上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

# Lecture 5: Local Search & Optimization

Tao Huang

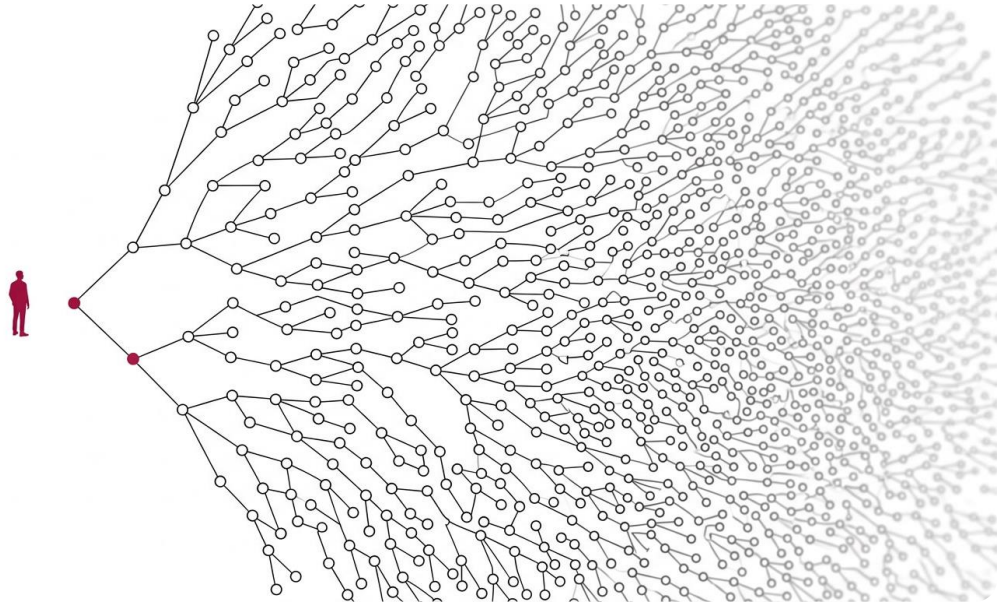
John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

# Huge Search Spaces

- Many real-world problems have enormous search spaces
- Exploring the entire search tree is impossible
- Need faster ways to find good solutions (scheduling, chip design, etc.)



many problems have astronomical numbers of possible states

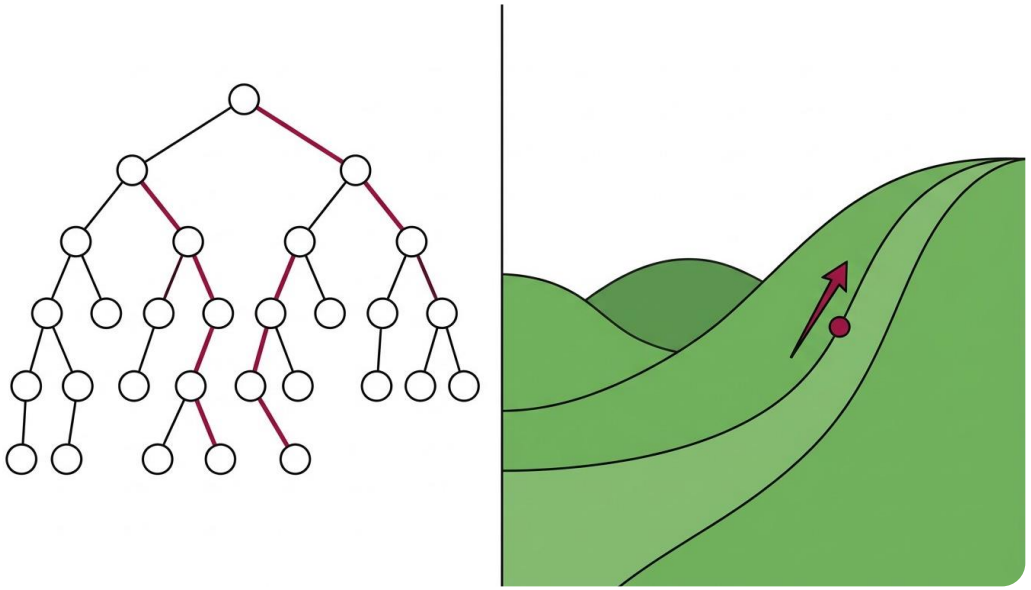
# Two Types of Search

## Classical Search

- Explores a search tree
- Keeps many states in memory
- Finds optimal paths to goal

## Local Search

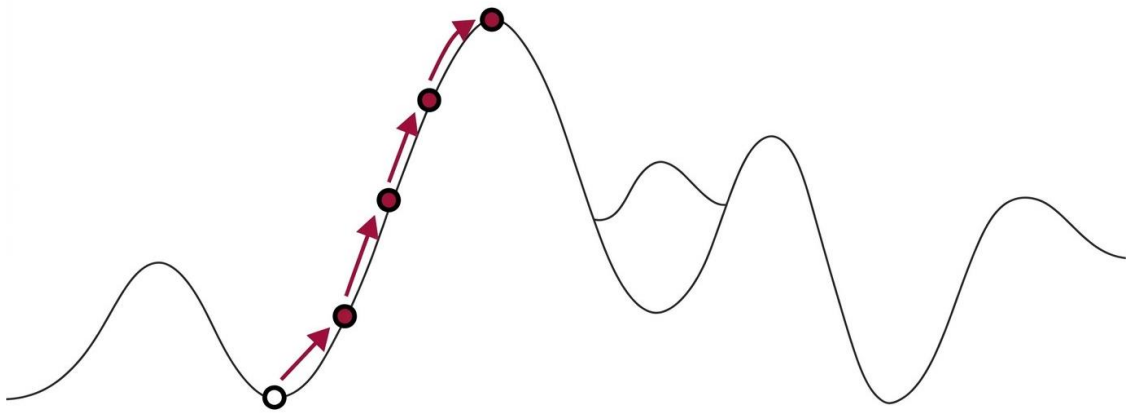
- Keeps only one current state
- Moves to neighboring states
- Focuses on improving quality



tree search vs optimization search

# Local Search

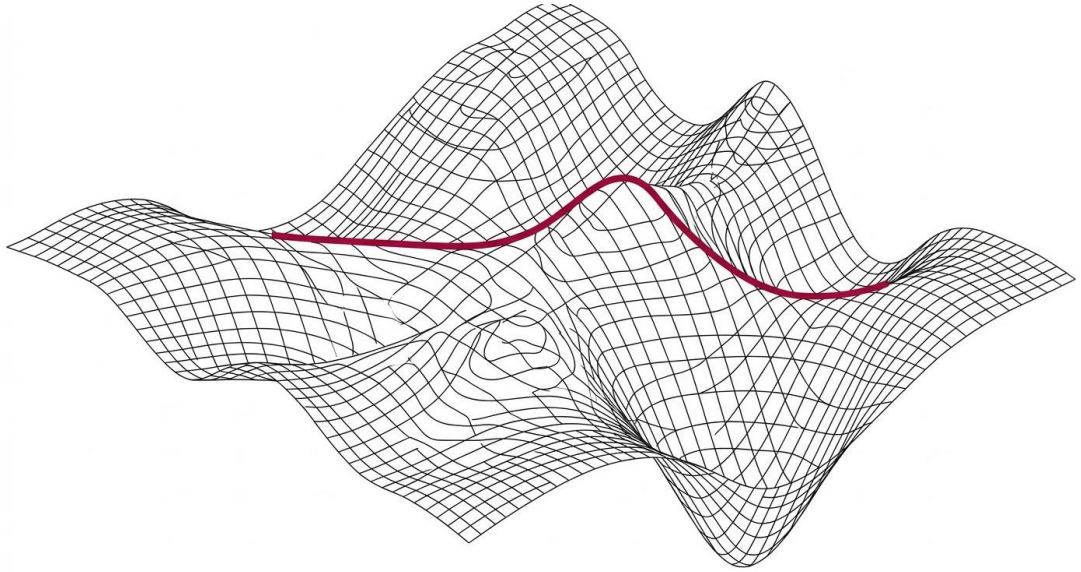
- Repeatedly move to a better neighbor
- Gradually improve solution quality
- **Goal:** find a high-quality state



moving toward better states

# Optimization Landscape

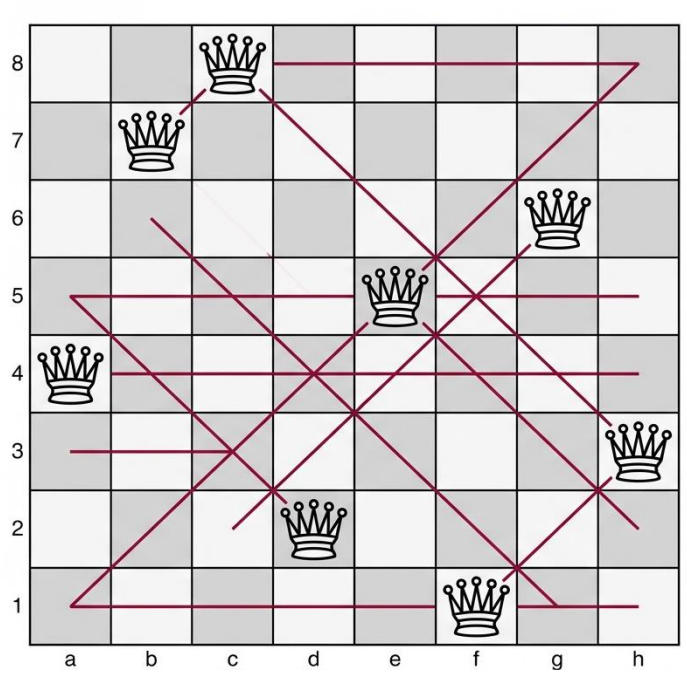
- Every state has a score or value
- **Objective function** defines the landscape
- Local search is a walk on this surface



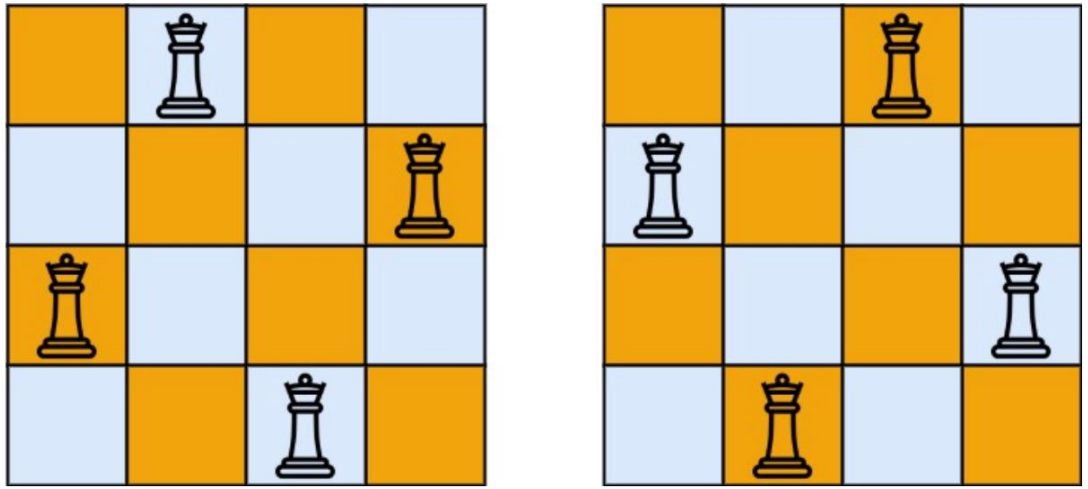
objective function defines the landscape

# Example: N-Queens

- **Goal:** place N queens with no attacks
- **Constraint:** no queens on same row/column/diag
- **Idea:** one queen per column




the N-Queens problem

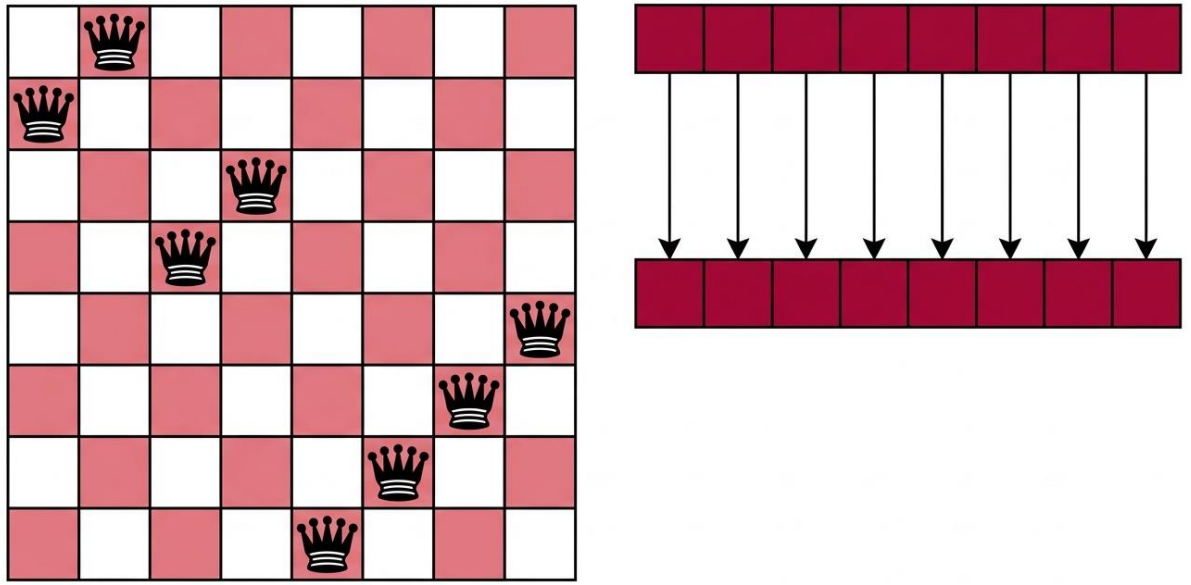


two solutions of 4-Queens

# Representing a State

State vector: `[2 1 4 3 8 7 6 5]`

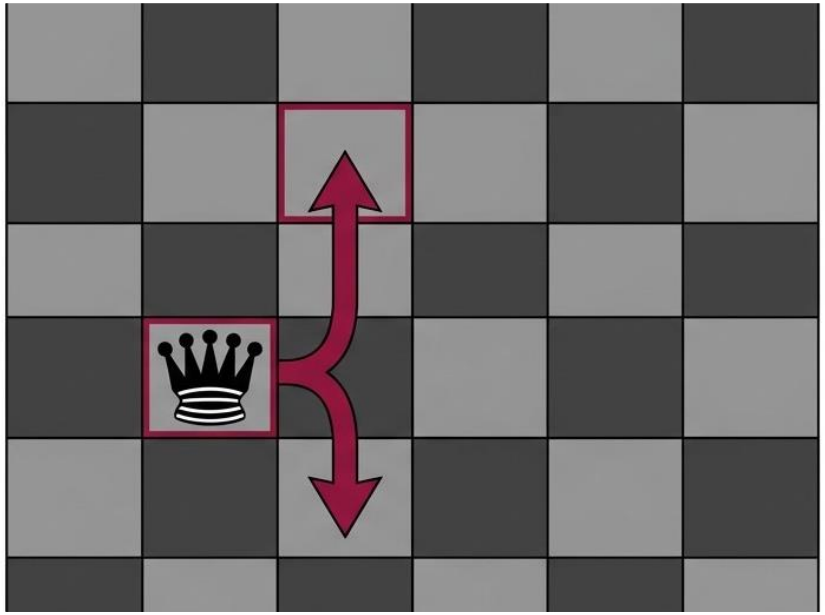
- Index = Column; Value = Row
- Benefits: compact; simple neighbor generation



mapping board state to vector

# Generating Neighbours

- Move one queen within its column
- Generates a neighboring state in search space



generating a neighboring state

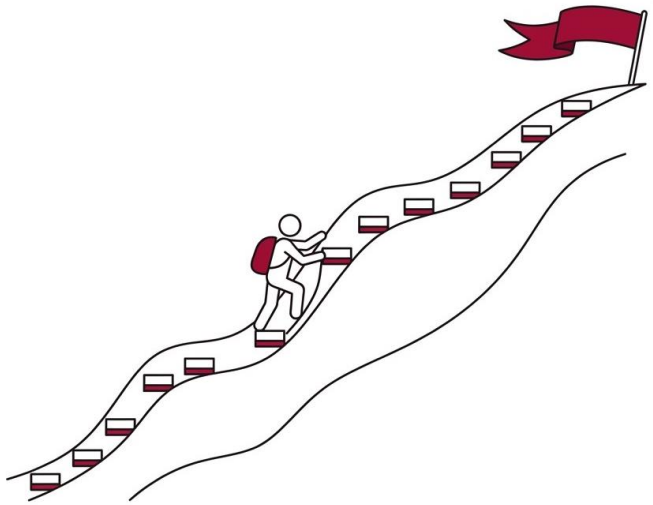
# How to Optimize the State?

- Imagine you are surrounded by many hills:
  - You cannot see the whole landscape
  - You only know the height around you

How can you reach the highest peak?

# Hill Climbing Algorithm

- Evaluate all neighboring states
- Choose the neighbor with the **best** score
- Move there immediately (Greedy)
- Repeat until no neighbor is better



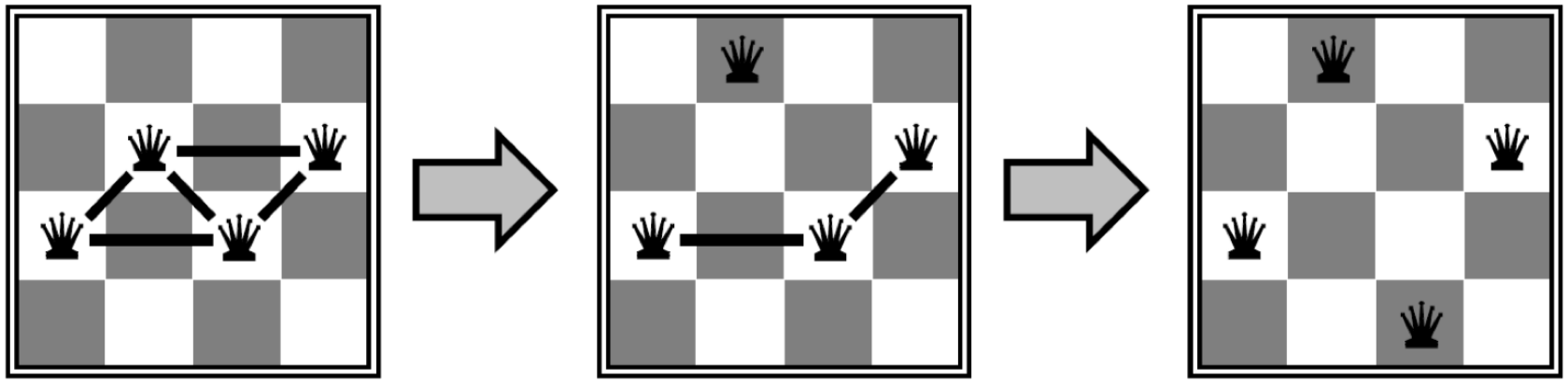
greedy improvement

# Hill Climbing

```
# Pseudo-algorithm
current ← initial state
loop
    neighbor ← best successor of current
    if neighbor is not better
    then
        stop
    current ← neighbor
```

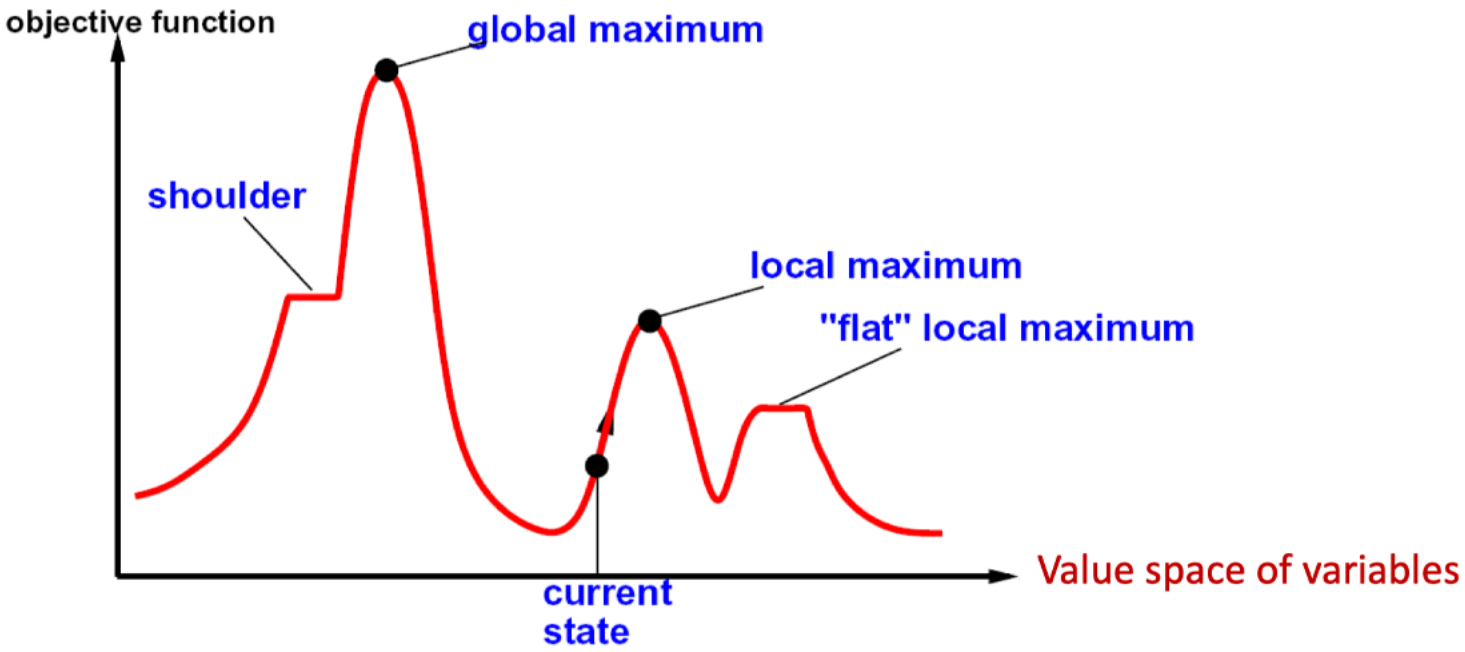
# N-Queens Search

- Start from a random initial board
- Evaluate moves with h(n) = number of attacks/conflicts
- Continue greedy improvement
- Goal test: no attacks



# Problems of Hill Climbing Algorithm

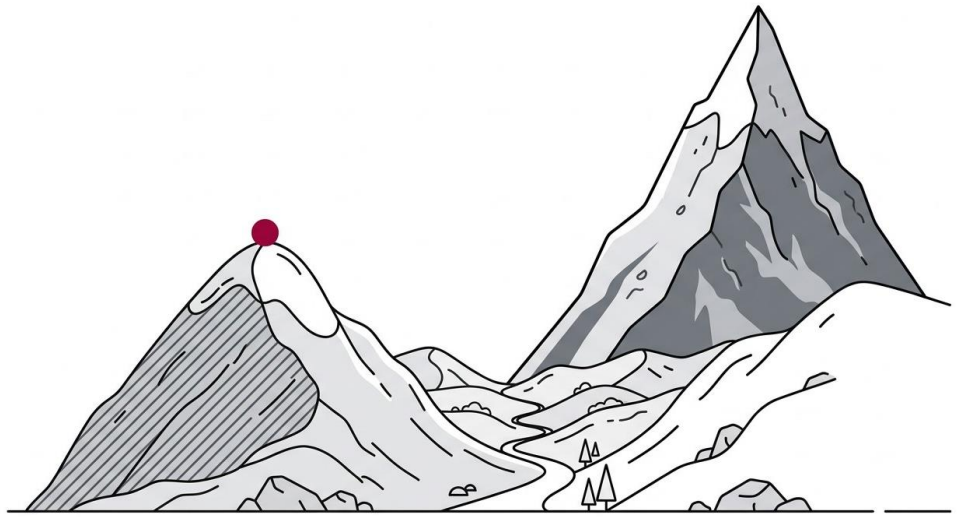
- Object landscape is not always ideal.



# Problem 1: Local Maximum

Hill climbing may stop too early:

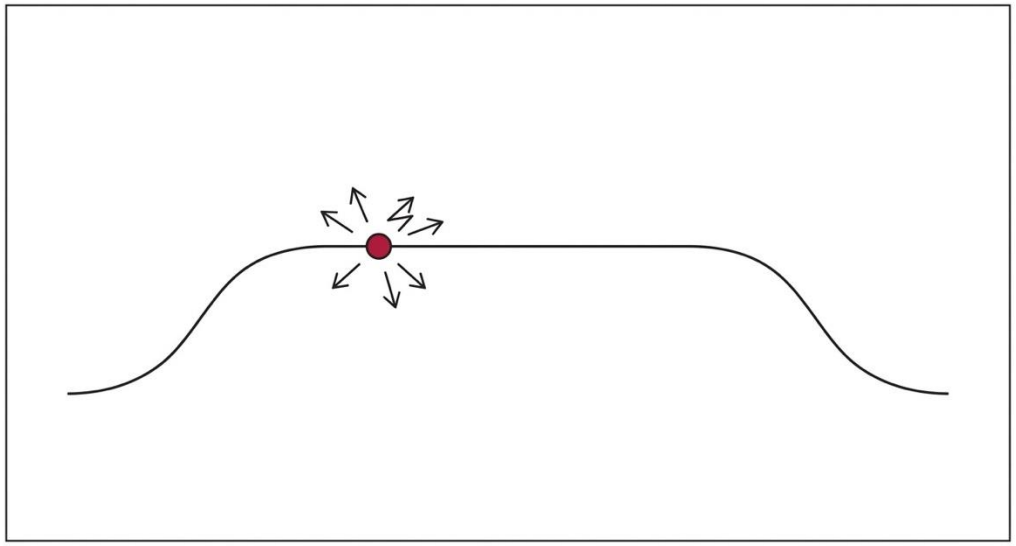
- Reached a peak better than all neighbors
- But worse than the global optimum
- No greedy step can escape the peak



getting stuck at a local optimum

# Problem 2: Plateau

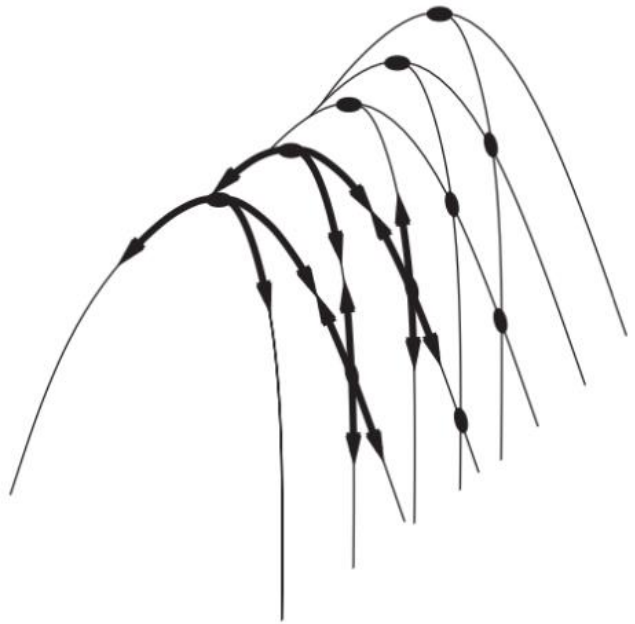
- Many neighboring states have the same score
- Landscape is locally flat
- Algorithm may wander randomly with no guidance



plateau region

# Problem 3: Ridge

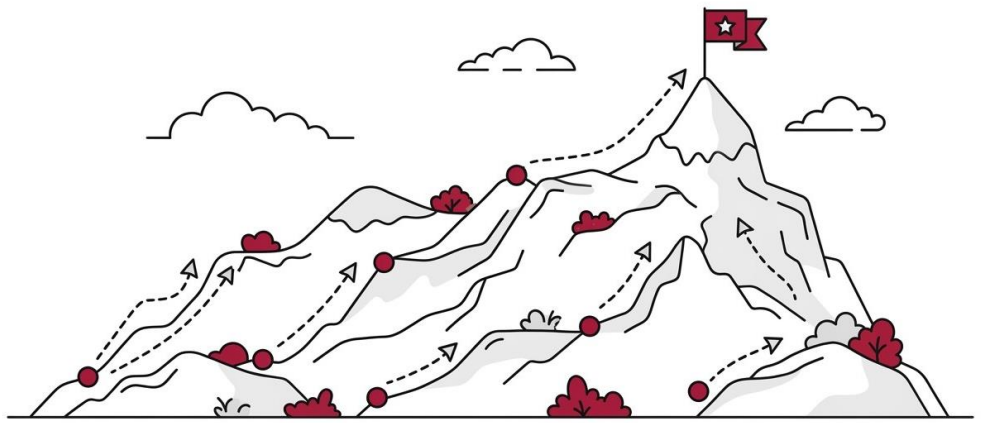
- The local slope does not point toward the optimum
- Improvement requires coordinate changes that aren't immediate
- Progress becomes extremely slow



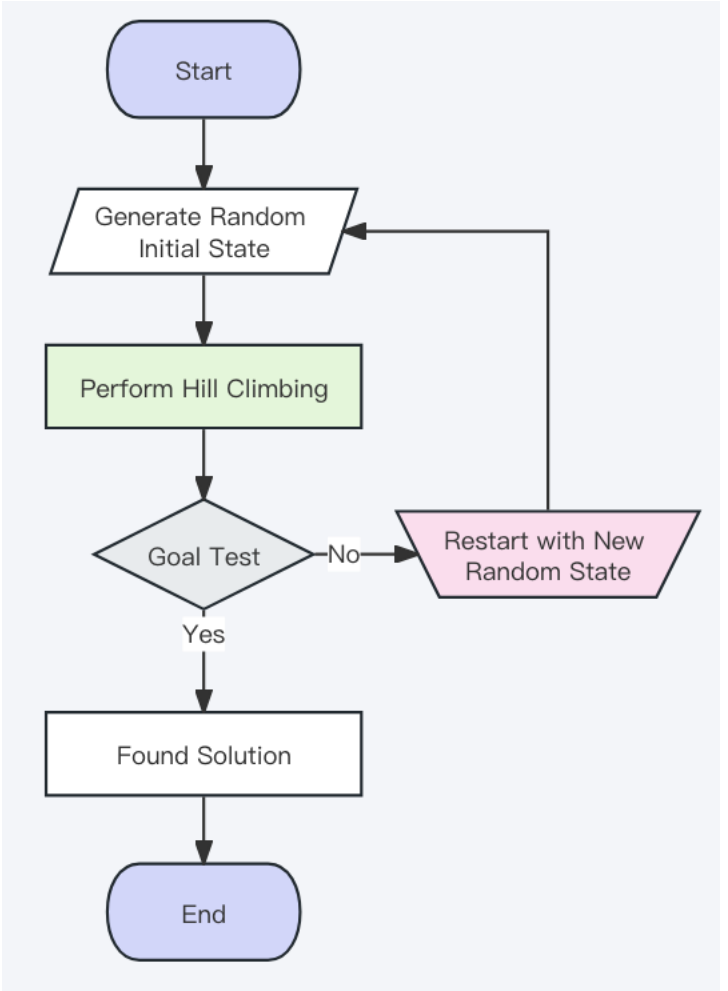
ridge problem

# Solution: Random Restart

- Run hill climbing multiple times
- Start each search from a **random state**
- Keep the best result found across all runs

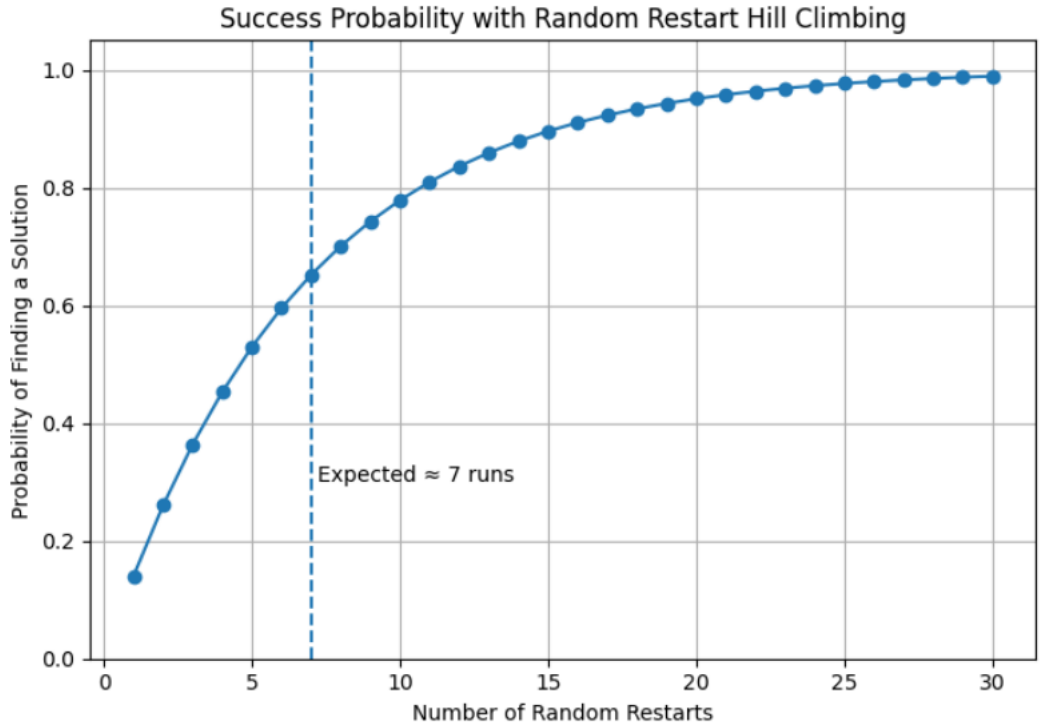


restarting search from different starting points



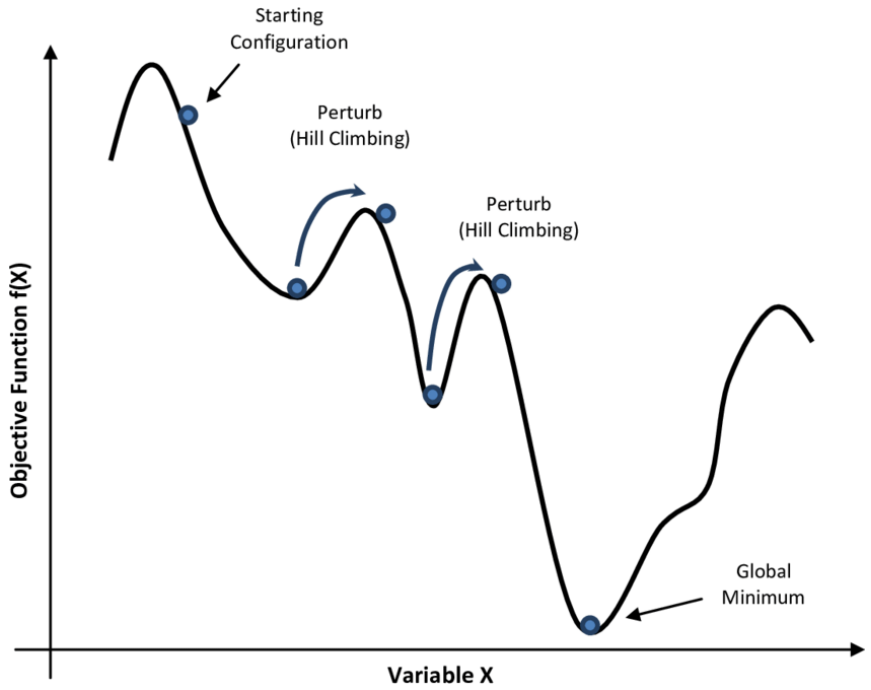
# Example: 8-Queens

- Success probability per run ≈ **0.14**
- Expected runs to find a solution ≈ **7**
- Random restart solves the problem very quickly



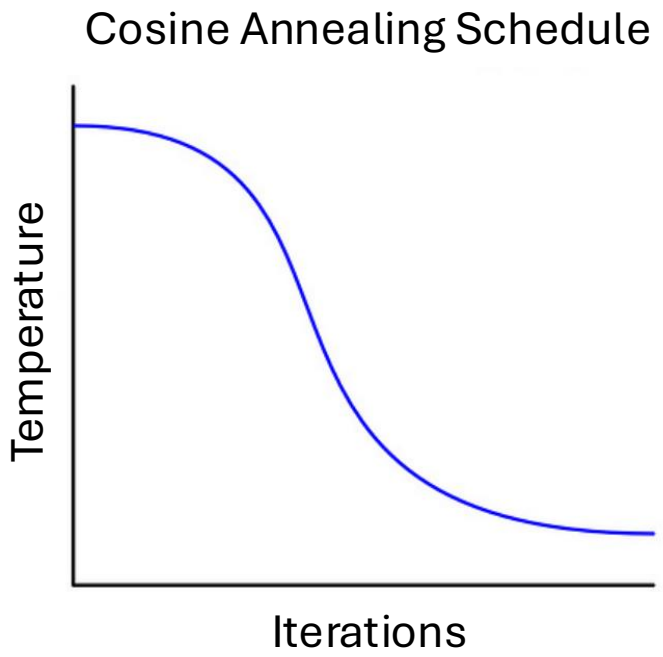
# Simulated Annealing

- **Idea:** sometimes accept moves that decrease score
- Allows escaping local suboptimal peaks
- Inspired by **annealing** (slow cooling) in physics
  - **High Temperature:** Accept worse moves frequently; explore the landscape widely
  - **Low Temperature:** Accept worse moves rarely; behave like greedy hill climbing
  - **Schedule:** Temperature decreases over time



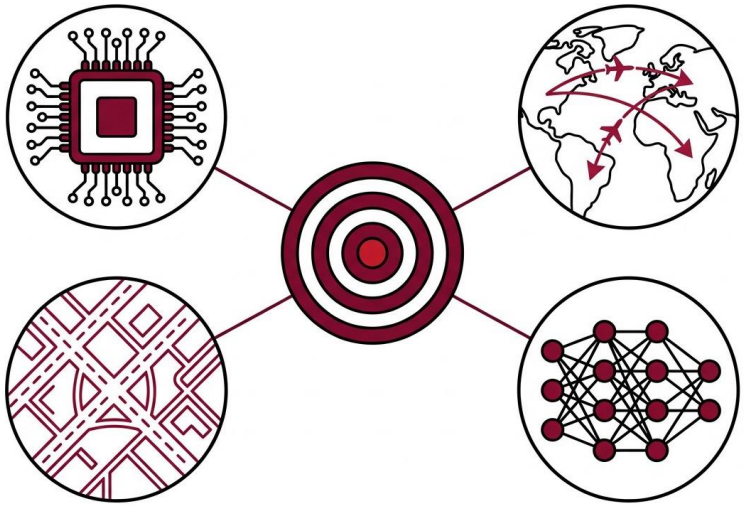
# Pseudo Code of Simulated Annealing

```
SIMULATED-ANNEALING(problem, schedule)
   current <= INITIAL-STATE(problem)
   for t = 1 to infinity do
      T <= schedule(t)
      if T = 0 then
         return current

      next <= random neighbor
      ΔE <= VALUE(next) − VALUE(current)

      if ΔE > 0 then
         current <= next
      else if random() < exp(ΔE / T) then
         current <= next
```



# Where Local Search Is Used

- Optimization in the real world:
  - Chip layout design
  - Airline crew scheduling
  - Route planning
  - Machine learning hyperparameter tuning



real-world applications

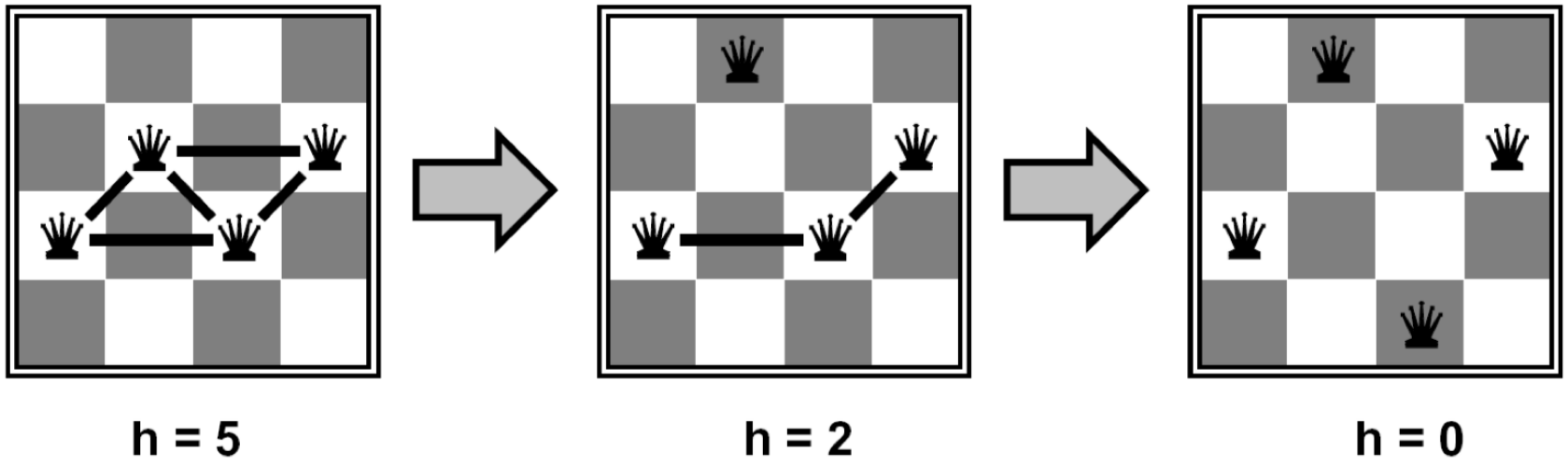
# 2-Minute Exercise

**Problem:** Solve **4-Queens** using hill climbing.

Initial state: [3 2 3 2]

- How many attacking pairs?
- Which neighbor improves the board?

*Take 1 minute to think…*



# Key Ideas

**Local Search summary:**

- Keeps one state (constant memory)
- Focuses on optimization (solution quality)
- Scales to astronomical search spaces

**Algorithms learned:**

- Hill climbing
- Random restart
- Simulated annealing

# Next Lecture

Adversarial Search