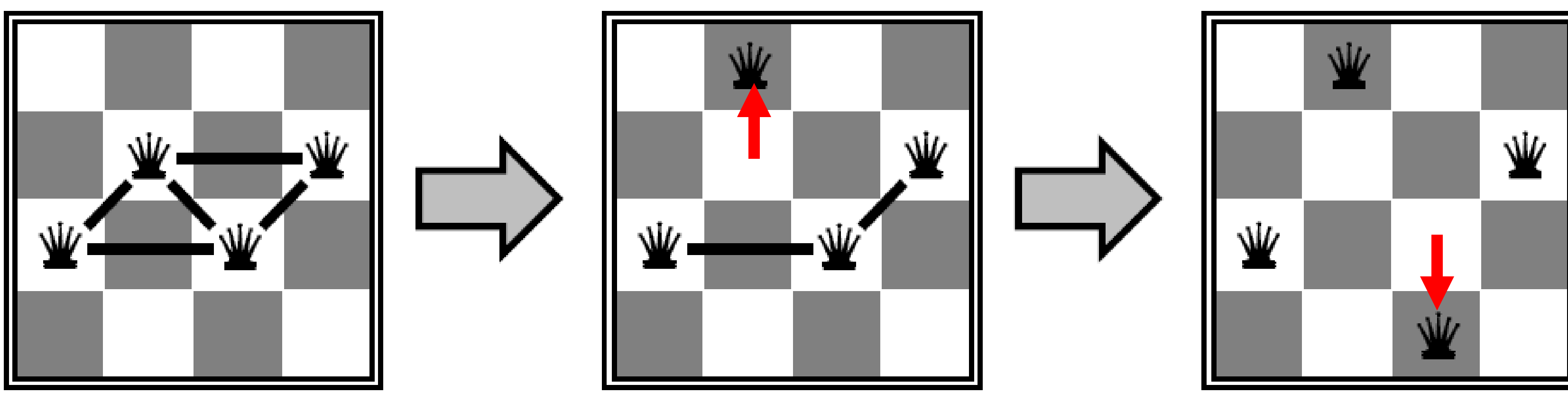
Example: *n*-queens problem

- **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. We can restrict the state space: Only one queen per column.
- **Objective function:** minimize the number of pairwise conflicts.

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

What is a possible local improvement strategy?

- Move one queen within its column to reduce conflicts



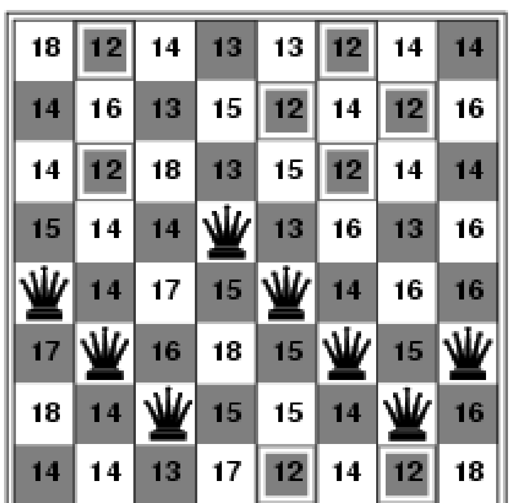
h = 5 h = 2 h = 0

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

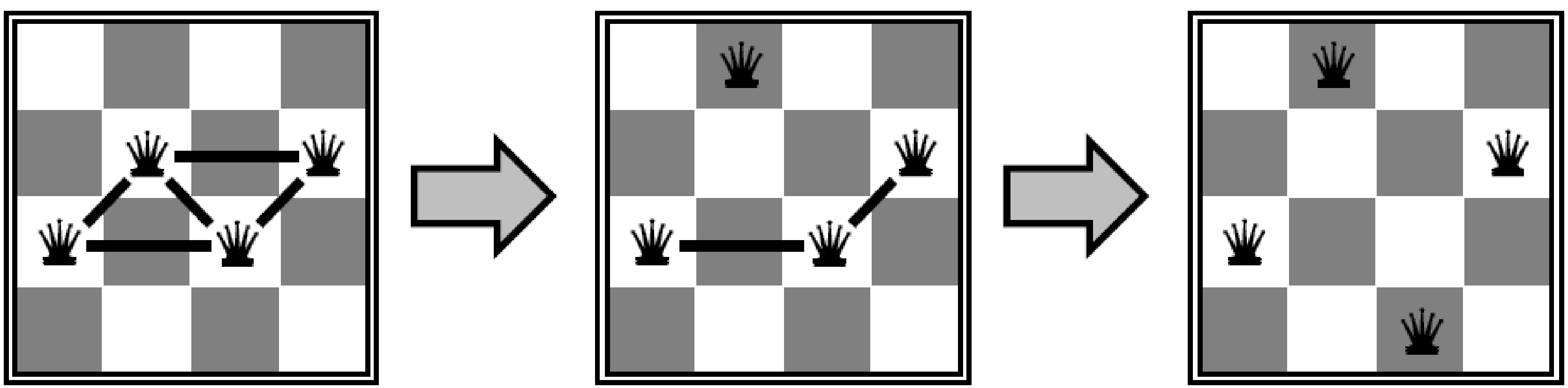
Example: *n*-queens problem

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

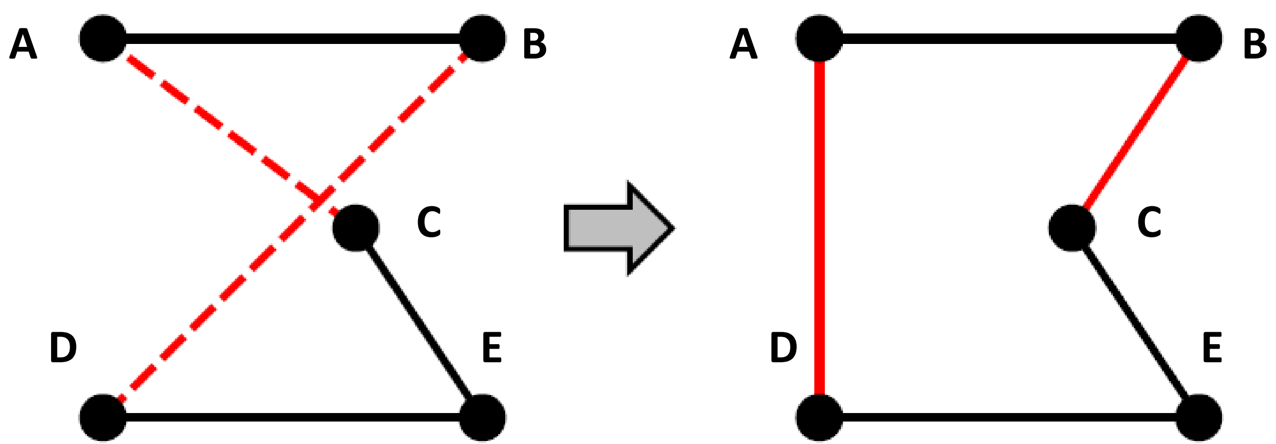
Example: Traveling Salesman Problem

- **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**



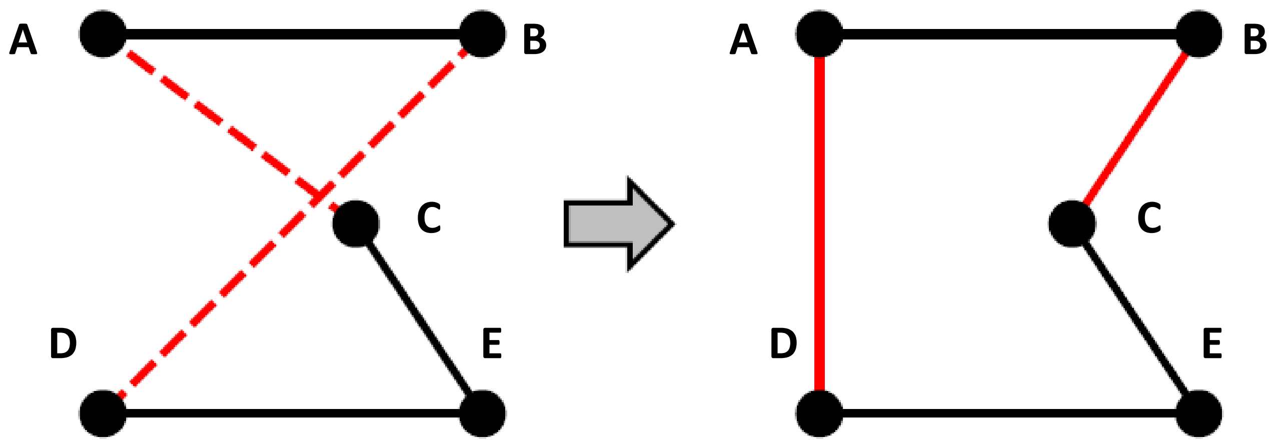
Example: Traveling Salesman Problem

Optimization problem: Find the best tour π

$$\operatorname{argmin}_{\pi}\left(\text{tourLength}(\pi)\right)$$

s.t. π is a valid permutation

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



Hill-climbing search (= Greedy local search)

Many variants

- **Steepest-ascend hill climbing**

```
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
```

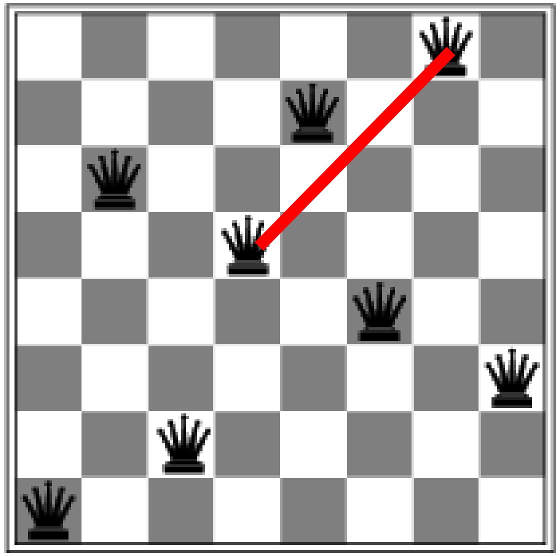
- **Stochastic hill climbing**
  - choose randomly among all uphill moves, or
  - generate randomly new states until a better one is found (first-choice hill climbing)
- **Random-restart hill climbing** – to deal with local optima

Hill-climbing search

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

Is it complete/optimal?

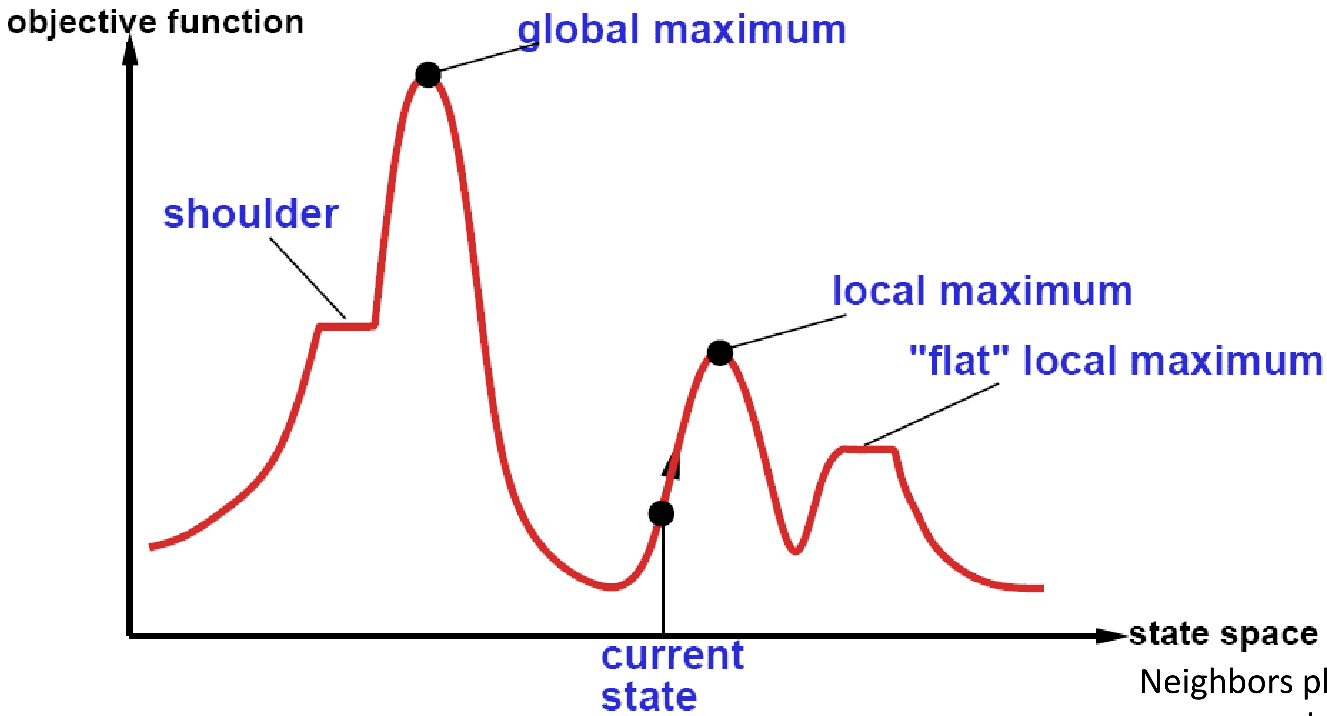
- No – can get stuck in local optima



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”



Neighbors placed next to each other

How to escape local maxima?

→ Random restart hill-climbing can help.

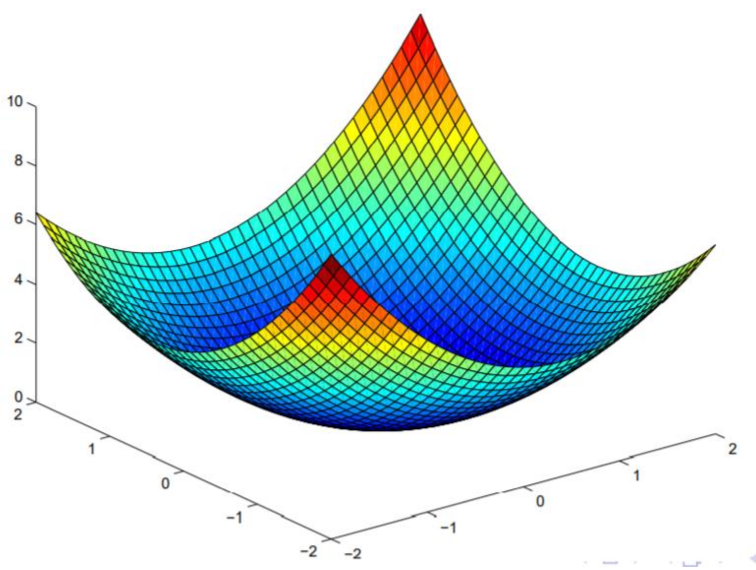
What about “shoulders” (“ridges” in higher dimensional space)?

What about “plateaux”?

→ Allow sideways moves.

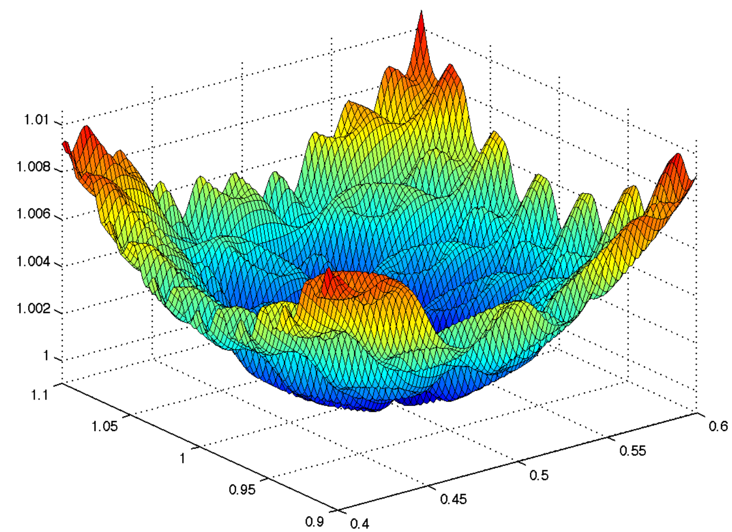
Non-convex/convex Optimization Problems

Convex Problem



One global optimum + smooth function → easy

Non-convex Problem



Many local optima → hard

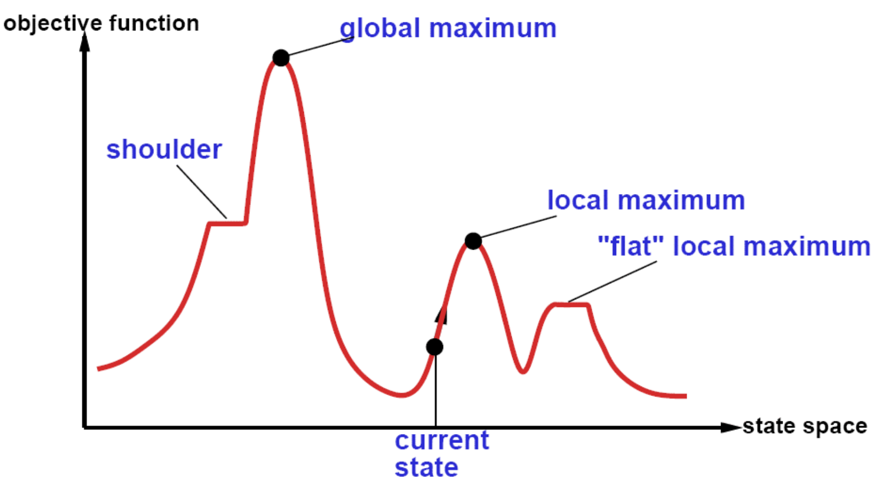
Many discrete optimization problems are like this.

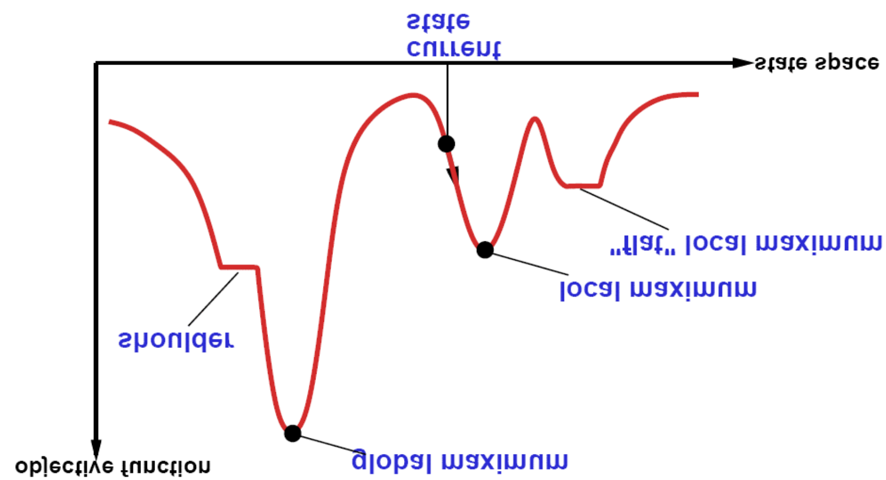
A Note on Minimization vs. Maximization

- Hill climbing implies **maximizing a function**.
- Optimizers like to state problems as **minimization problems**.

- Both types of problems are equivalent:

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



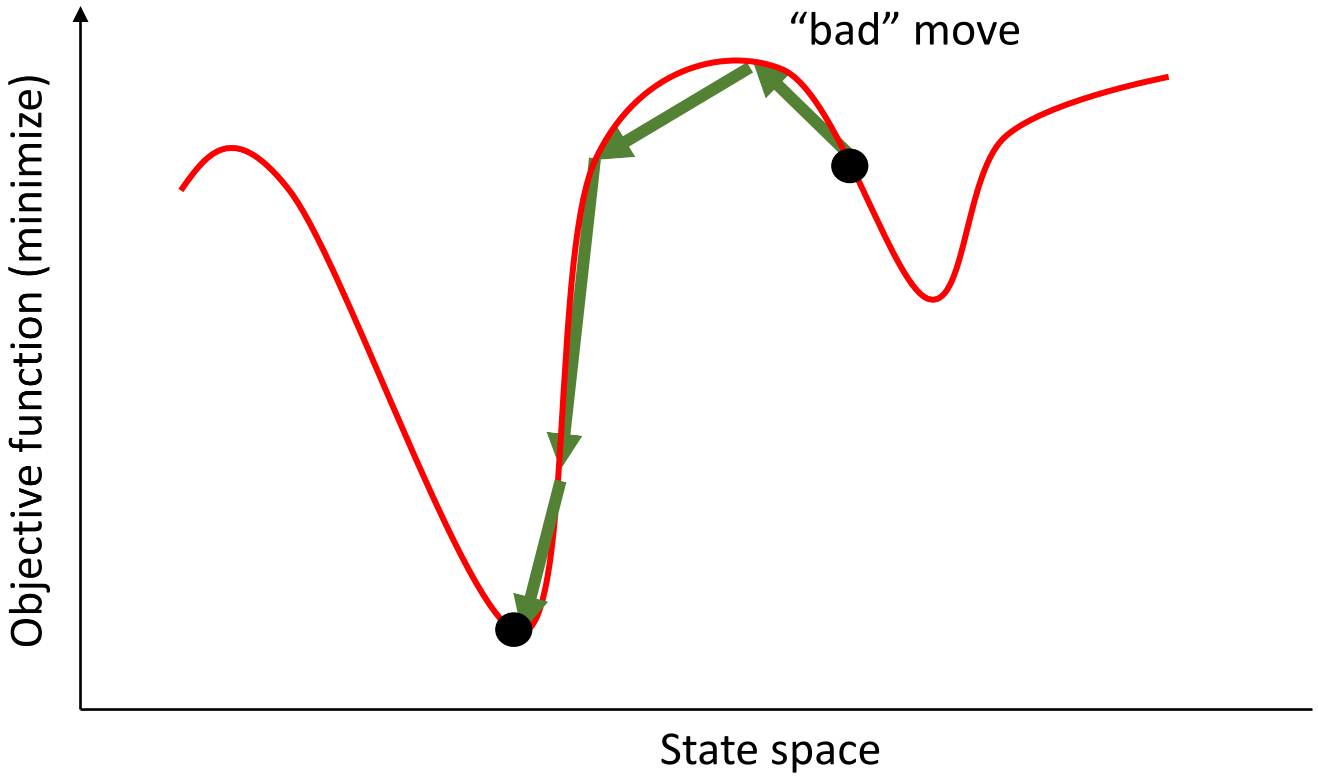


Simulated Annealing

To escape local maxima

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.

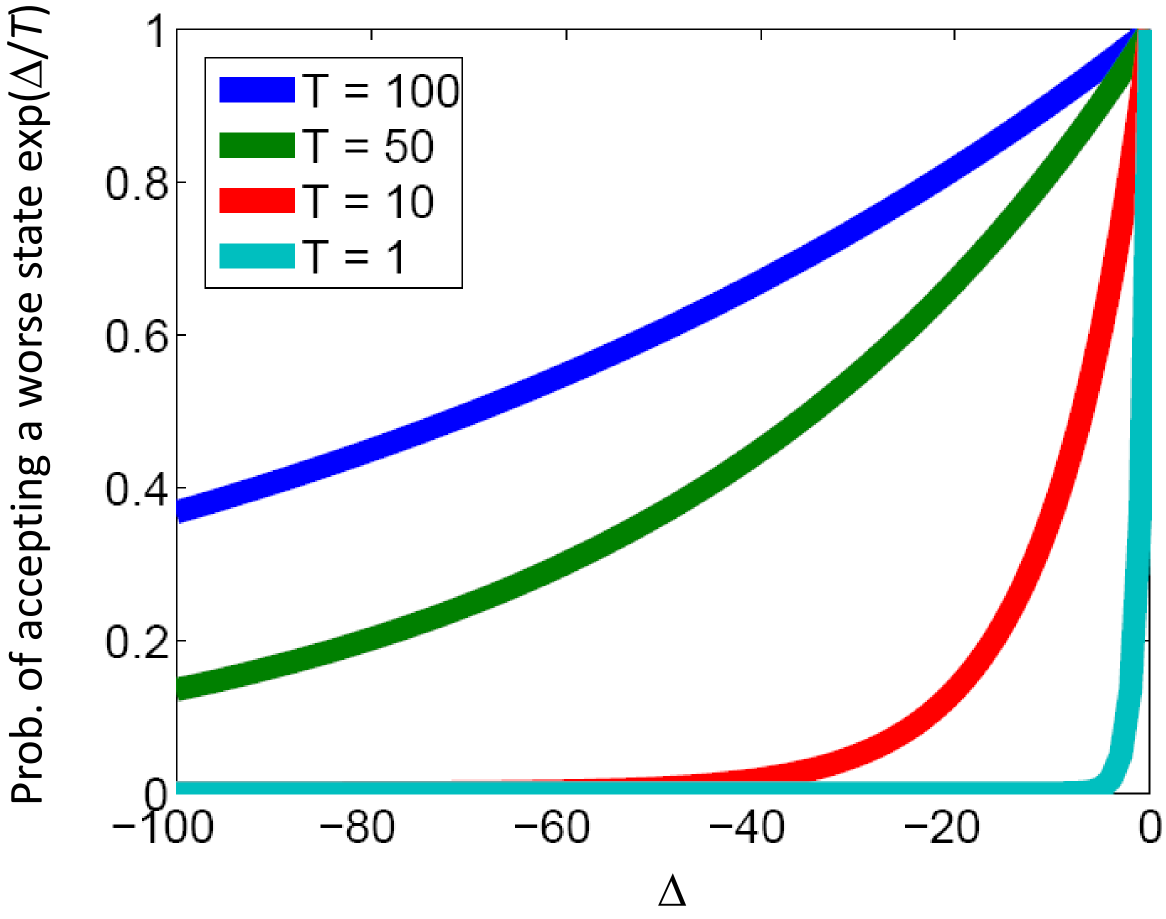


Simulated annealing

- **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 follows **an annealing schedule** that reduces the temperature T over time t.

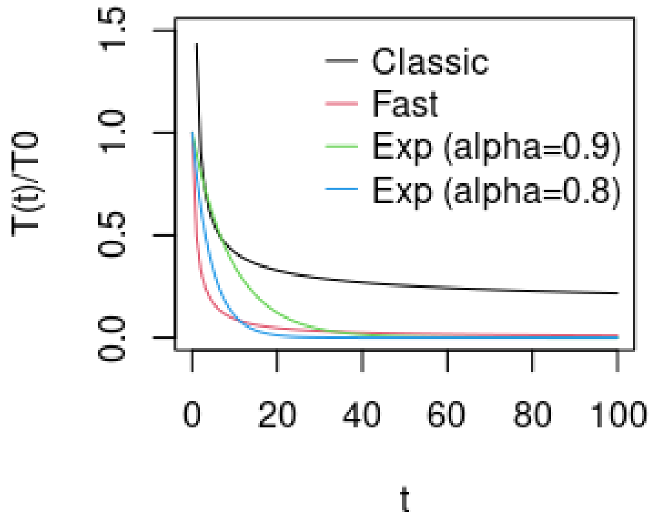
function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
    current $\leftarrow$ *problem*.INITIAL
    for $t = 1$ **to** ∞ **do**
        $T \leftarrow schedule(t)$
        if $T = 0$ **then return** *current*
        next $\leftarrow$ a randomly selected successor of *current*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

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)

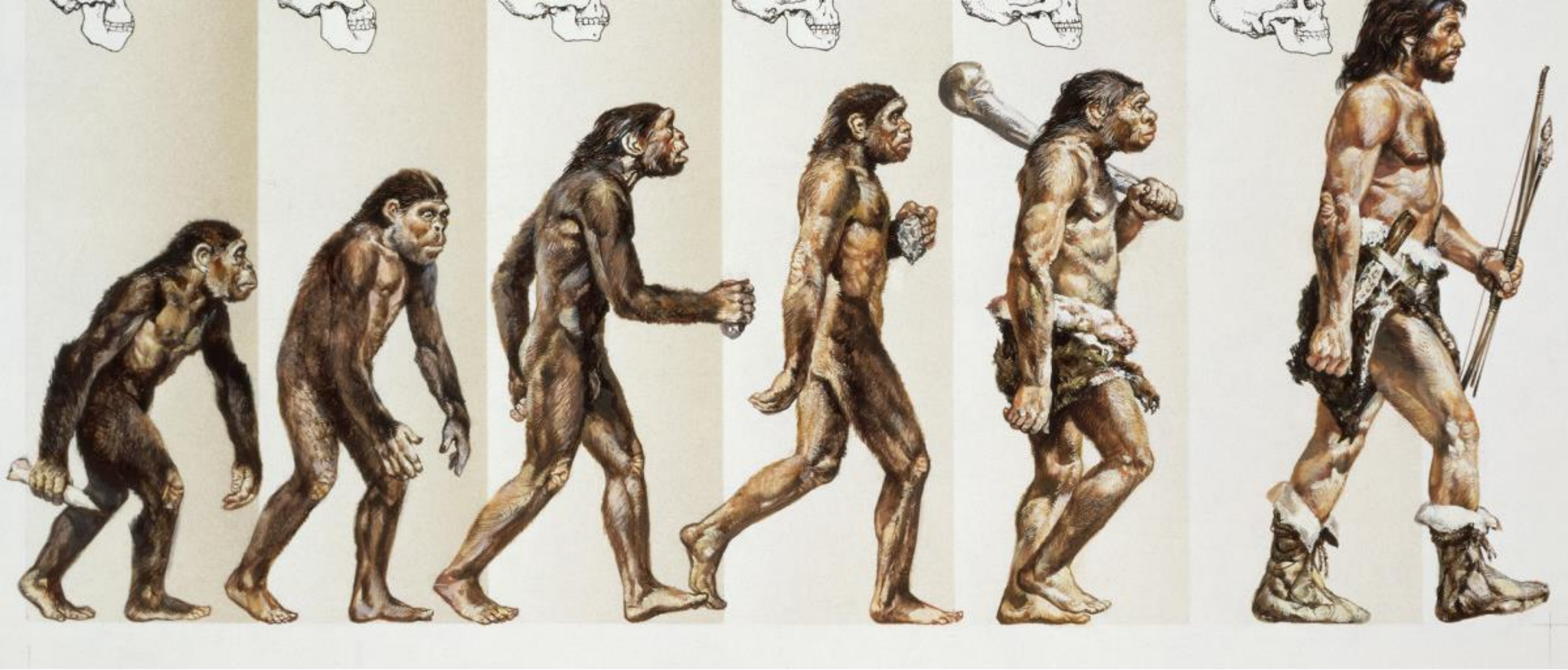
$$T_t = T_0 \alpha^t \quad \text{for} \quad 0.8 < \alpha < 1$$

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.
- More modern techniques: general family of *Markov Chain Monte Carlo* (MCMC) algorithms for exploring complicated state spaces.

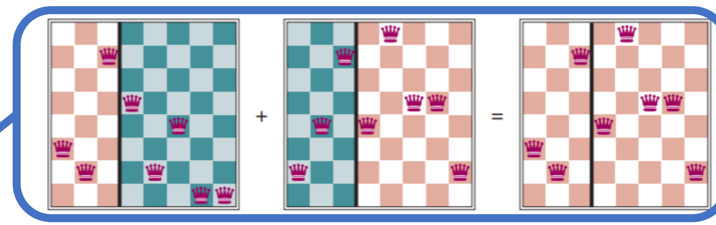


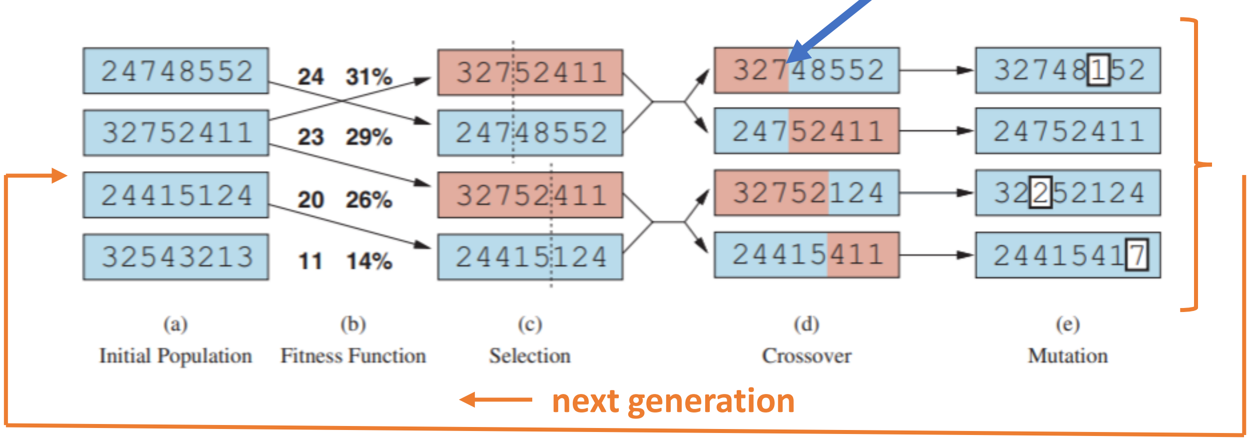
Evolutionary Algorithms

A Population-based Metaheuristics

Evolutionary algorithms / Genetic Algorithms

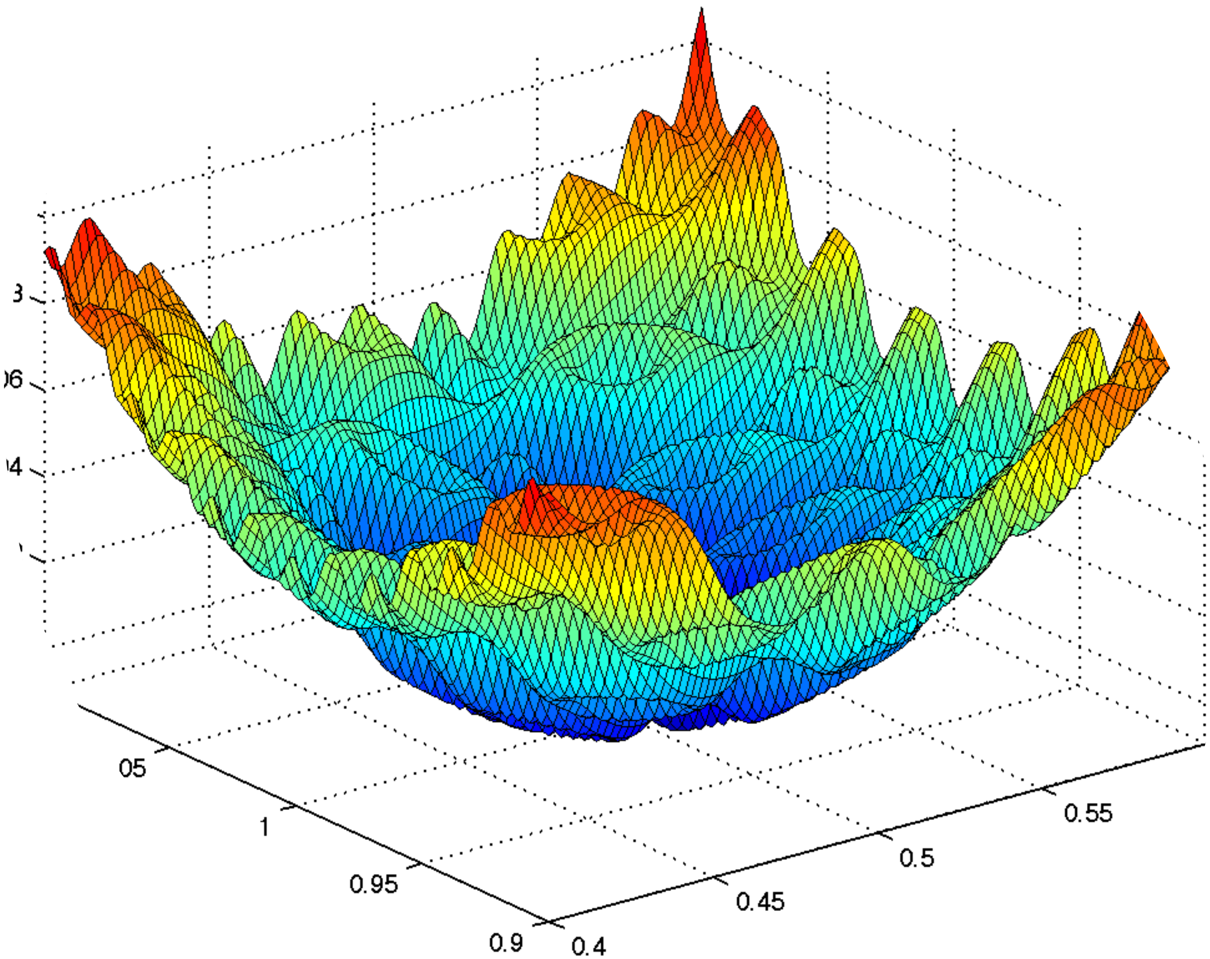
- A **population-**based metaheuristic optimization algorithm.
- Uses mechanisms inspired by biological evolution (genetics):
  - Reproduction: Random selection with probability based on a **fitness** function.
  - Random recombination (crossover)
  - Random mutation
  - Repeated for many generations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





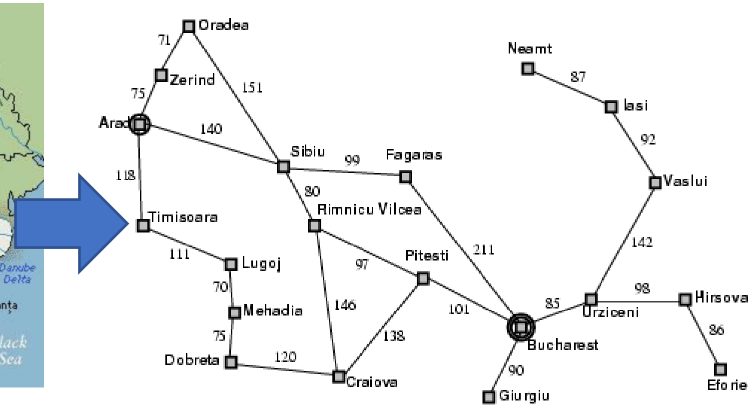
Encoding: position of the queen in each column

Search in Continuous Spaces



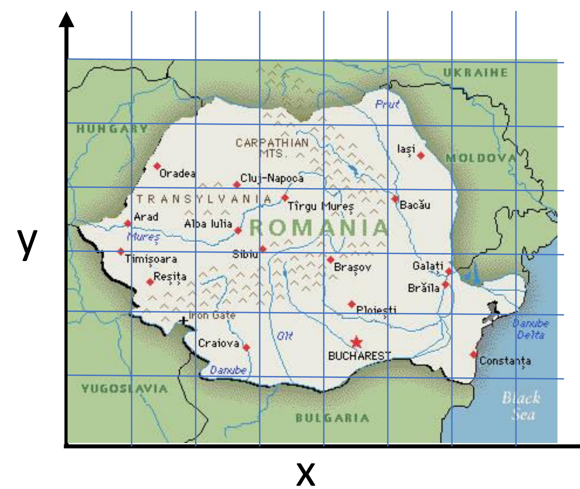
Discretization of the continuous space

- Use atomic states



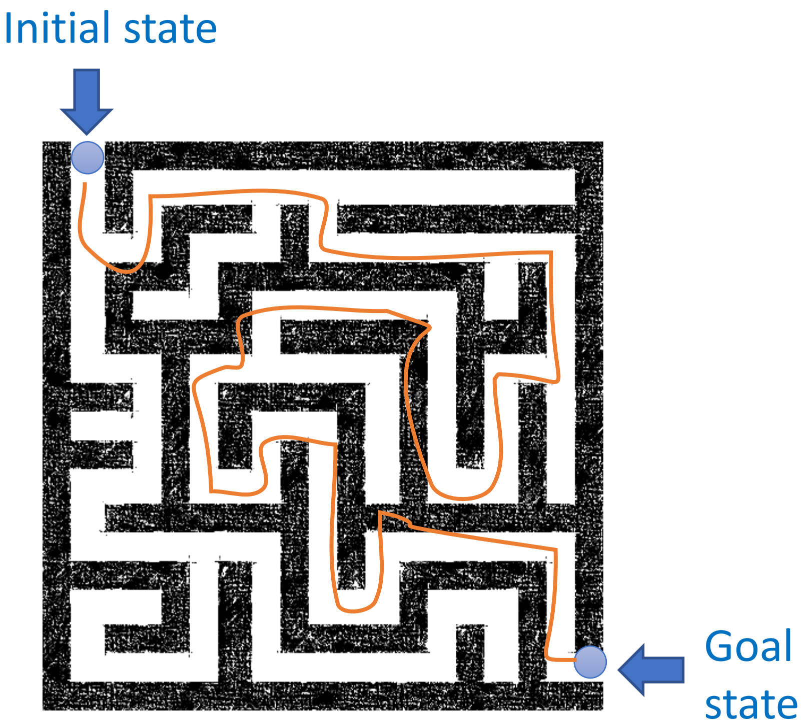
- Use a grid with spacing of size δ

  Note: You probably need a way finer grid!

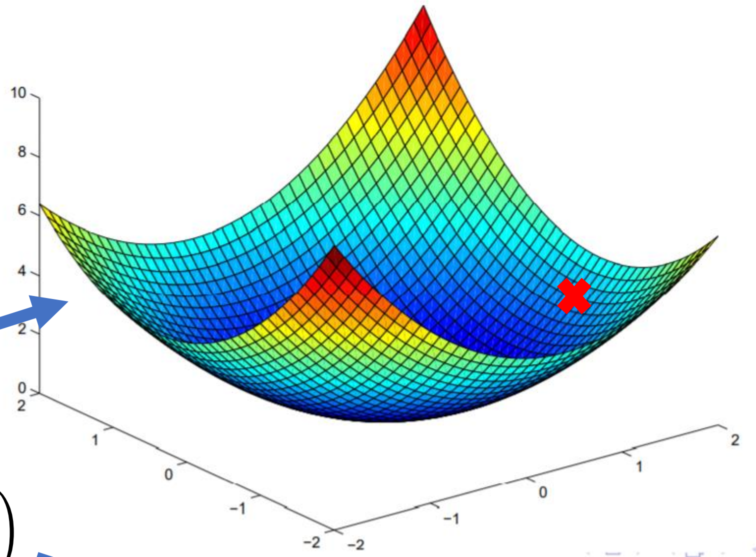


Discretization of the continuous space

How did we discretize this space?



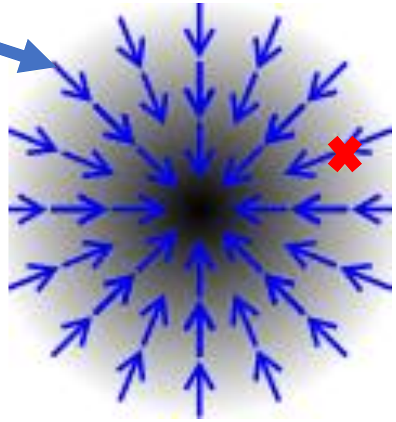
Search in continuous spaces: Gradient



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

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

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



- **Steepest-ascend hill climbing ("gradient descend" 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 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.**