

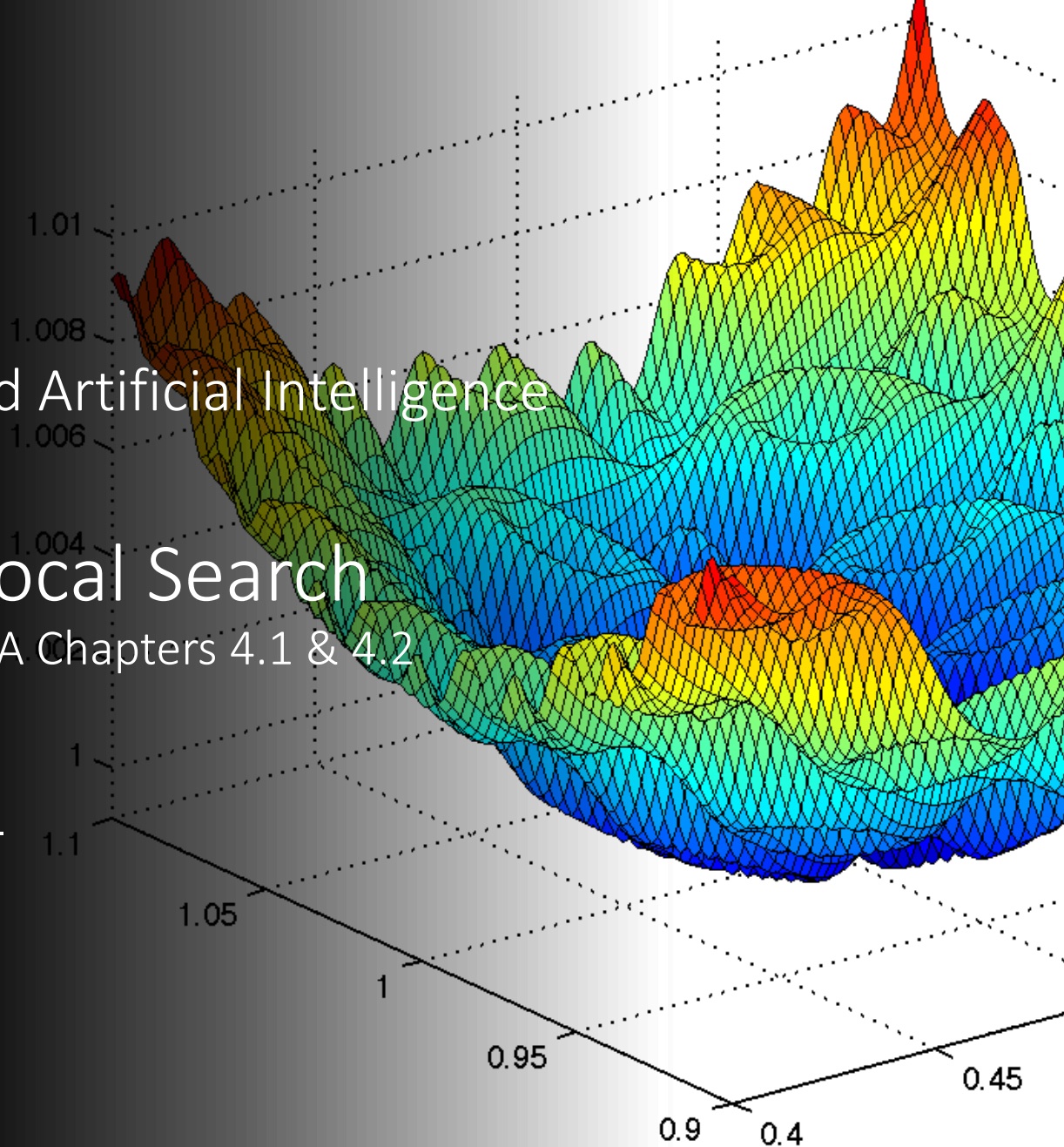


# Advanced Artificial Intelligence

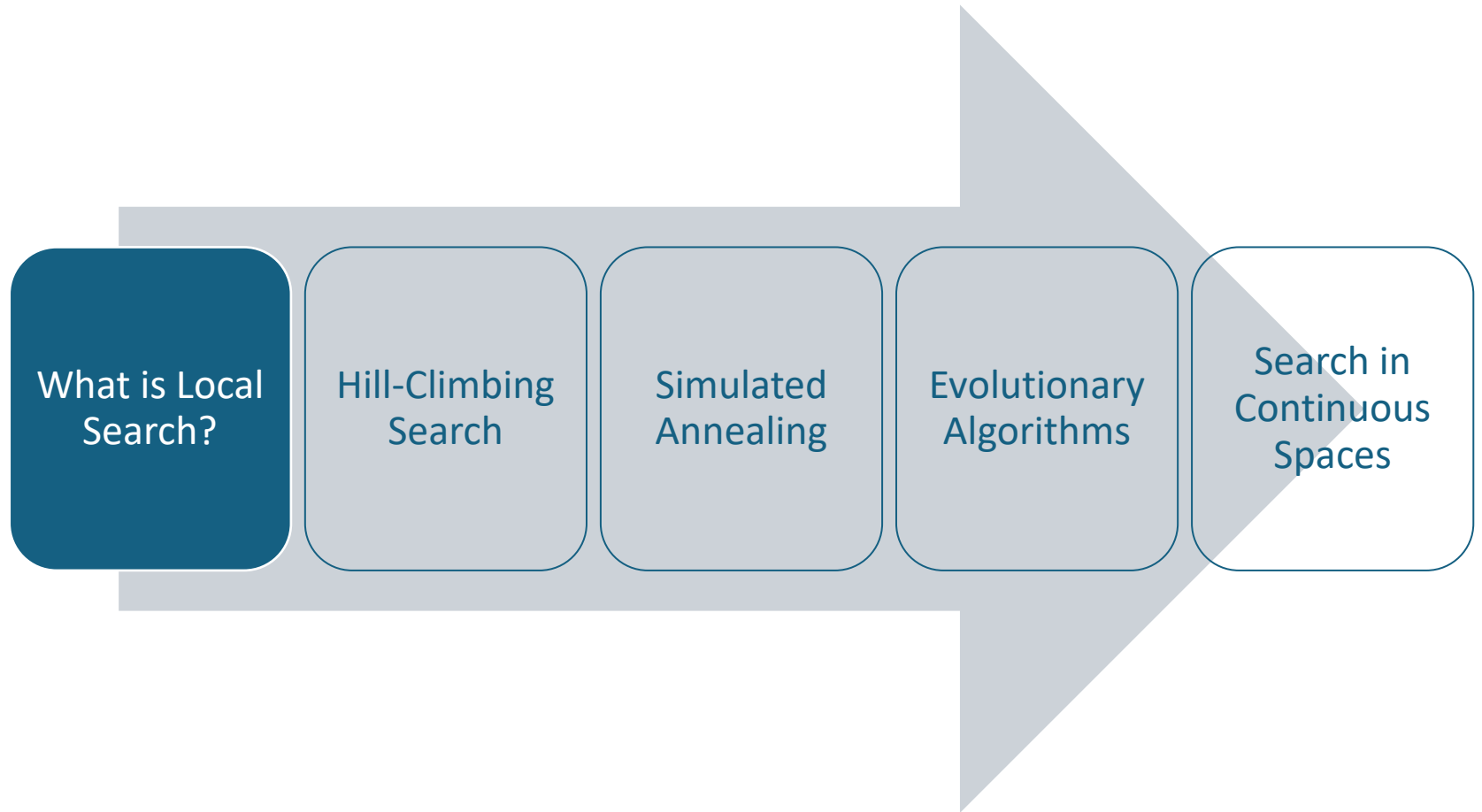
## Local Search

AIMA Chapters 4.1 & 4.2

---



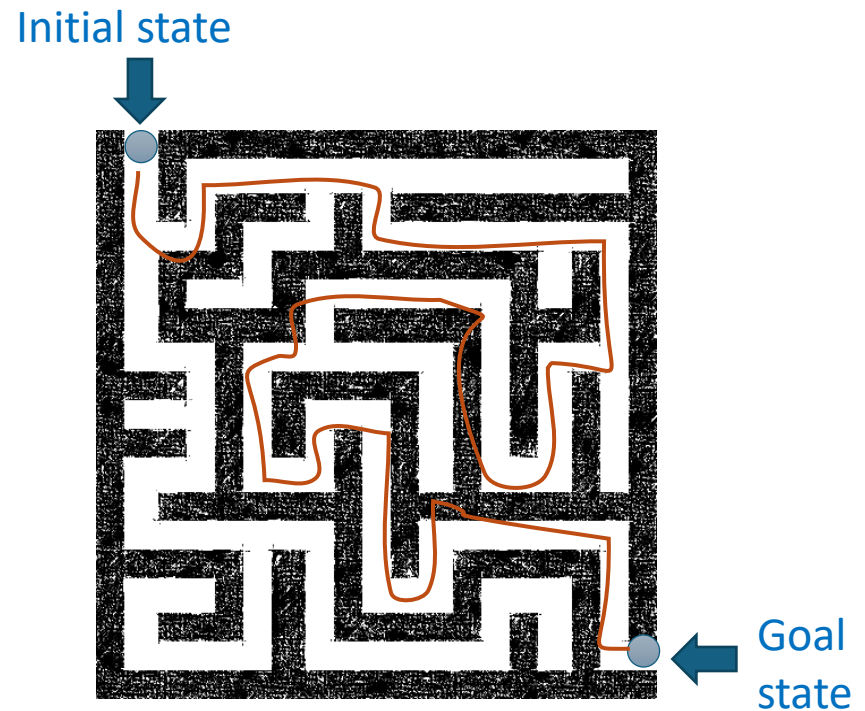
# Contents



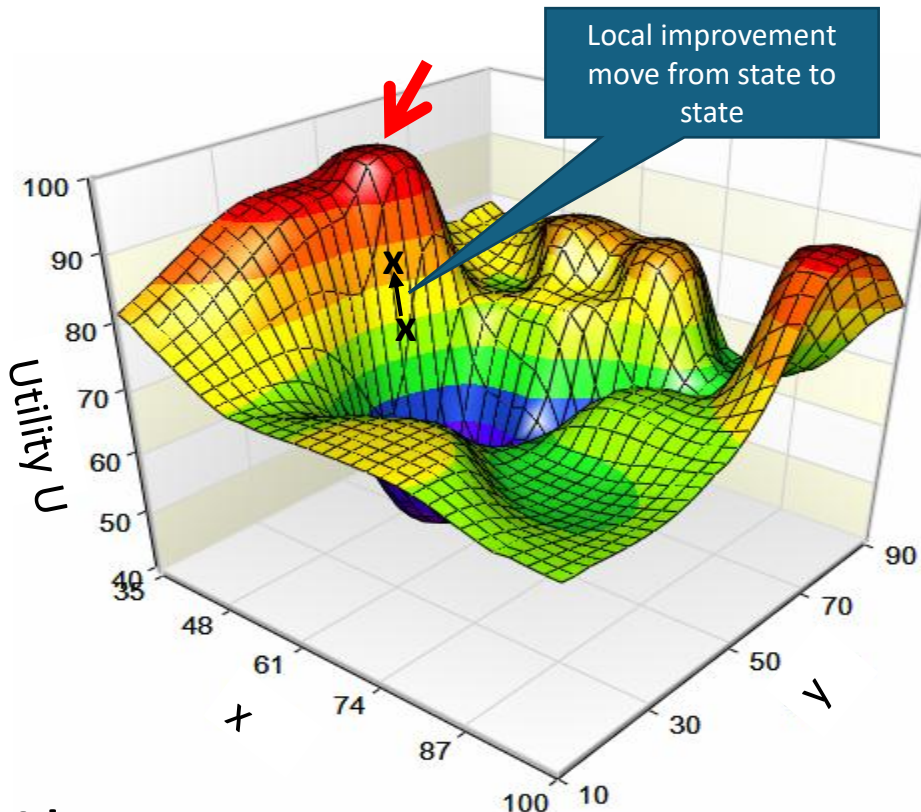
# Recap: Uninformed and Informed Search

Tries to **plan** the  
**best path**  
from a  
**given initial state**  
to a  
**given goal state.**

- Often comes with completeness and optimality guarantees (BFS, A\* Search, IDS).
- **Issue:** Typically have to search a large portion of the search space and therefore need a lot of time and memory.



# Local Search



## Idea:

Start with a current solution (a state) and improve the solution by moving from the current state to a “neighboring” better state (a.k.a. performing a series of **local moves**).

- Assume we know the utility of each possible state given by a utility function
$$U = u(s)$$
- We use a factored state description. Here  $s = (x, y)$
- How can we identify the best or at least a “good” state?
- This is the **optimization problem**:
$$s^* = \operatorname{argmax}_{s \in S} u(s)$$
- We need a fast and memory-efficient way to find the best/a good state.

# Main Differences to Tree Search

- a) **The goal states are unknown**, but we know or can calculate the utility for each state. We want to identify a high-utility state.
- b) Often, no explicit initial state is given, and the **path to the goal and the path cost are not important**.
- c) **No search tree**. Just stores the current state and moves to a “better” state if possible. Needs little memory and only requires very simple code.

# Use of Local Search in AI


- **Goal-based agent:** Identify a good goal state with a good utility before planning a path to that state using a planning agent.
- **Utility-based agent:** Always move to a neighboring state with higher utility. A simple greedy method used for
  - complicated/large state spaces or
  - online search.
- **General optimization:**  $u(s)$  can be replaced by a general objective function. Local search is an effective heuristic to find good solutions in complicated search spaces.  
E.g., stochastic gradient descent to train neural networks (minimize the prediction error)

# Defining A Local Search Problem

- **State space:** How large is the state space?
- **State representation:** How do we define a factored state representation?
- **Objective function:** What is a possible utility function given the state representation?
- **Local neighborhood:** What states are close to each other?

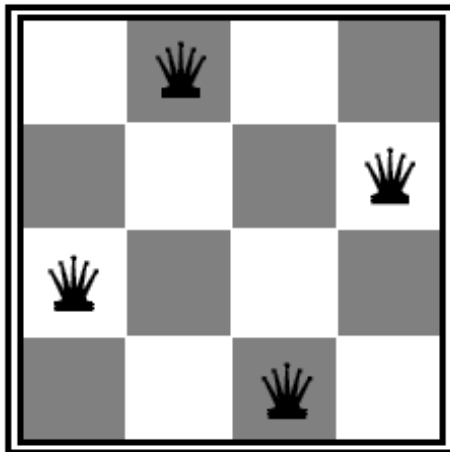
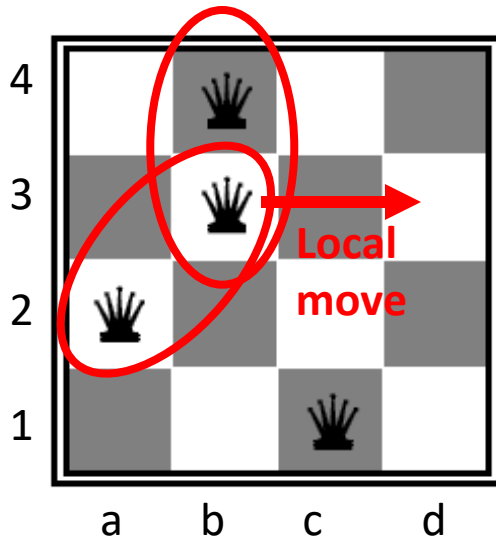


Replaces the goal state



Replaces the transition function.

2 conflicts = utility of -2



0 conflicts = utility of 0

## Example: $n$ -Queens Problem

Place  $n$  queens on a  $n \times n$  chess board so no two queens are in the same row, column or diagonal.

**Defining the search problem:**

**State space:** All possible  $n$ -queen configurations. How many are there?

- 4-queens problem:  $\binom{16}{4} = 1820$

**State representation:** How do we define a factored representation?

- E.g.  $(a2, b3, b4, c1)$

**Objective function:** What is a possible utility function given the state representation?

- Maximizing utility means minimize the number of pairwise conflicts based on the state representation.

**Local neighborhood:** What states are close to each other?

- Move a single queen.

Has its optimum at the goal state. Similar to a heuristic in A\* search. No need to be admissible.






## Example: Traveling salesman problem

Find the shortest tour connecting a given set of cities

- **State space:** all possible tours (states are not individual cities!)
- **State representation:** Order of cities in the tour.
- **Objective function:** minimize the length of the tour.
- **Local neighborhood:** Change the order of visiting a few cities.

**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.

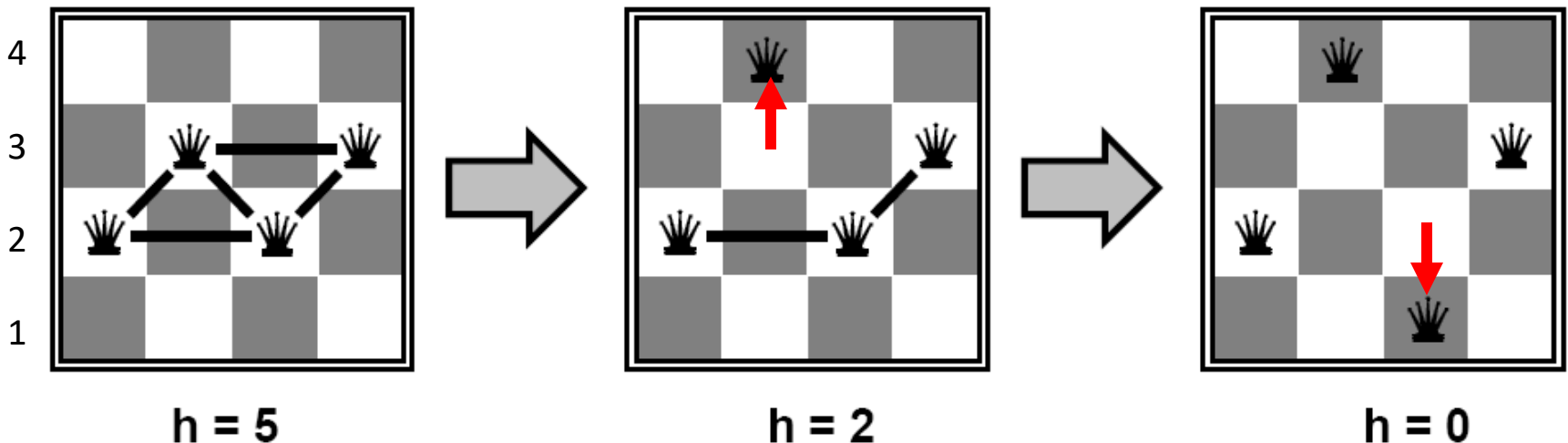
# Example: $n$ -Queens Problem

- **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.
- **State representation:** row position of each queen in its column (e.g., 2, 3, 2, 3)
- **Objective function:** minimize the number of pairwise conflicts.
- **Local neighborhood:** Move one queen anywhere in its column.

4-queens problem: State space is reduced from 1820 to  $4^4 = 256$

## Improvement strategy

- Find a local neighboring state (move one queen within its column) to reduce conflicts.



# Example: $n$ -Queens Problem 2

Find the best local move: **Evaluate the objective function for all local neighbors** (moving a single queen in its column while leaving the others in place).

Objective value after moving the queen in column 1 to this square

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♙  | 13 | 16 | 13 | 16 |
| ♙  | 14 | 17 | 15 | ♙  | 14 | 16 | 16 |
| 17 | ♙  | 16 | 18 | 15 | ♙  | 15 | ♙  |
| 18 | 14 | ♙  | 15 | 15 | 14 | ♙  | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |

Current objective value:  $h = 17$

Best local improvement has  $h = 12$

## Notes:

- There are many options with  $h = 12$ . We typically pick one randomly.
- **Calculating all the objective values may be expensive!**

# Example: $n$ -Queens Problem 3

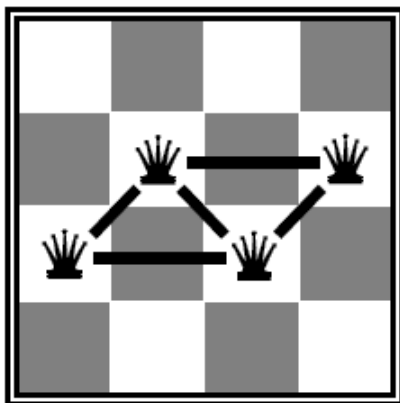
Formulation as an optimization problem:

Find the best state  $s^*$  representing an arrangement of queens.

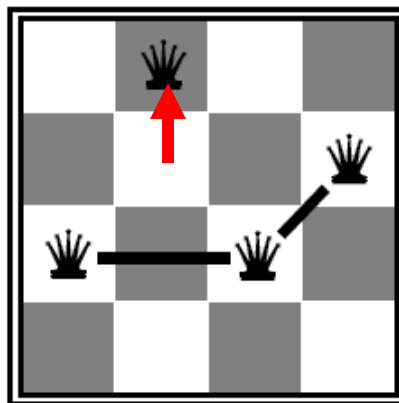
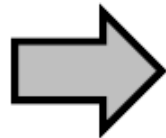
$$s^* = \operatorname{argmin}_{s \in \mathcal{S}} \text{conflicts}(s)$$

subject to:  $s$  has one queen per column

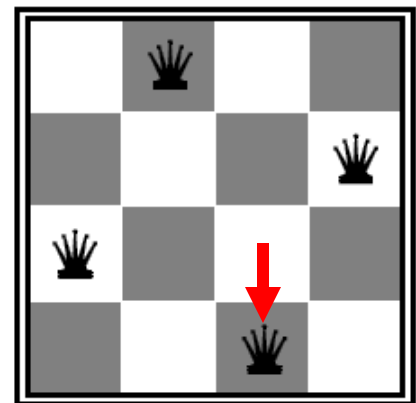
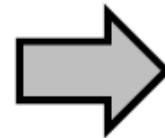
Remember: This makes the problem a lot easier.



$h = 5$



$h = 2$



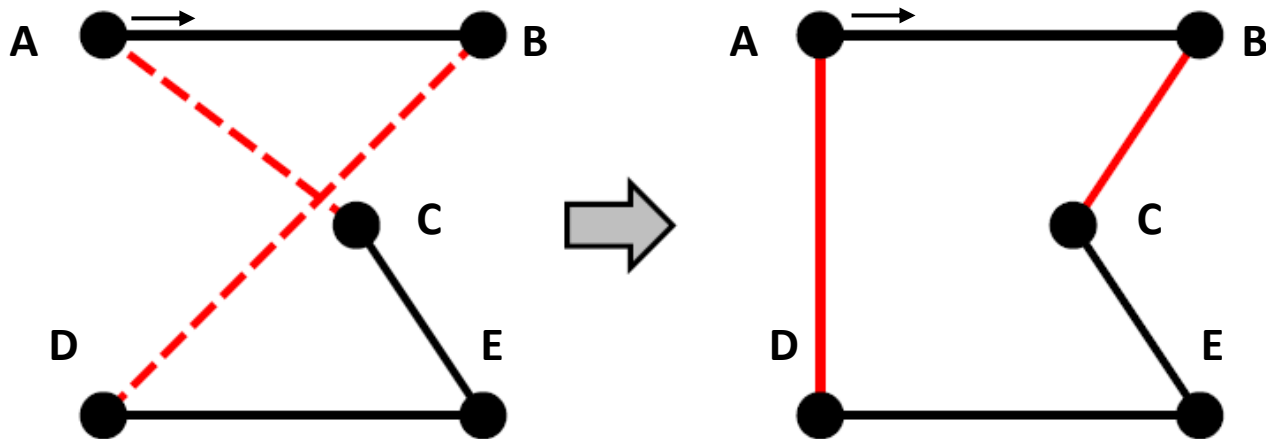
$h = 0$

# Example: Traveling Salesman Problem 2



- **Goal:** Find the shortest tour connecting  $n$  cities
- **State space:** all possible tours
- **State representation:** tour (order in which to visit the cities) = a permutation. There are  $n!$  Many permutations.
- **Objective function:** length of tour
- **Local neighborhood:** reverse the order of visiting a few cities

Local move to reverse the order of cities C, E and D:



State representation  
(permutation):

**ABDEC**

**ABCED**

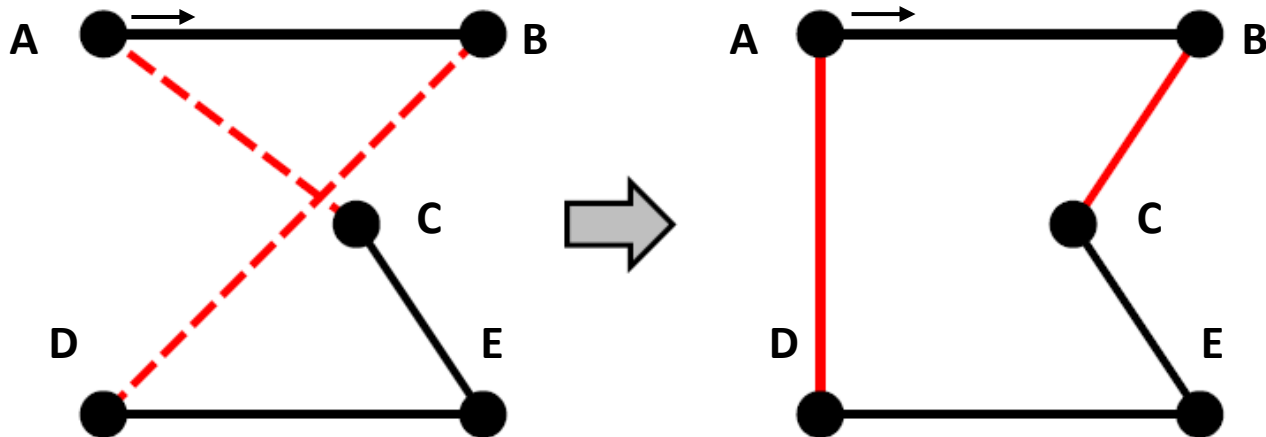
# Example: Traveling Salesman Problem 3

Formulation as an optimization problem:  
Find the best tour  $\pi$

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

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

Local move to reverse the order of cities C, E and D:



State representation: **ABDEC**

**ABCED**



# Hill-Climbing Search (Greedy Local Search)

Maximization

**function** HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

*current*  $\leftarrow$  *problem*.INITIAL

We often start with a random state

**while** *true* **do**

*neighbor*  $\leftarrow$  a highest-valued successor state of *current*

**if** VALUE(*neighbor*)  $\leq$  VALUE(*current*) **then return** *current*

*current*  $\leftarrow$  *neighbor*

Use  $\geq$  for minimization

## Variants:

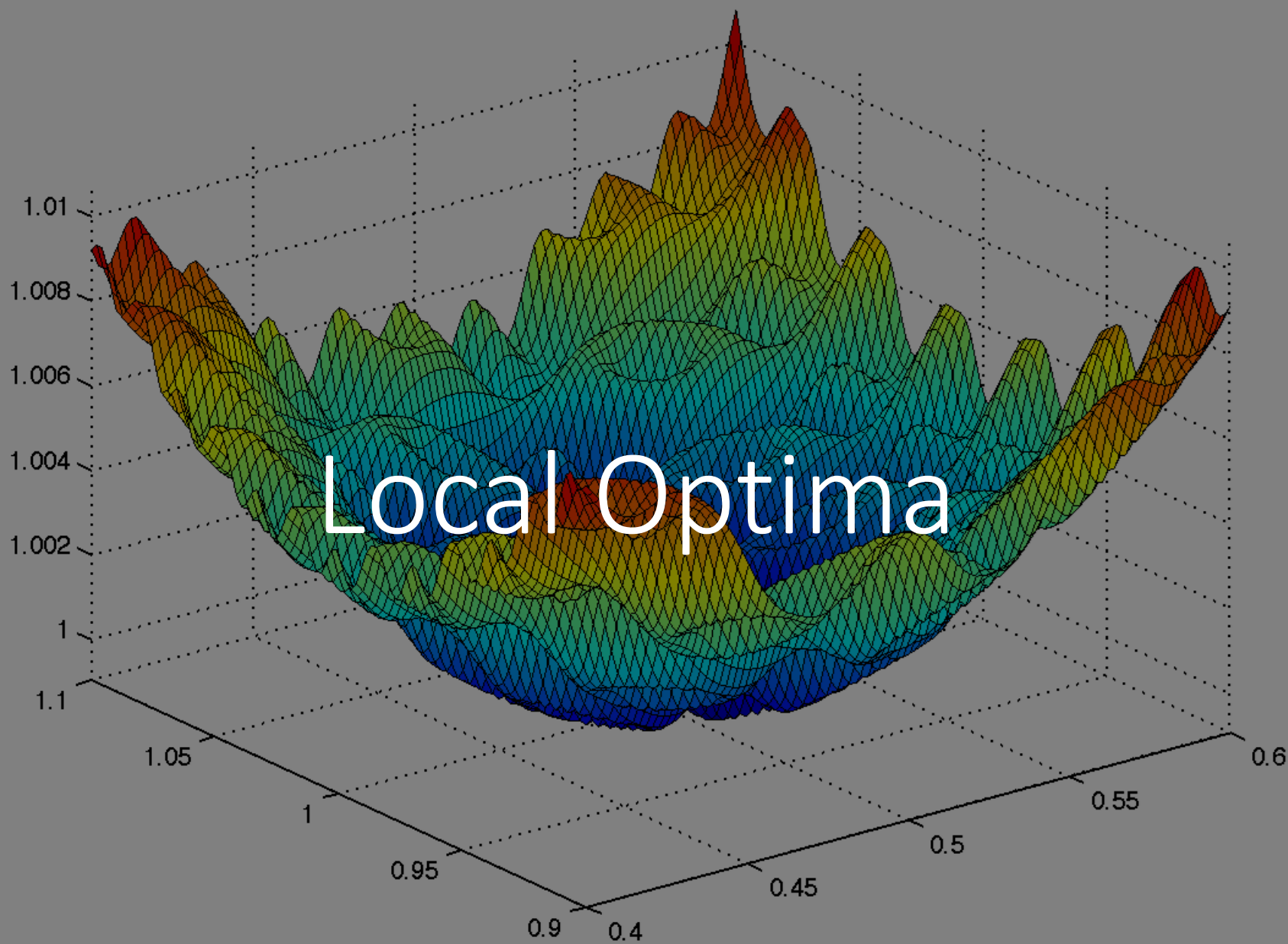
- **Steepest ascent hill climbing:** Check all possible successors and choose the highest-valued successor.
- **Stochastic hill climbing:** Choose randomly among all uphill (improvement) moves.
- **First-choice stochastic hill climbing:** Generate randomly one new successor at a time and only move to better ones. This is what people often mean by “stochastic hill climbing.” It is equivalent to a, but computationally much cheaper.

Minimization

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 18 | 12 | 14 | 13 | 13 | 12 | 14 | 14 |
| 14 | 16 | 13 | 15 | 12 | 14 | 12 | 16 |
| 14 | 12 | 18 | 13 | 15 | 12 | 14 | 14 |
| 15 | 14 | 14 | ♔  | 13 | 16 | 13 | 16 |
| ♔  | 14 | 17 | 15 | ♔  | 14 | 16 | 16 |
| 17 | ♔  | 16 | 18 | 15 | ♔  | 15 | ♔  |
| 18 | 14 | ♔  | 15 | 15 | 14 | ♔  | 16 |
| 14 | 14 | 13 | 17 | 12 | 14 | 12 | 18 |

$$h = 17$$



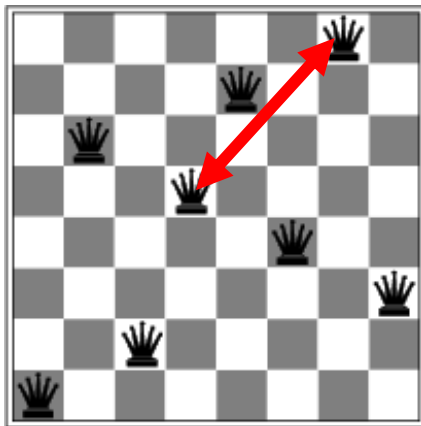


# Local Optima

Hill-climbing search is like greedy best-first search with the objective function as a (maybe not admissible) heuristic. It only stores the current state (has no frontier data structure) and just stops at a dead end.

Is it complete/optimal?

- No – can get stuck in local optima.



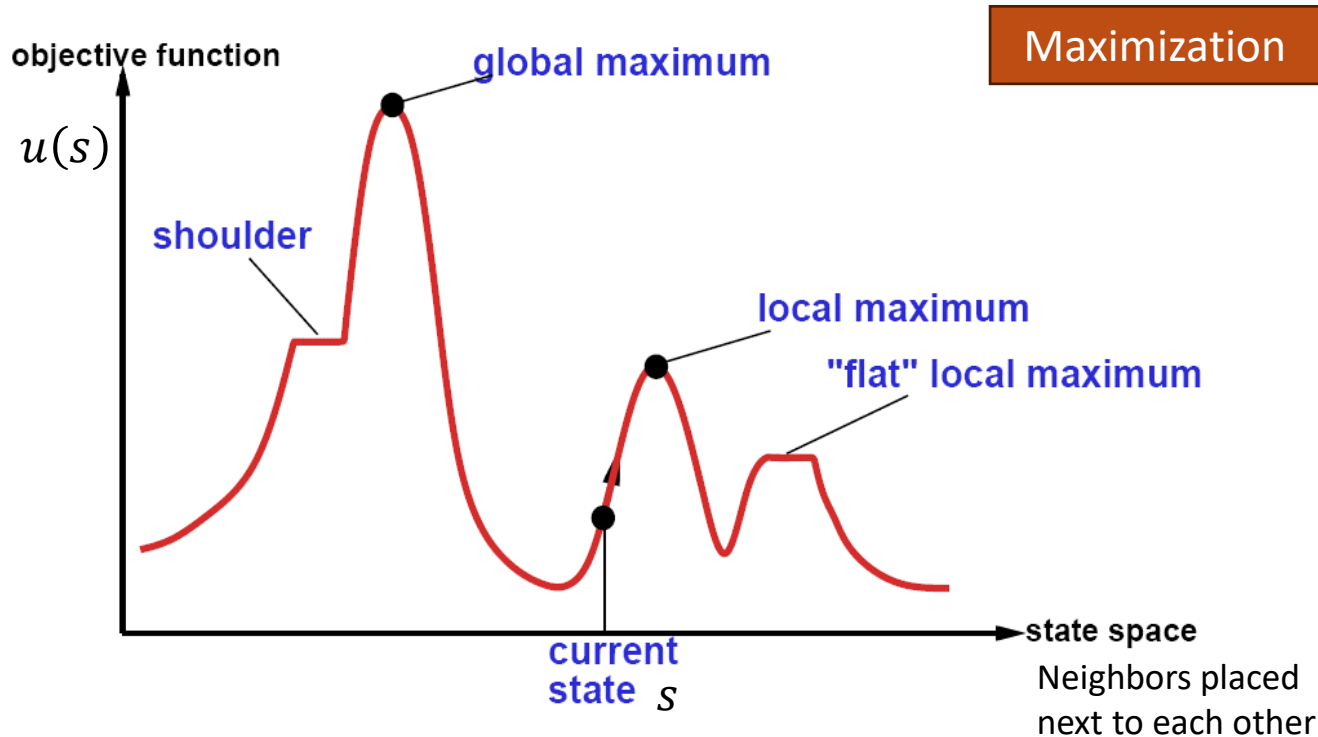
$$h = 1$$

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

Simple approach that can help with local optima:

**Random-restart hill climbing:** Restart hill-climbing many times with random initial states and return the best solution. This strategy can be used for any stochastic (i.e., randomized) algorithm.

# The State Space “Landscape”



How to escape local maxima?

→ Random restart hill-climbing can help.

What about “shoulders” (called “ridges” in higher-dimensional spaces)?

→ Hill-climbing that allows sideways moves and uses momentum.

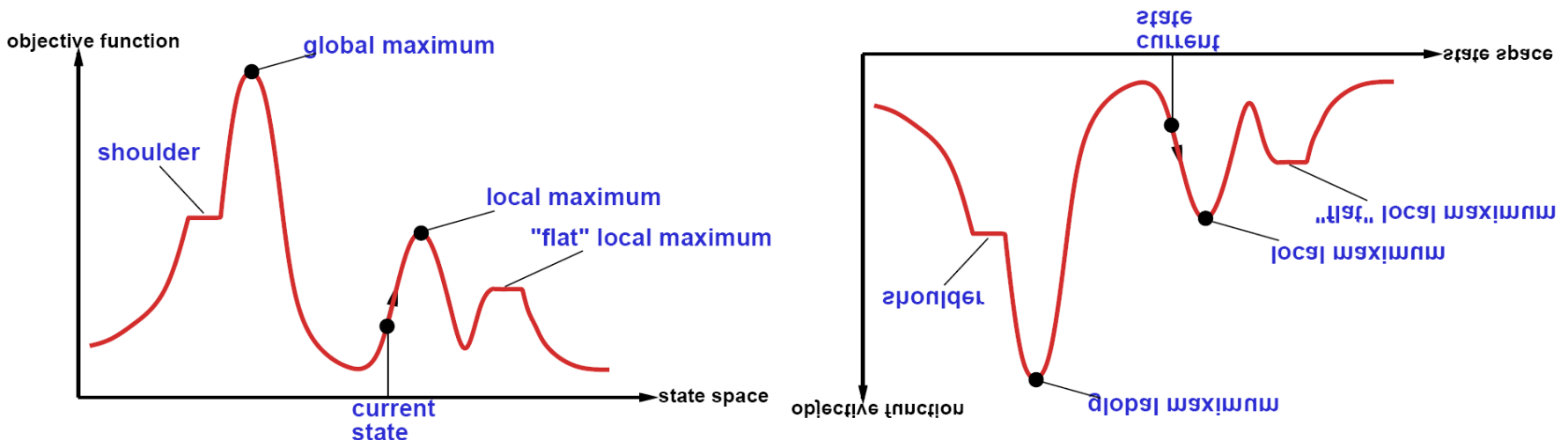
# Minimization vs. Maximization

- The name **hill climbing** used in AI implies **maximizing a function**.
- Optimizers often prefer to state problems as **minimization** problems and refer to hill climbing as **gradient descent**.
- Minimization and maximization are equivalent problems:

$$\max(f(x))$$

$\Leftrightarrow$

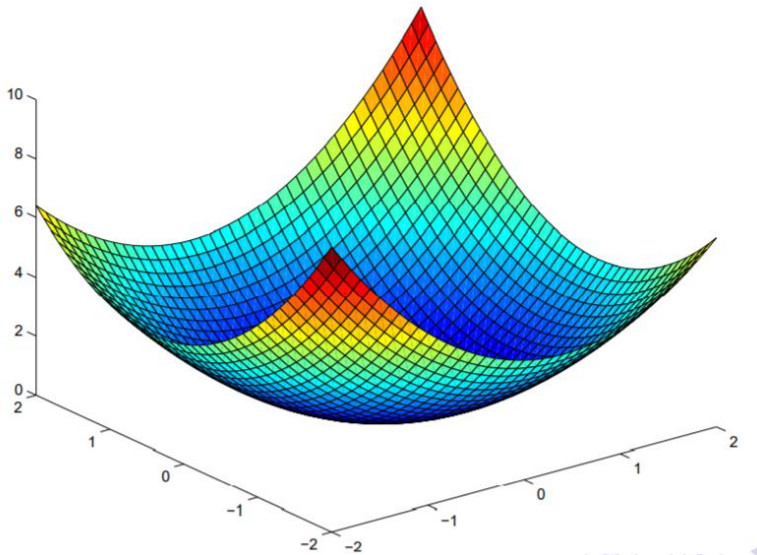
$$\min(-f(x))$$



# Convex vs. Non-Convex Optimization Problems

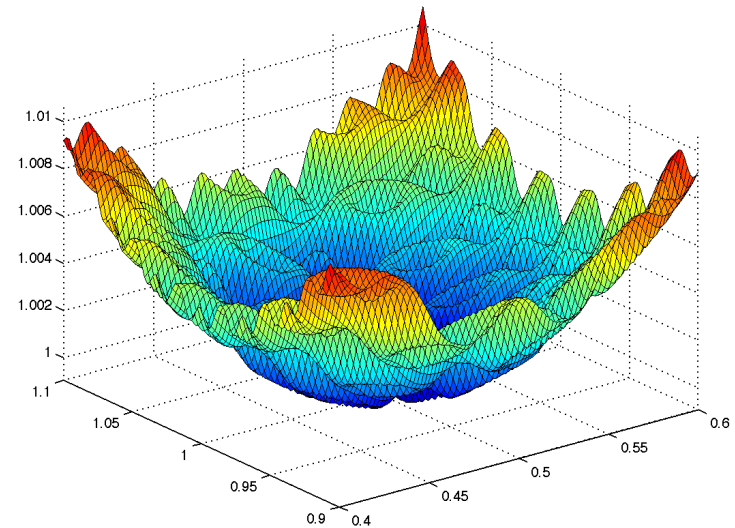
## Minimization

Convex Problem



One global optimum +  
continuous smooth function  
→ calculus makes it easy  
(solve  $f'(x) = 0$ )

Non-convex Problem



Many local optima → hard

Many AI problems are in addition discrete  
(the objective function is not differentiable).  
We often have to settle for a local optimum.



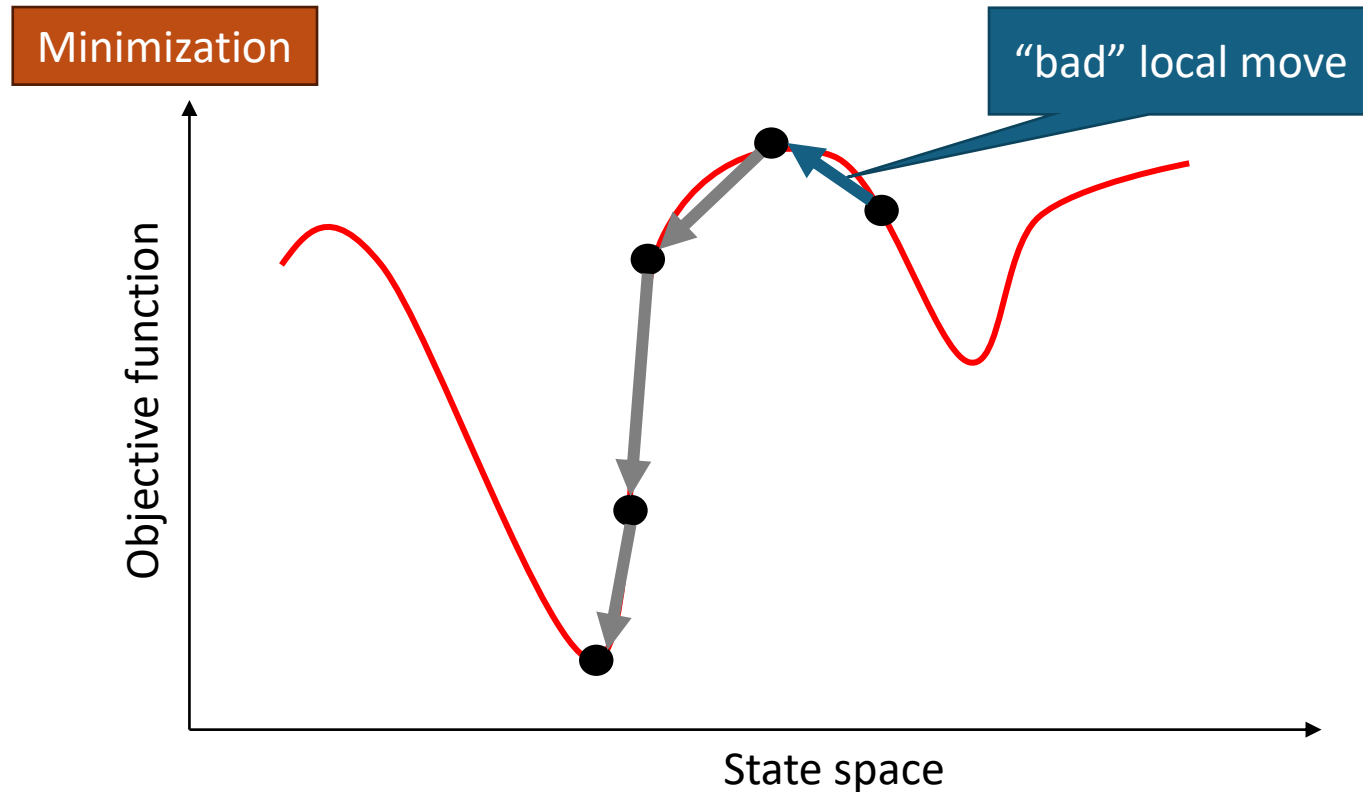


# Simulated Annealing

Using heat to escape local optima...

# Idea of Simulated Annealing

- Use first-choice stochastic hill climbing + escape local minima by **allowing some “bad” moves** but gradually decreasing their frequency.
- Inspired by the process of controlled cooling of glass or metals. Decreasing the temperature means decreasing the chance of accepting bad moves.



# Simulated Annealing Algorithm

- Use 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.
- Annealing tries to reach a low energy state: A negative  $\Delta E$  means the solution gets better.
- The probability of accepting “bad” moves follows the annealing schedule, which reduces the temperature  $T$  over time  $t$ .

Maximization

**function** SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

*current*  $\leftarrow$  *problem*.INITIAL

Typically, we start with a random state

**for**  $t = 1$  **to**  $\infty$  **do**

$T \leftarrow \text{schedule}(t)$

**if**  $T = 0$  **then return** *current*

*next*  $\leftarrow$  a randomly selected successor of *current*

$\Delta E \leftarrow \text{VALUE}(\text{current}) - \text{VALUE}(\text{next})$

**if**  $\Delta E \leq 0$  **then** *current*  $\leftarrow$  *next*

**else** *current*  $\leftarrow$  *next* only with probability  $e^{-\Delta E/T}$

Always accept good moves that reduce the energy.

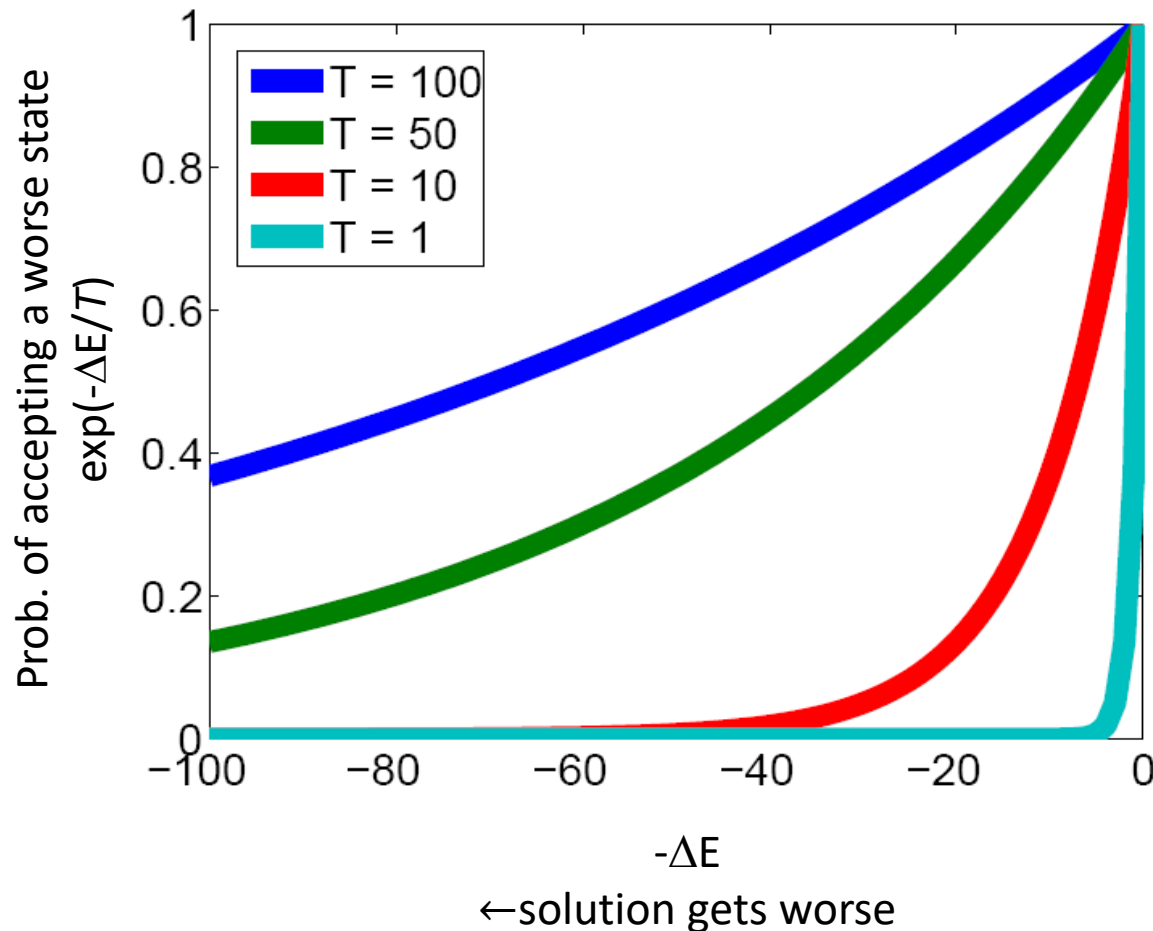
Accept “bad” moves with a probability inspired by the acceptance criterion in the Metropolis–Hastings MCMC algorithm.

Note: Use  $\text{VALUE}(\text{next}) - \text{VALUE}(\text{current})$  for minimization



# The Effect of Temperature

Convert the changes due to “bad” moves into an acceptance probability depending on the temperature. The criterion uses the negative part of the exponential function.



# Cooling Schedule

The cooling schedule is very important.

Popular schedules for the temperature at time  $t$ :

- **Classic simulated annealing:**  $T_t = T_0 \frac{1}{\log(1+t)}$
- **Exponential cooling** (Kirkpatrick, Gelatt and Vecchi; 1983)

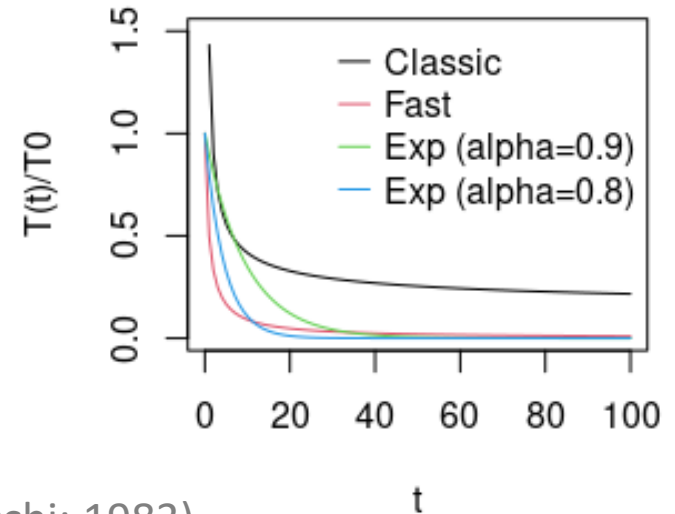
$$T_t = T_0 \alpha^t \quad \text{for } 0.8 < \alpha < 1$$

- **Fast simulated annealing** (Szy and Hartley; 1987)

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

Notes:

- Choose  $T_0$  to provide a high probability  $p_0 = e^{-\frac{\Delta E}{T_0}}$  that any move will be accepted at time  $t = 0$ .  $\Delta E$  is determined by the worst possible move.
- $T_t$  will not become 0 but very small. Stop when  $T < \epsilon$  ( $\epsilon$  is a very small constant).
- The best schedule (cooling rate) is typically determined by trial-and-error. The goal is to have a low chance of getting stuck in a local optima.



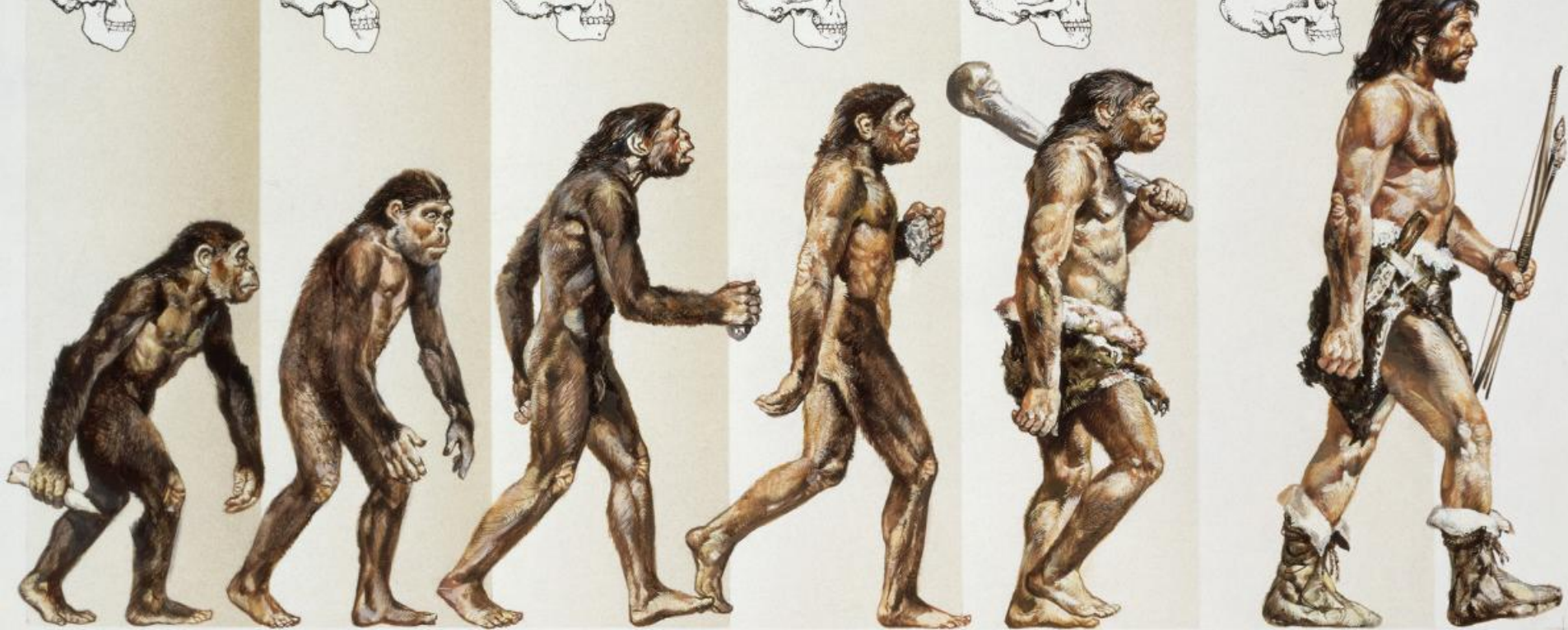
# Simulated Annealing Search

## Guarantee

If the temperature is decreased **slowly enough**, then simulated annealing search will find a **global optimum** with a probability approaching one.

However:

- This usually takes an impractically long time.
- The best cooling schedule and local move need to be determined experimentally.



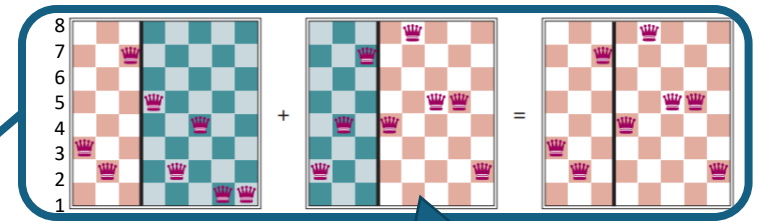
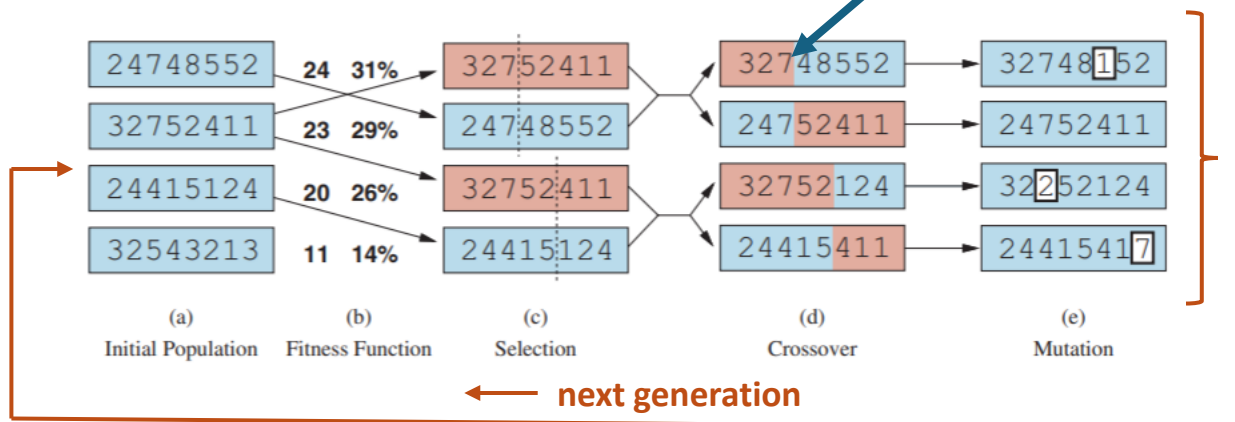
# Evolutionary Algorithms

A Population-based Metaheuristics

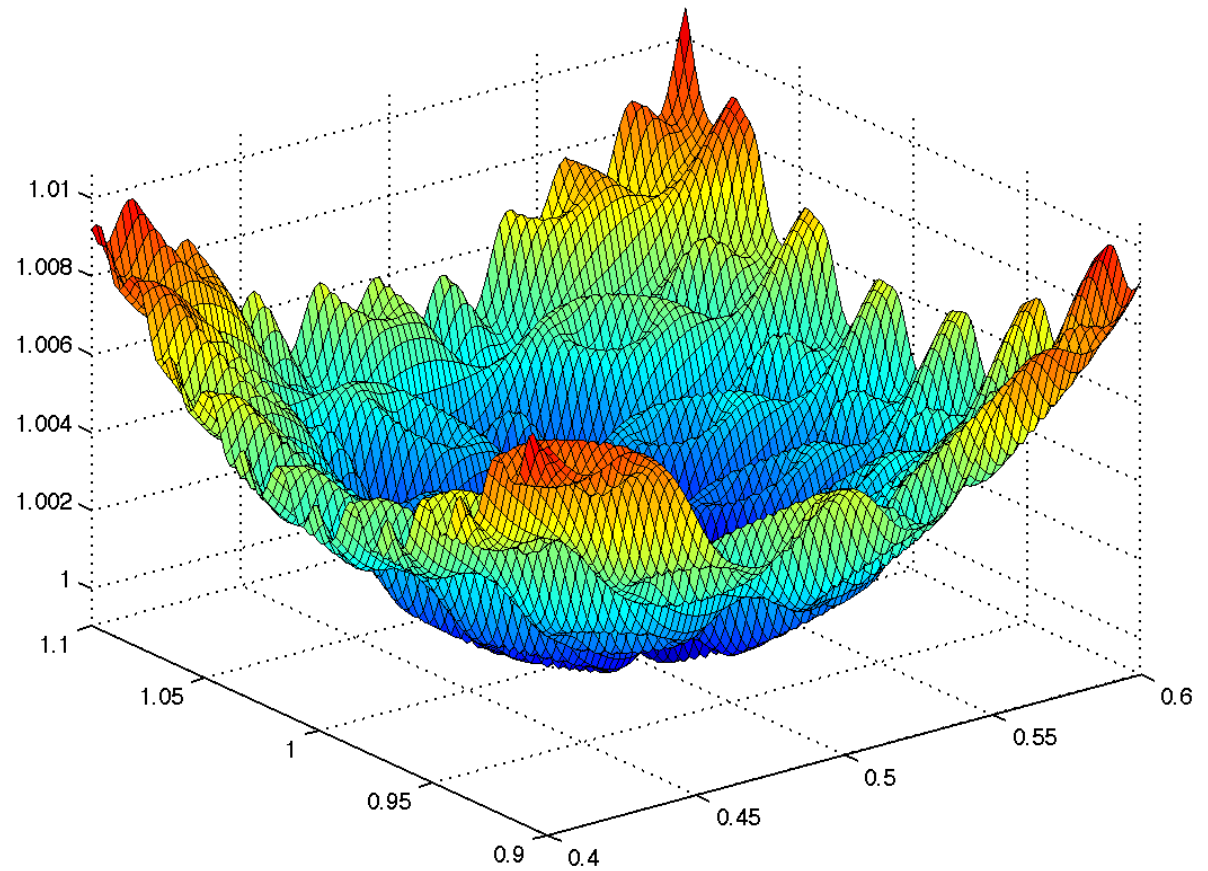
# Evolutionary Algorithms / Genetic Algorithms

- 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



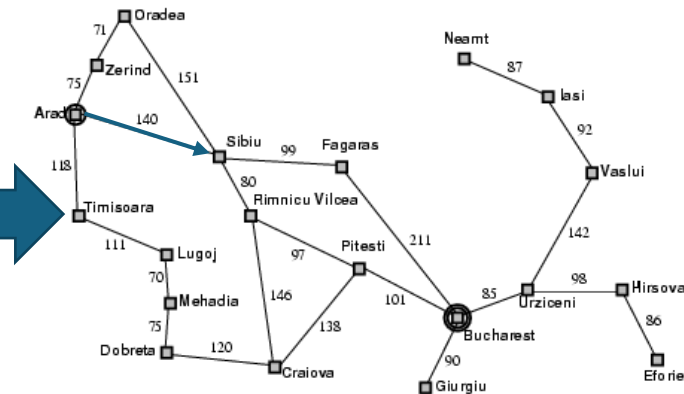
Individual = state  
**representation** as  
a chromosome:  
row of the queen  
in each column



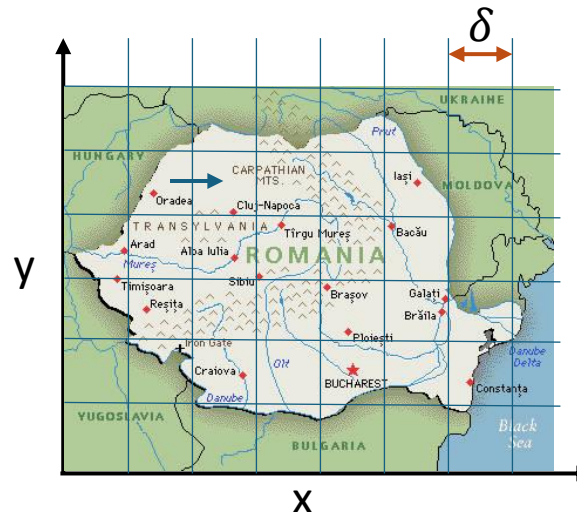
# Search in Continuous Spaces

# Methods: Discretization of Continuous Space

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



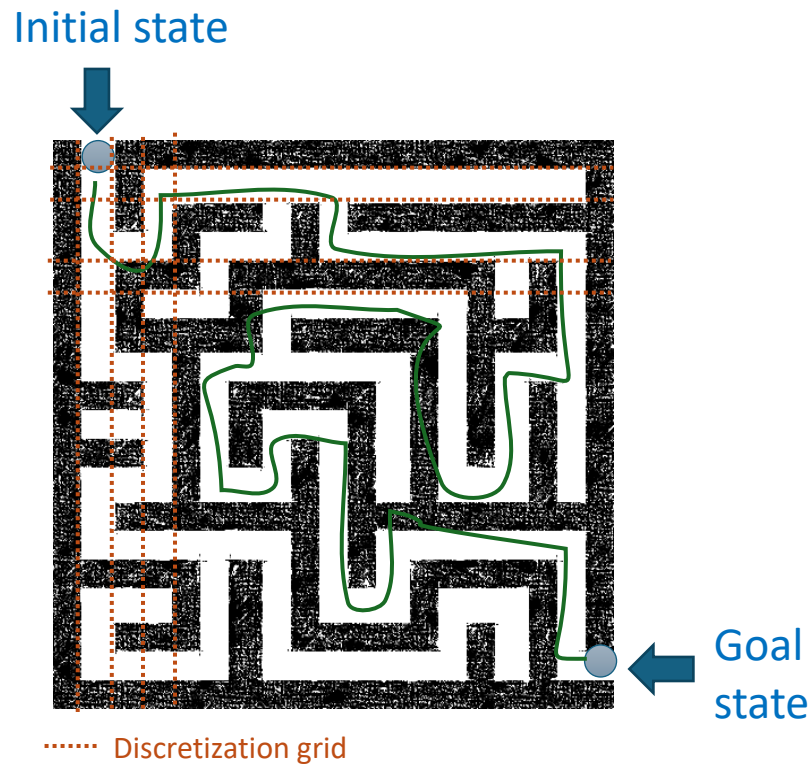
- Use a grid with spacing of size  $\delta$   
Note: You probably need a way finer grid!





# Example: Discretization of Continuous Space

How did we discretize this space?





# Search in Continuous Spaces: Gradient Descent

**State representation:**  $\mathbf{x} = (x_1, x_2, \dots, x_k)$

**State space size:** infinite

**Objective function:**  $\min f(\mathbf{x})$

**Local neighborhood:** small changes in  $x_1, x_2, \dots, x_k$

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

Find optimum by solving:  $\nabla f(\mathbf{x}) = 0$

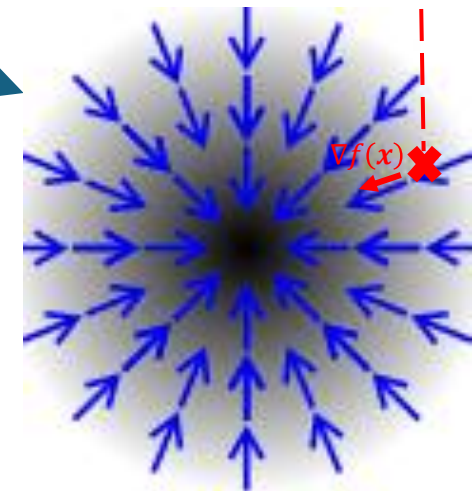
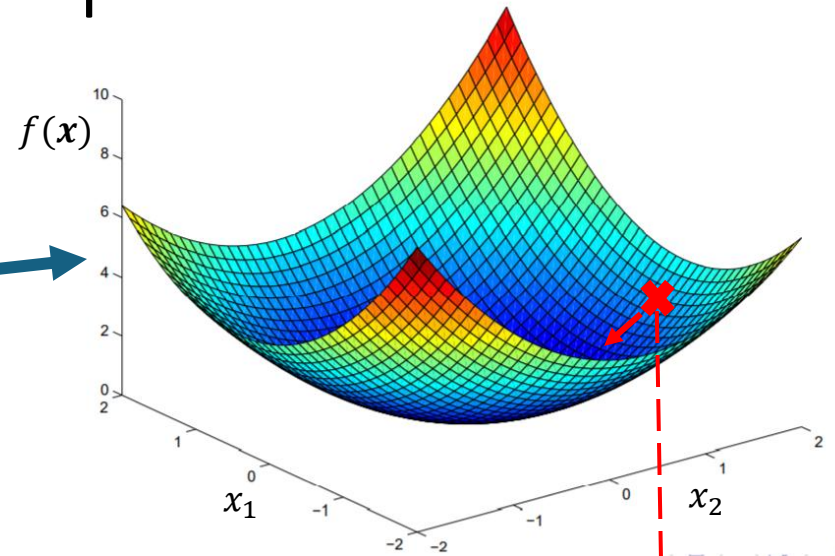
- **Gradient descent (= Steepest-ascent hill climbing for minimization)**  
with step size  $\alpha$  (typically reduced over time)

$$\text{Repeat: } \mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$$

- **Newton-Raphson method**  
uses the inverse of the Hessian matrix (second-order partial derivative of  $f(\mathbf{x})$ )

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \text{ as the optimal step size}$$

$$\text{Repeat: } \mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$



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

# Search in Continuous Spaces: Stochastic Gradient Descent

- 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 perform gradient descent with an approximation of the gradient using the data points as a sample. This is called **stochastic gradient descent (SGD)**.

→ We will talk more about search in continuous spaces with loss functions using gradient descent when we discuss **parameter learning for learning from examples (machine learning)**.



---

## Conclusion

- Local search provides a fast method to find good solutions to many difficult optimization problems.
- Local optima are a big issue that can be addressed with random restarts and simulated annealing.