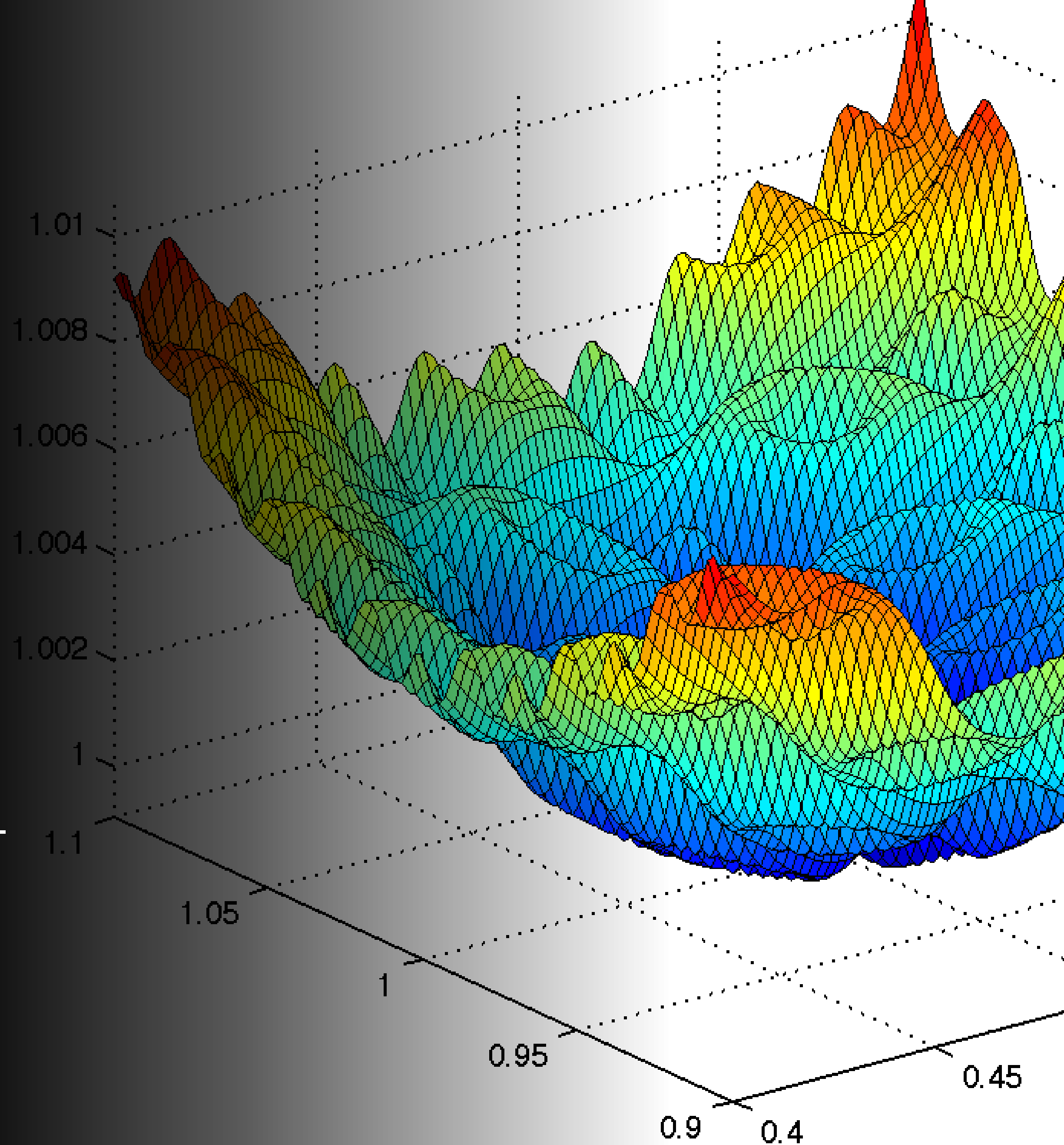
CS 5/7320
Artificial Intelligence

Local Search
AIMA Chapters 4.1 & 4.2

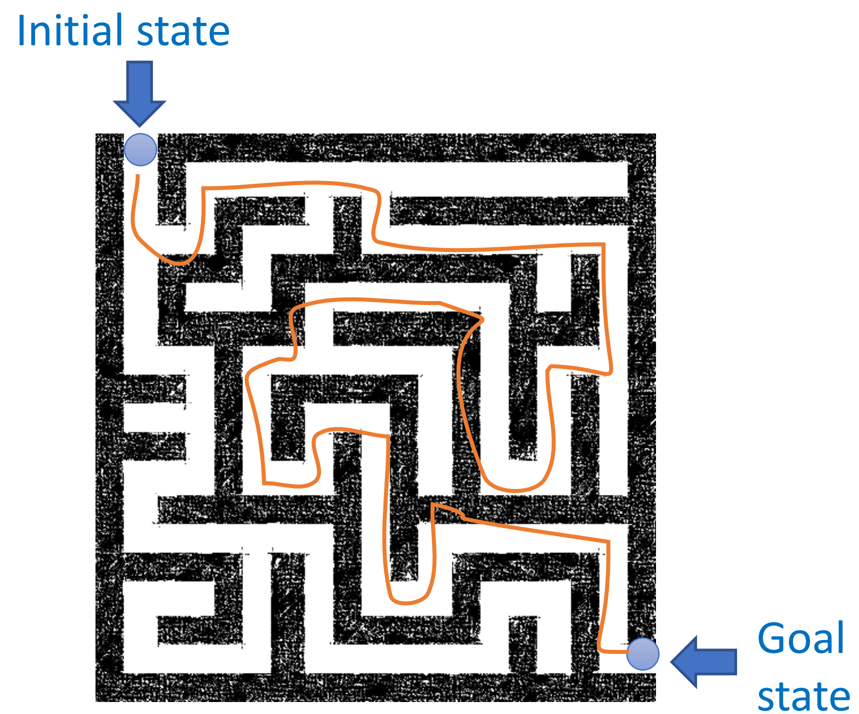


Slides by Michael Hahsler
based on slides by Svetlana Lazepnik
with figures from the AIMA textbook.

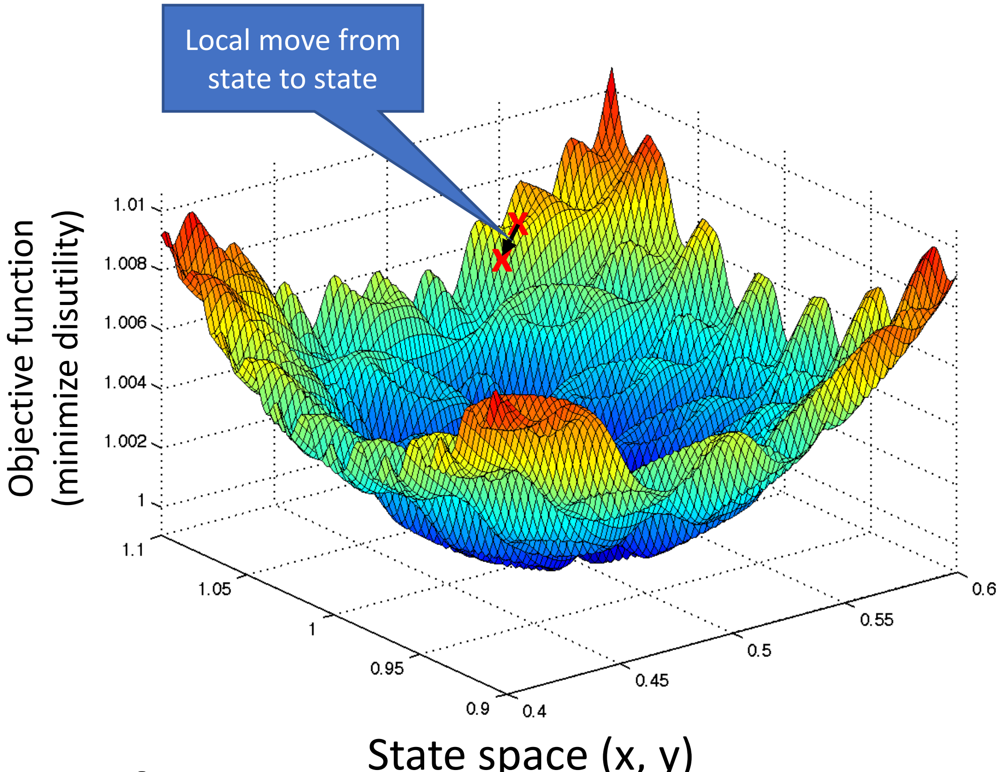
Recap: Uninformed and Informed Search

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



- What if we do not know a goal state, but how "good" different states are? This is typically called the **objective function** → finding the "best state" is an **optimization problem**.
- We need a fast and memory-efficient way to find **the best/a good state.**

Idea:

- Improve the current solution by moving from the current state to a neighboring better state (a.k.a. performing a local move).
- This is fast and needs little memory (no search tree).

Local Search Algorithms

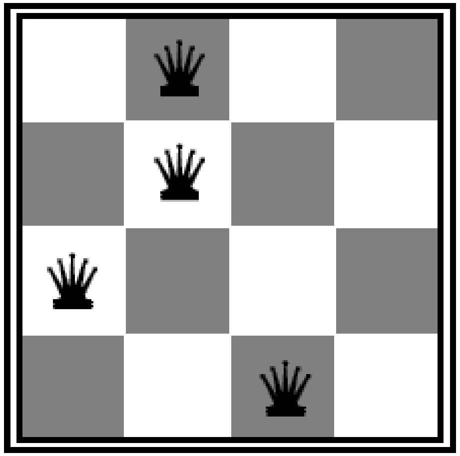
Difference to search from the previous chapter:

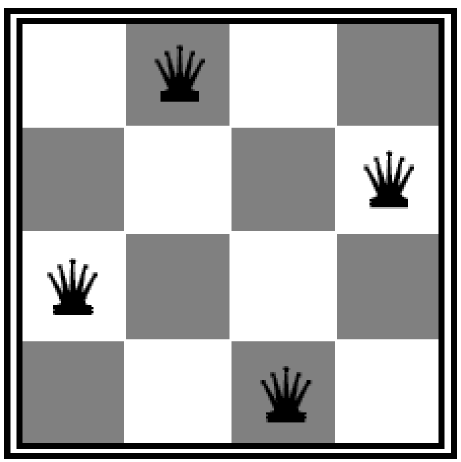
a) **Goal state is unknown,** but we know or can calculate the utility for each state the utility. We want to identify the state with the highest utility.
b) Often no explicit initial state + **path to goal and path cost are not important.**
c) **No search tree**. Just stores the current state and move to a “better” state if possible.

Use in AI

- **Goal-based agent**: Identify a good goal state with a good objective function value (utility) before planning the path to that state.
- **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.
- **General optimization**: Use for effective heuristic search to find good solutions in large or continuous search spaces. E.g., gradient descend to train neural networks.

Example: *n*-Queens Problem





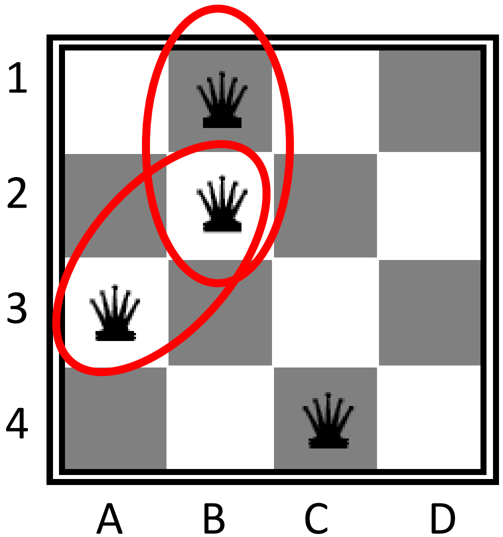
- **Goal**: Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal.

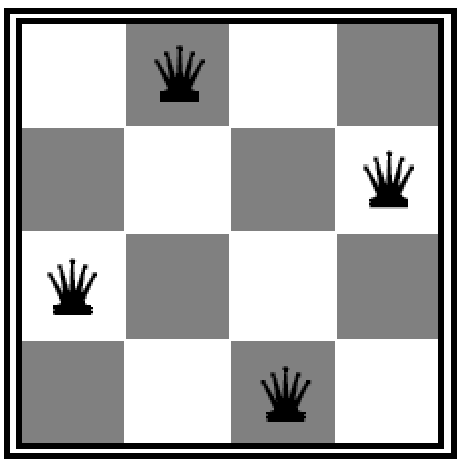
- **State representation:** Structured storing the position of the queens.

- **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 representation:** Structured storing the position of the queens.
  E.g. $(A3, B1, B2, C4)$
- **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 based on the state representation.

Note: this can be seen as a heuristic used in informed search, but it may not be an admissible heuristic.

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.

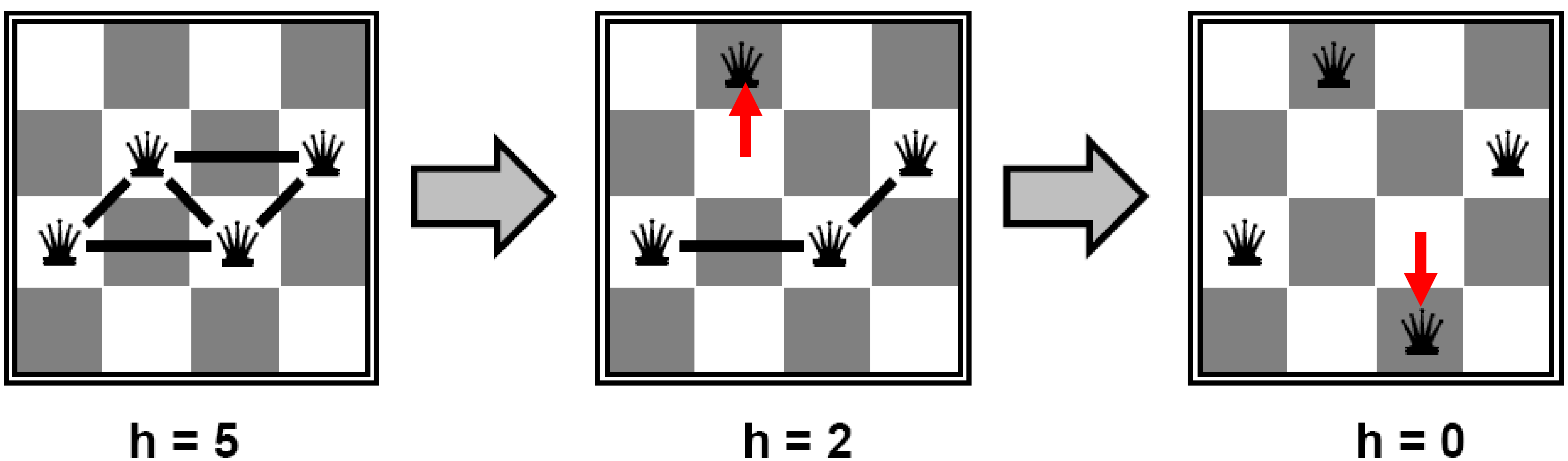
Example: *n*-Queens Problem

- **Goal:** Put n queens on an $n \times n$ 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 from 1820 to $4^4 = 256$

What is a possible local improvement strategy?

- Move one queen within its column to reduce conflicts

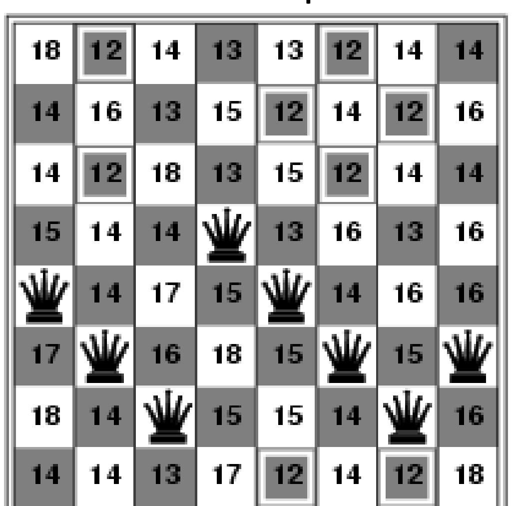


Example: *n*-Queens Problem

- **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!

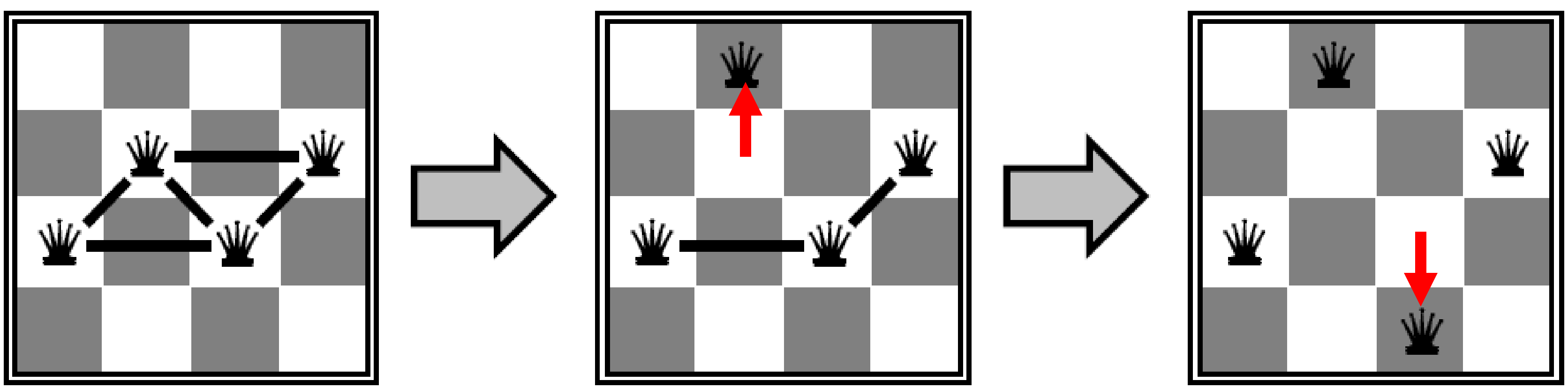
Example: *n*-Queens Problem

Optimization problem: find the best arrangement a

$$a^* = \mathrm{argmin}_a \ \mathrm{conflicts}(a)$$

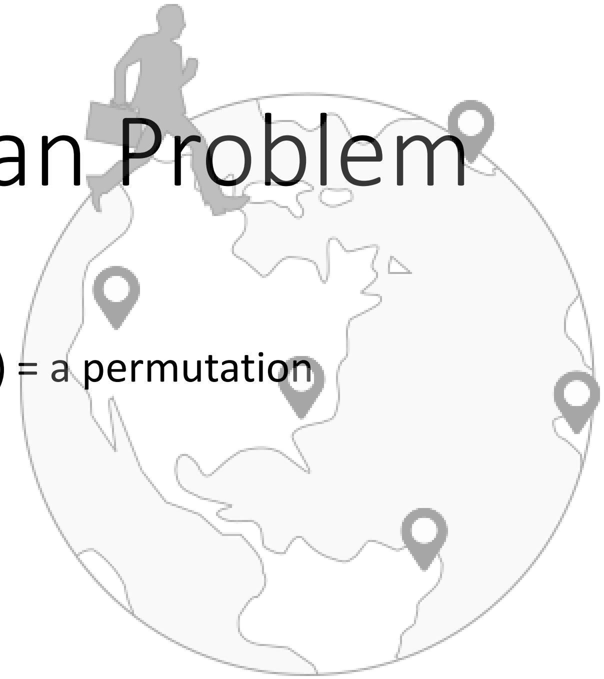
s.t. a has one queen per column

Remember: This makes the problem a lot easier.



h = 5 h = 2 h = 0

Example: Traveling Salesman Problem

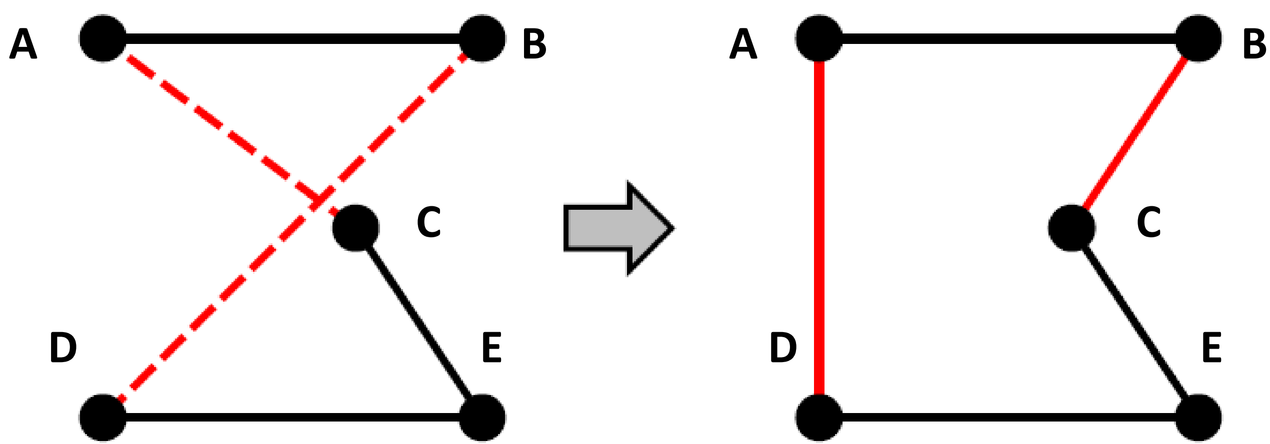


- **Goal:** Find the shortest tour connecting n cities
- **State representation:** tour (order in which to visit the cities) = a permutation
- **State space:** all possible tours
- **Objective function:** length of tour

What's a possible local improvement strategy?

- Start with any complete tour, perform pairwise exchanges.

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



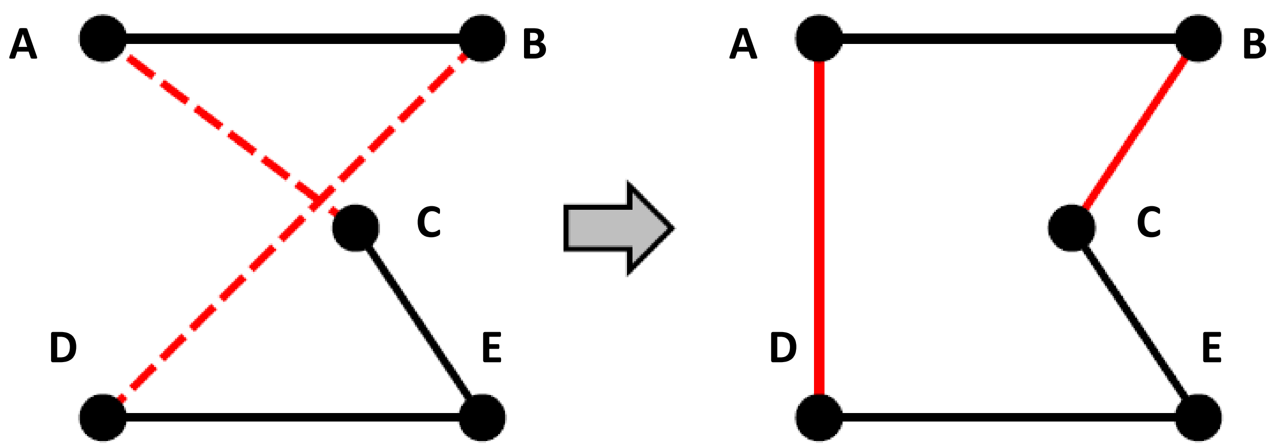
Example: Traveling Salesman Problem

Optimization problem: Find the best tour π

$$\pi^* = \mathrm{argmin}_{\pi} \;\; \mathrm{tourLength}(\pi)$$

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*

Typically, we start with a random state

Variants:

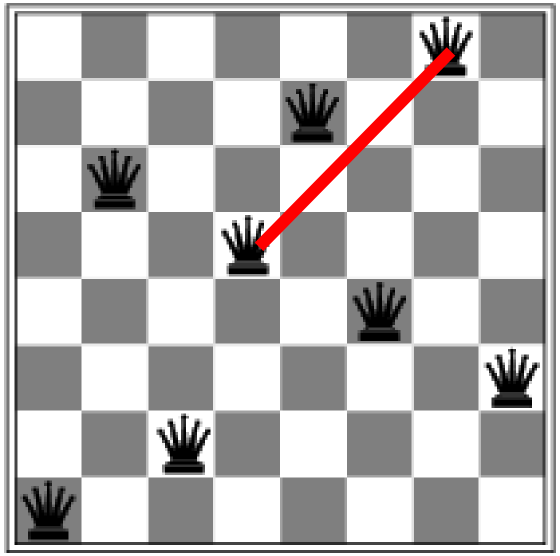
- **Steepest-ascend hill climbing**
  - Check all possible successors and choose the highest-valued successors.
- **Stochastic hill climbing**
  - choose randomly among all uphill moves, or
  - generate randomly one new successor at a time until a better one is found = first-choice hill climbing – **the most popular variant**, this is what people often mean when they say "stochastic hill climbing"
- **Random-restart hill climbing** – to deal with local optima
  - Restart hill-climbing many times with random initial states and return the best solution.

Hill-Climbing Search

Hill-climbing search is like greedy best-first search with the objective function as a (maybe not admissible) heuristic and no frontier (just stops in a dead end).

Is it complete/optimal?

- No – can get stuck in local optima

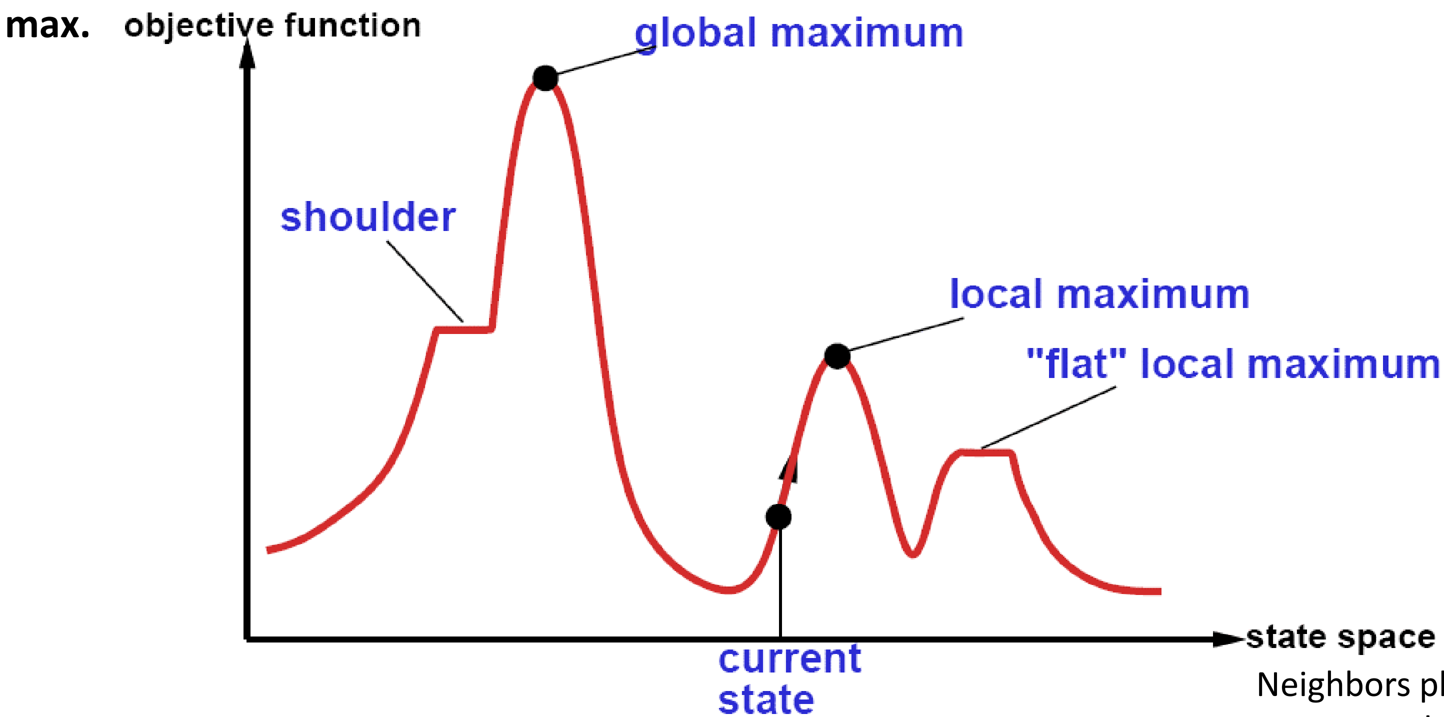


Example: local optimum for the 8-queens problem. No single queen can be moved within its column to improve the objective function.

h = 1

The State Space "Landscape"

We typically can calculate the utility (objective function value) from the state description.



Neighbors placed next to each other

How to escape local maxima?

→ Random restart hill-climbing can help.

What about "shoulders" (called "ridges" in higher dimensional space)?

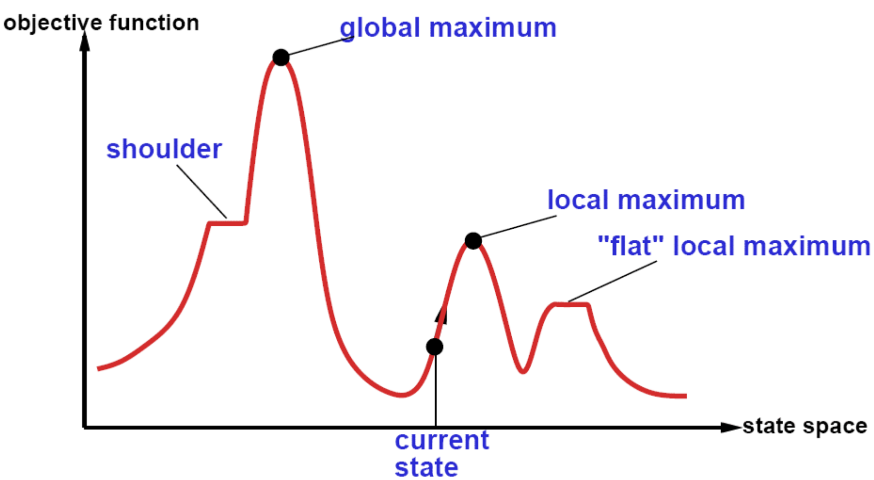
What about "plateaus"?

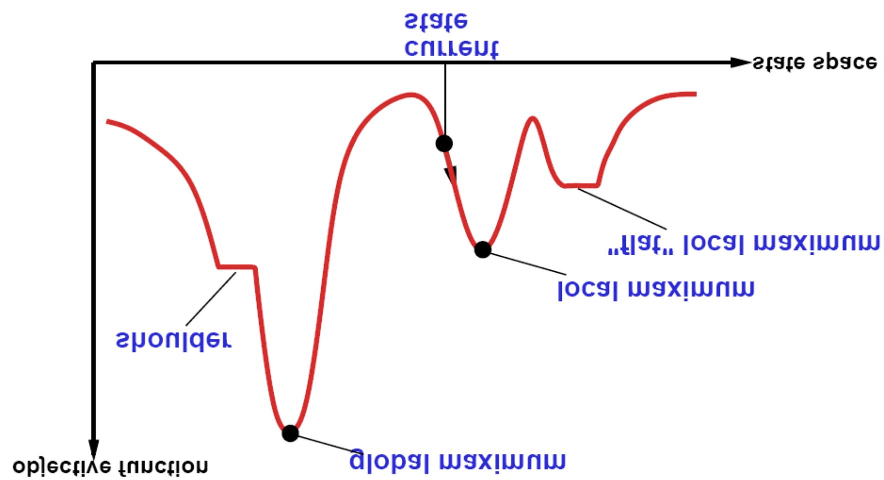
→ Hill-climbing with sideways moves.

Minimization vs. Maximization

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

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

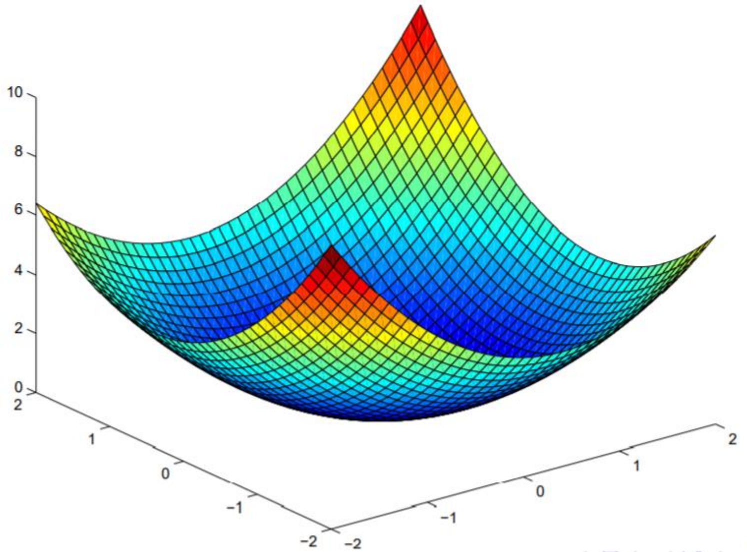




Convex vs. Non-Convex Optimization Problems

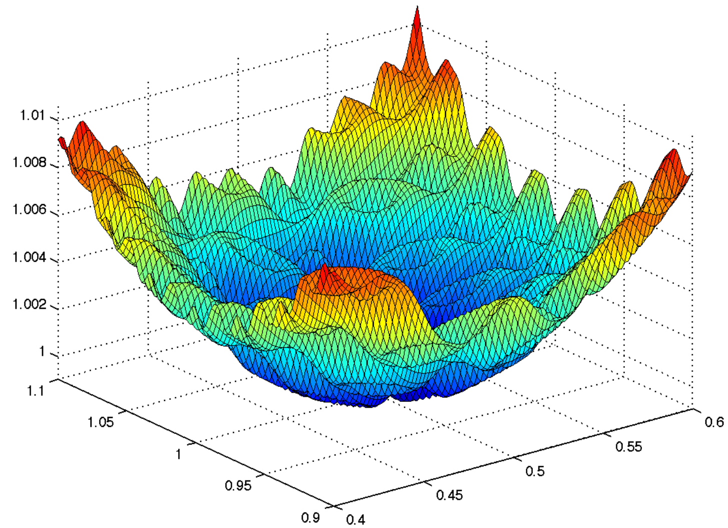
Minimization problems

Convex Problem



One global optimum + smooth function → calculus makes it easy

Non-convex Problem



Many local optima → hard

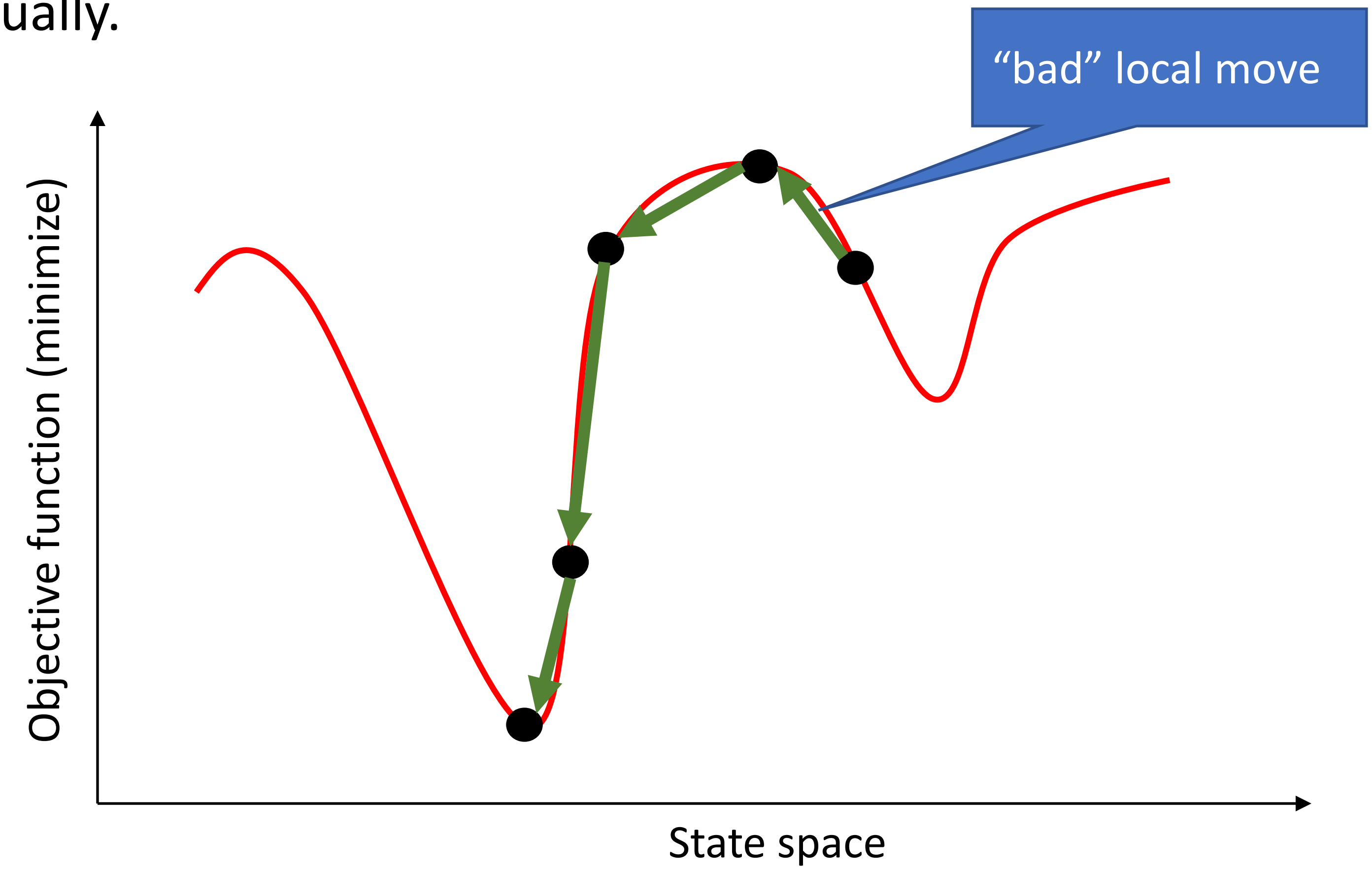
Many discrete optimization problems are like this.

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 (chance of accepting bad moves) gradually.



Simulated Annealing

- **Idea**: First-choice stochastic hill climbing + escape local minima by allowing some "bad" moves but gradually decreasing their frequency as we get closer to the solution.
- 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 ← *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 $e^{-\Delta E/T}$

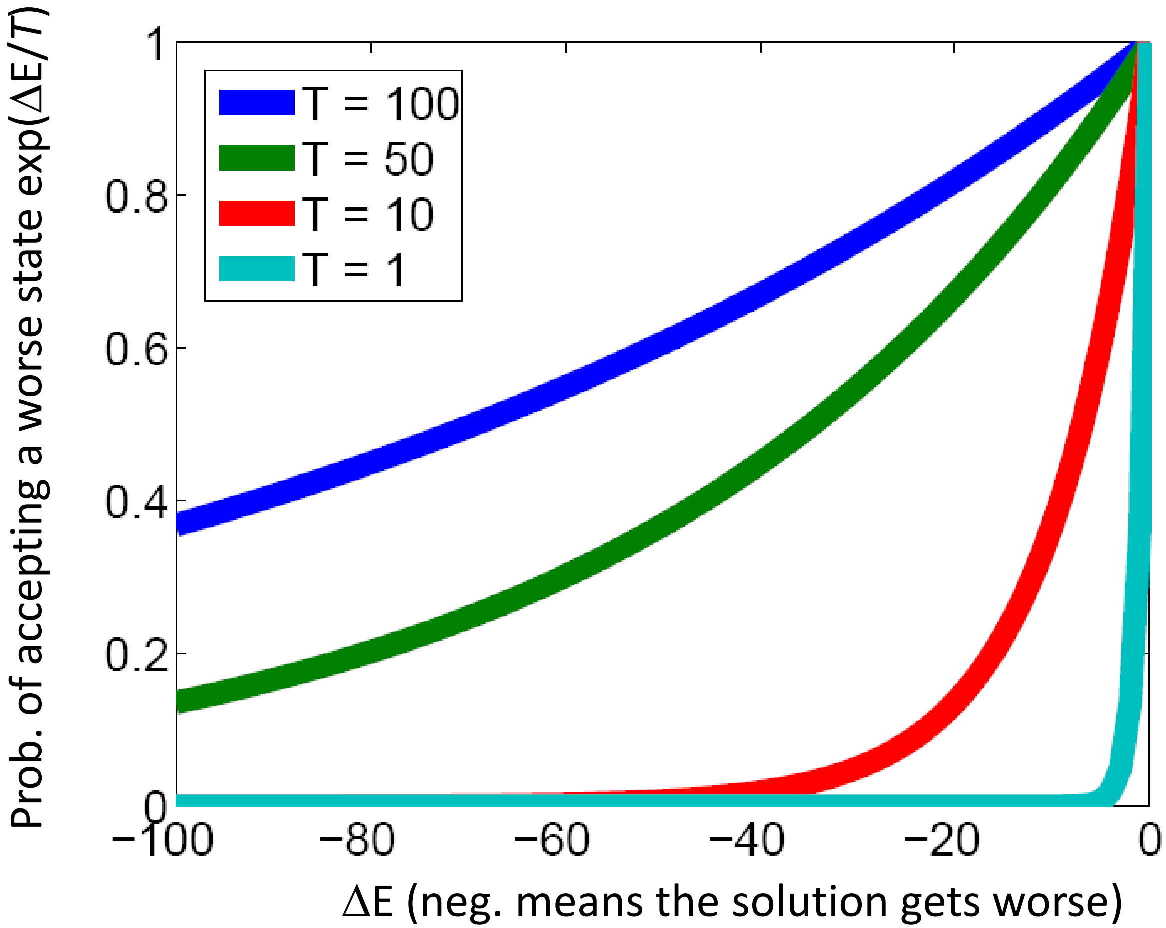
Typically, we start with a random state

Always do good moves

Uses the Metropolis acceptance criterion to accept "bad" moves

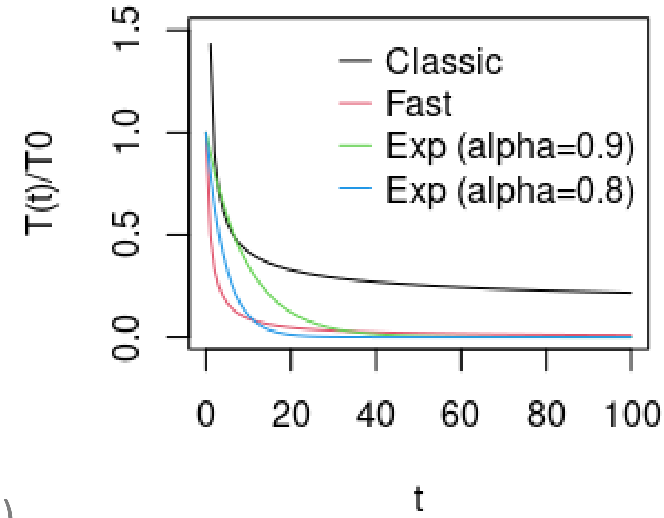
Note: Use VALUE(current) – VALUE(next) for minimization

The 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)

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

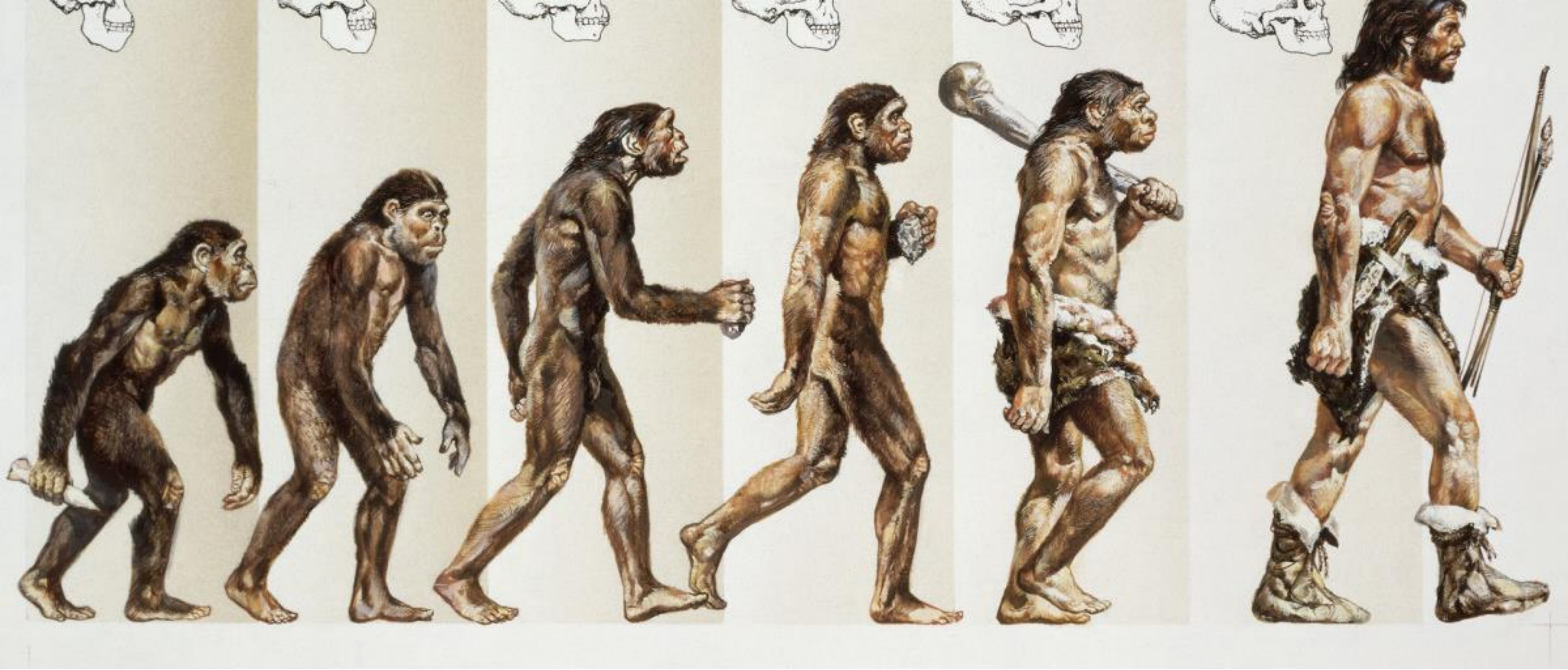
Notes:

- 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 become 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/uphill steps you need to escape a local optimum, the less likely you are to make all of them in a row.

- The related **Markov Chain Monte Carlo (MCMC)** method is a general family of randomized algorithms for exploring complicated state spaces.

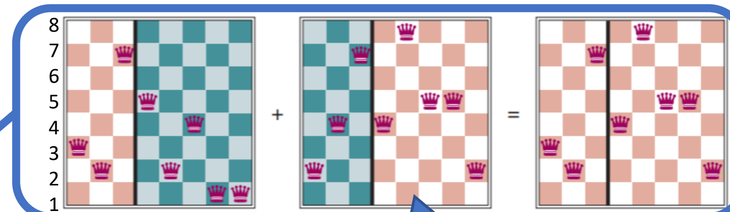


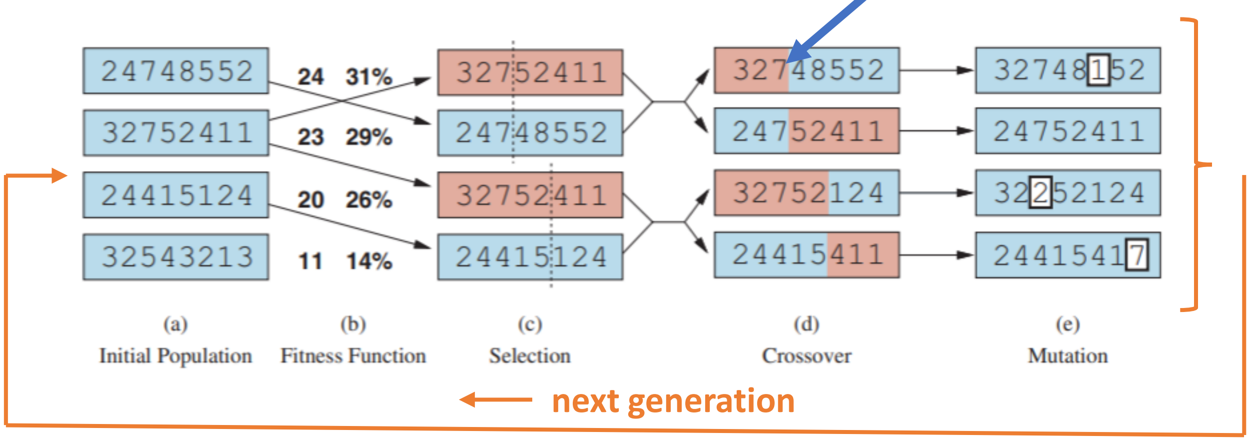
Evolutionary Algorithms

A Population-based Metaheuristics

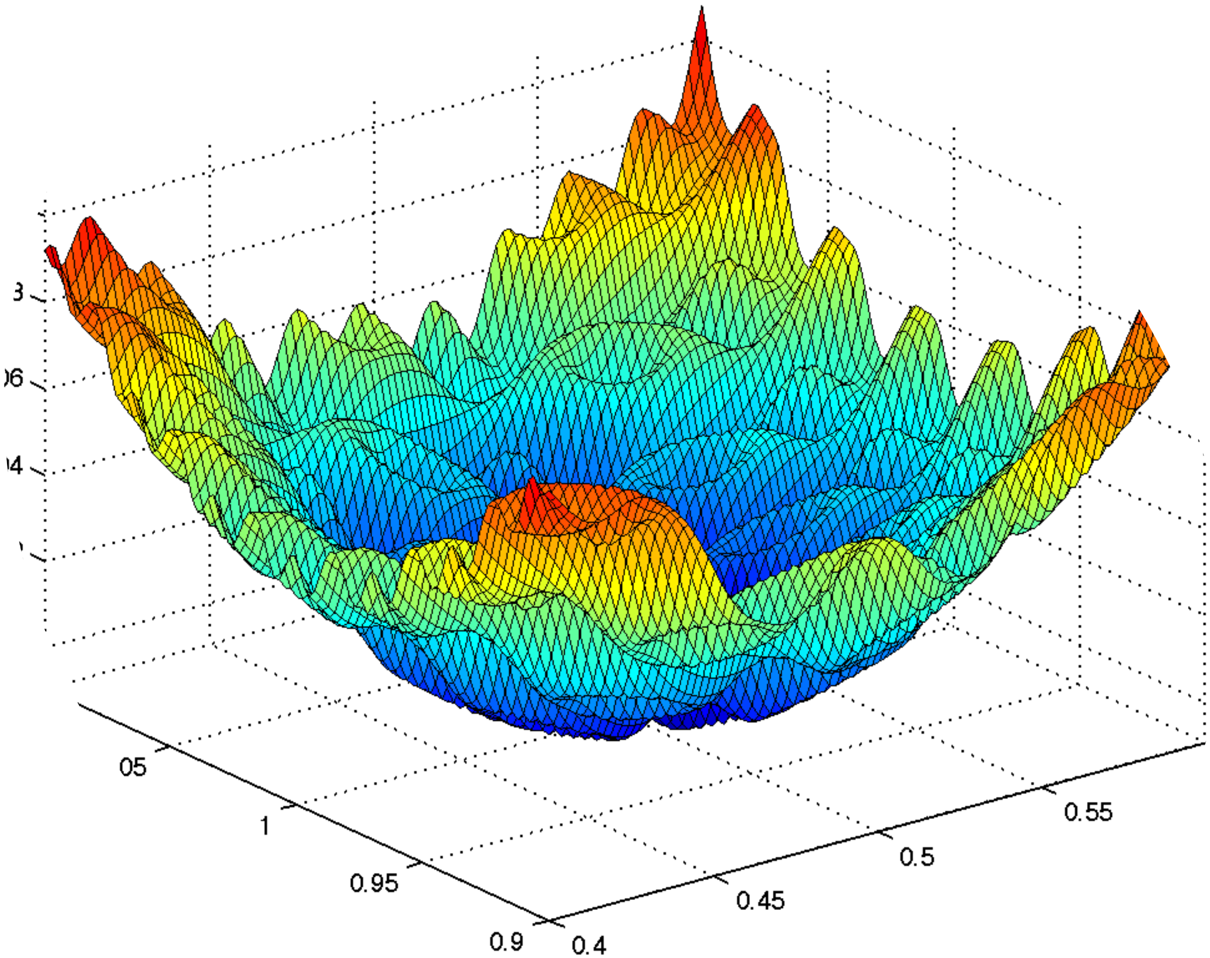
Evolutionary Algorithms / Genetic Algorithms

- 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





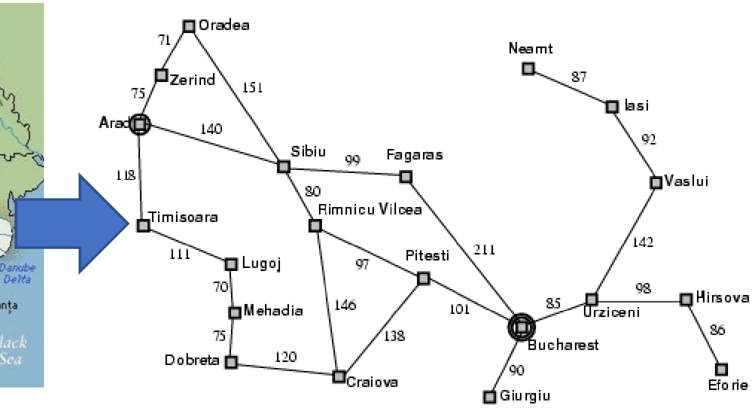
Individual = state **representation** as a chromosome: row of the queen in each column



Search in Continuous Spaces

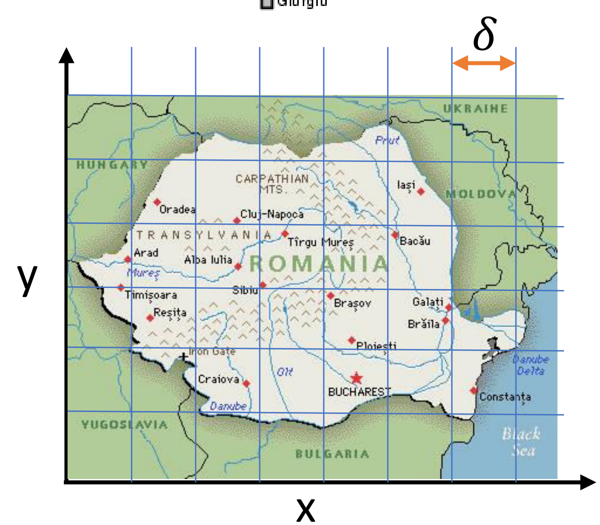
Discretization of Continuous Space

- Use atomic states and create a graph as the transition function.



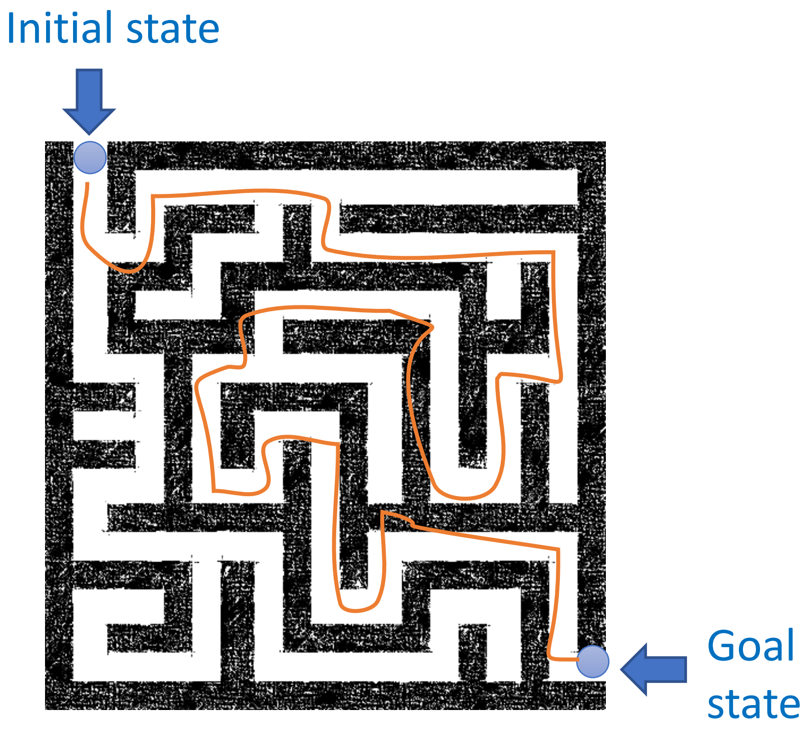
- Use a grid with spacing of size δ

  Note: You probably need a way finer grid!

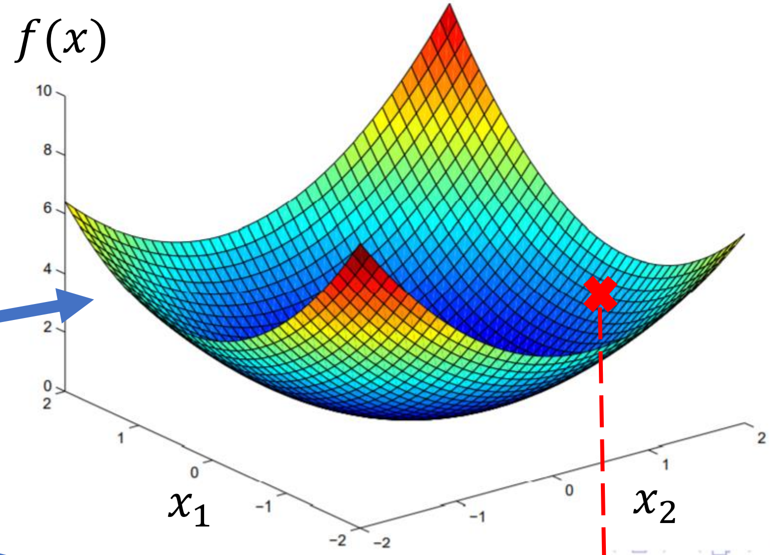


Discretization of Continuous Space

How did we discretize this space?



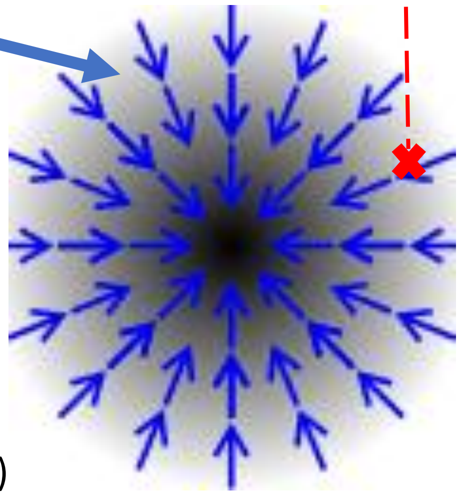
Search in Continuous Spaces: Gradient



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

Gradient at point $\boldsymbol{x}$: $\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots, \frac{\partial f(x)}{\partial x_k}\right)$
(=evaluation of the Jacobian matrix at x)

Find optimum 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 (second-order partial derivative) $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ for the step size α

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

Note: May get stuck in a local optima if the search space is non-convex! Use simulated annealing, momentum or other methods to escape local optima.

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.**