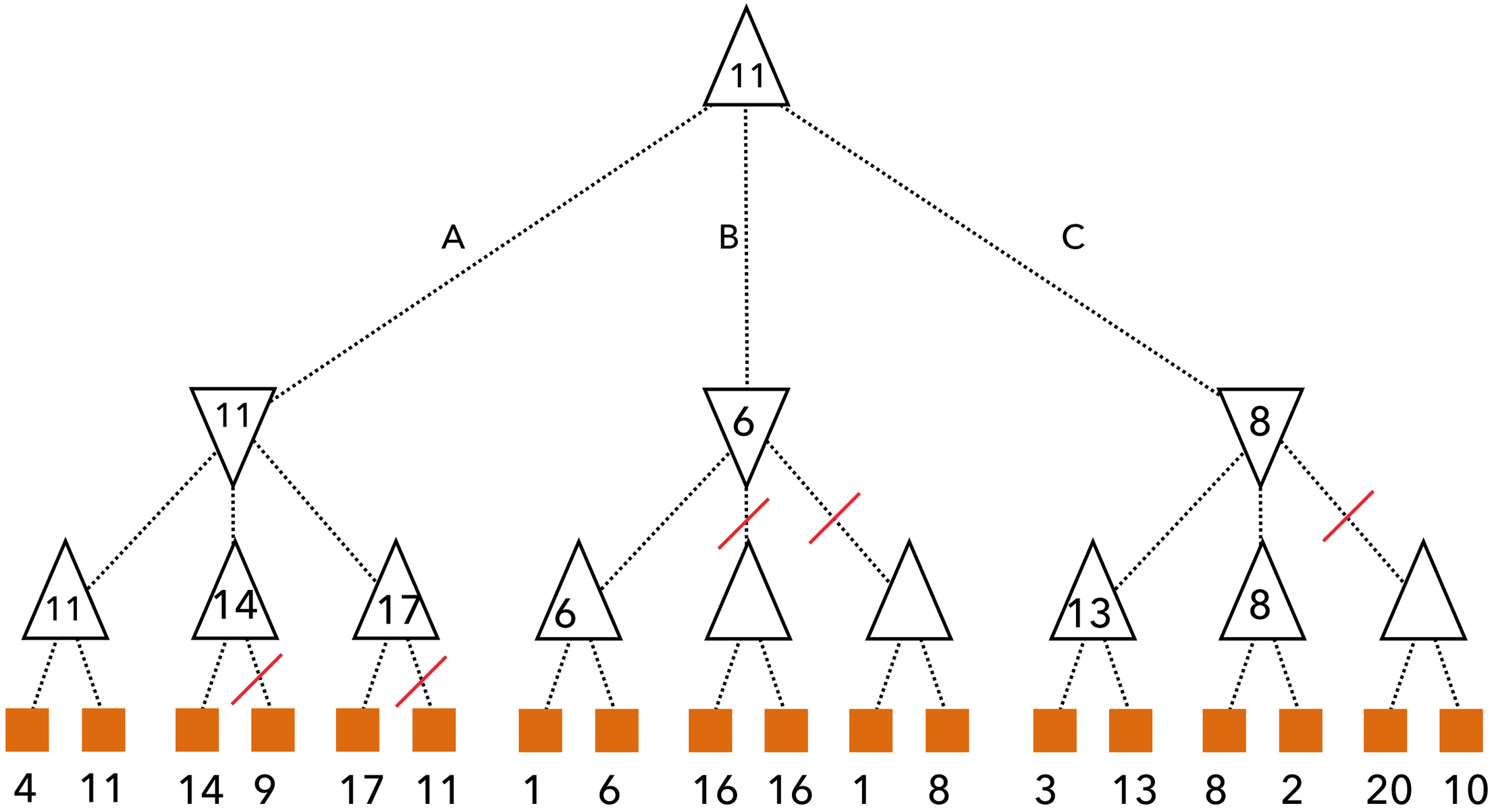
CSC 442: ARTIFICIAL INTELLIGENCE

LEC 07: LOCAL SEARCH

LAST WEEK

- Adversarial Search
  - Minimax
  - Alpha-Beta Pruning
- Expectations and Heuristics


11
A
B
C
11
6
8
11
14
17
6
13
8
4 11 14 9 17 11 1 6 16 16 1 8 3 13 8 2 20 10

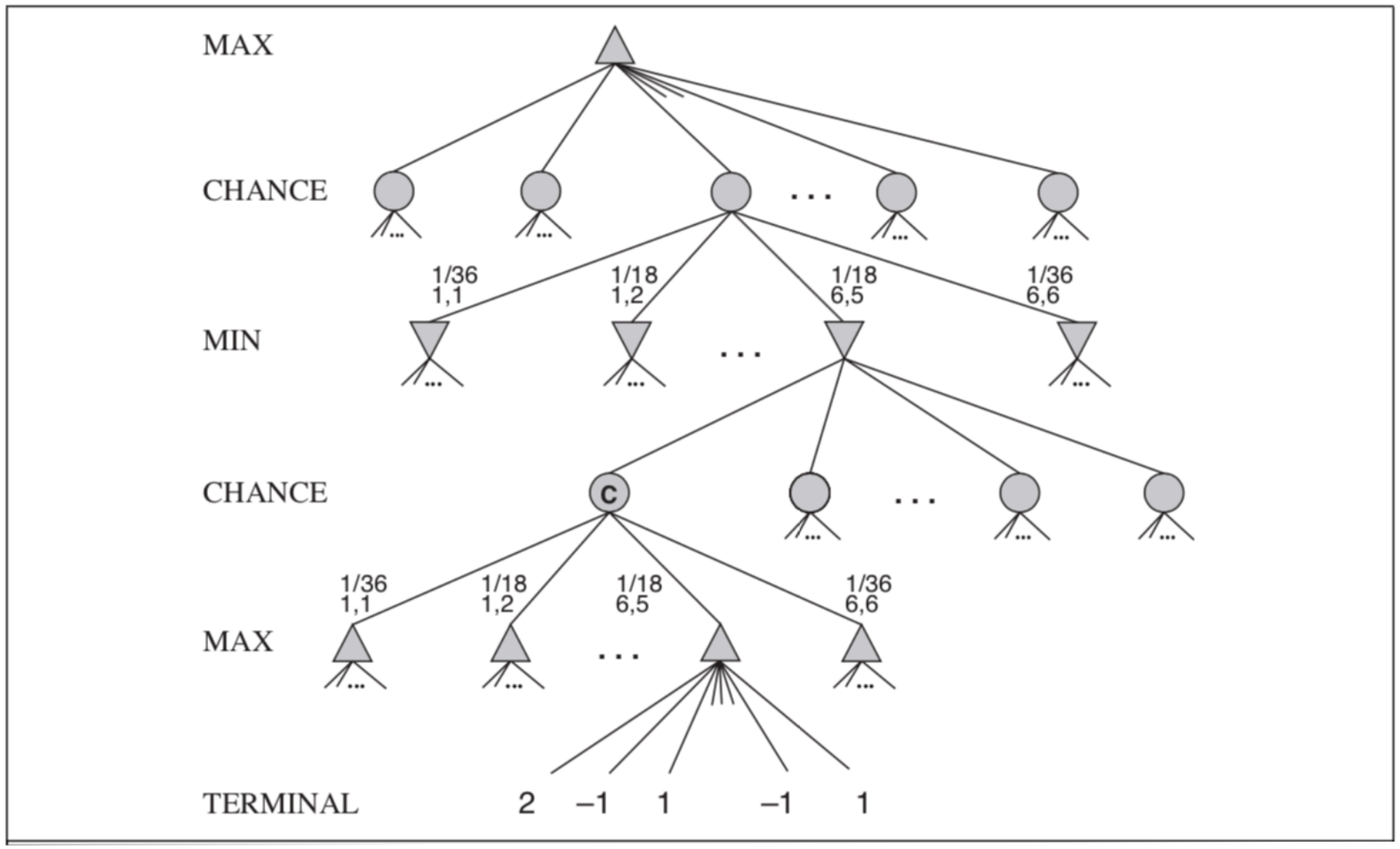
HEURISTIC MINIMAX

- Generalize terminal-test to **cutoff-test** and use **heuristic evaluation** after the cutoff, instead of searching all the way to terminals.

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MIN.} \end{cases}$$

AIMA, p171

EXPECTI-MINIMAX



AIMA, Fig 5.11, p178

QUIZ

- Online quiz *TODAY*.
- Please complete via blackboard before 11:59PM.

MOTIVATING INTRODUCTION

- hard problems; np-hardness; lack of tools; “good enough”; paths vs solutions

HARD PROBLEMS

- Traveling Salesman Problem
  - Find the shortest path which visits each of N cities.
- Satisfiability Testing
  - We'll discuss next week.
- Language Model Decoding
  - Find the "most likely" sentence uttered by a neural network in response to a prompt.

HARD PROBLEMS

- We have seen several search algorithms which are complete and optimal.
- This comes at a cost – computational complexity.
- Exponential complexity means many interesting problems are intractable.

HARD PROBLEMS

- **Local search algorithms** are algorithms which let go of optimality and completeness in exchange for **tractability.**

- For example:
  - You might not be able to find the BEST solution to a TSP in a given time limit, but you might be able to find one which is “good enough”!

LOCAL SEARCH ALGORITHMS

- We're going to look at four key algorithm families:
  - Hillclimbing
  - Simulated Annealing
  - Local Beam Search
  - Genetic Algorithms

HILLCLIMBING

- greedy algorithm
- random initial state
- repeat:
  always go uphill

Photo: The Karavanks, by Johann Jaritz, Creative Commons license.

HILL CLIMBING

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

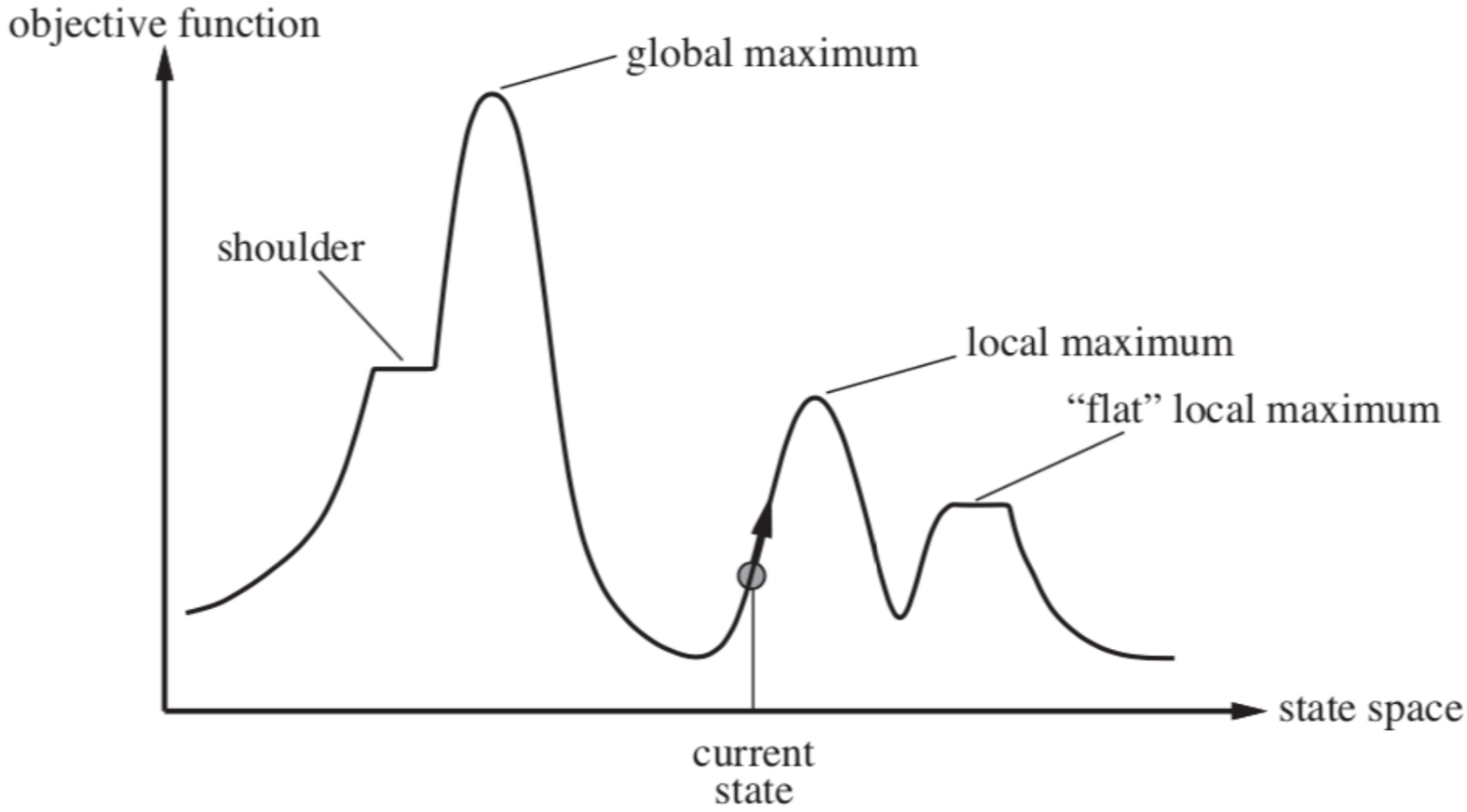
current ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
neighbor ← a highest-valued successor of *current*
if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
current ← *neighbor*

AIMA, Fig 4.2, p122

DIFFICULTIES FOR HILLCLIMBING

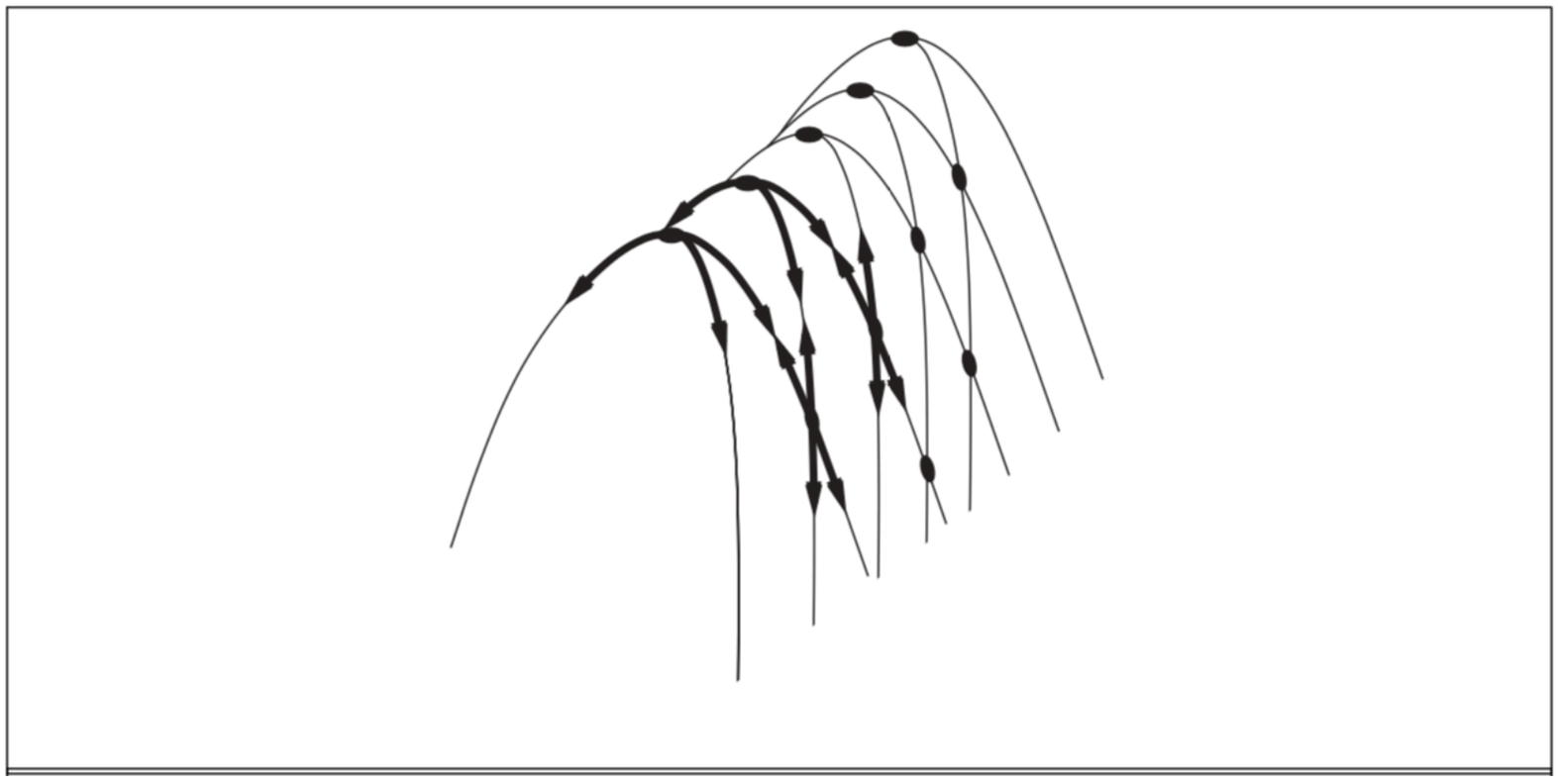
- **Local maxima** – a peak that is higher than each of its neighboring states but lower than the global maximum.
- **Ridges** – a narrow path up that is very difficult for greedy algorithms to navigate.
- **Plateaux** – a flat area of the state-space landscape.

A SAMPLE STATE-SPACE LANDSCAPE



AIMA, Fig 4.1, p121

RIDGES: A DIFFICULT HILL TO CLIMB



AIMA, Fig 4.4, p124

INFINITE PORCUPINES

- "Many real problems have a landscape that looks more like a widely scattered family of balding porcupines on a flat floor, with miniature porcupines living on the tip of each porcupine needle, ad infinitum."

Photo: Smithsonian National Zoo

INFINITE PORCUPINES

- "Many real problems have a landscape that looks more like a widely scattered family of balding porcupines on a flat floor, with miniature porcupines living on the tip of each porcupine needle, ad infinitum."

Photo: Smithsonian National Zoo

INFINITE PORCUPINES

- "Many real problems have a landscape that looks more like a widely scattered family of balding porcupines on a flat floor, with miniature porcupines living on the tip of each porcupine needle, ad infinitum."

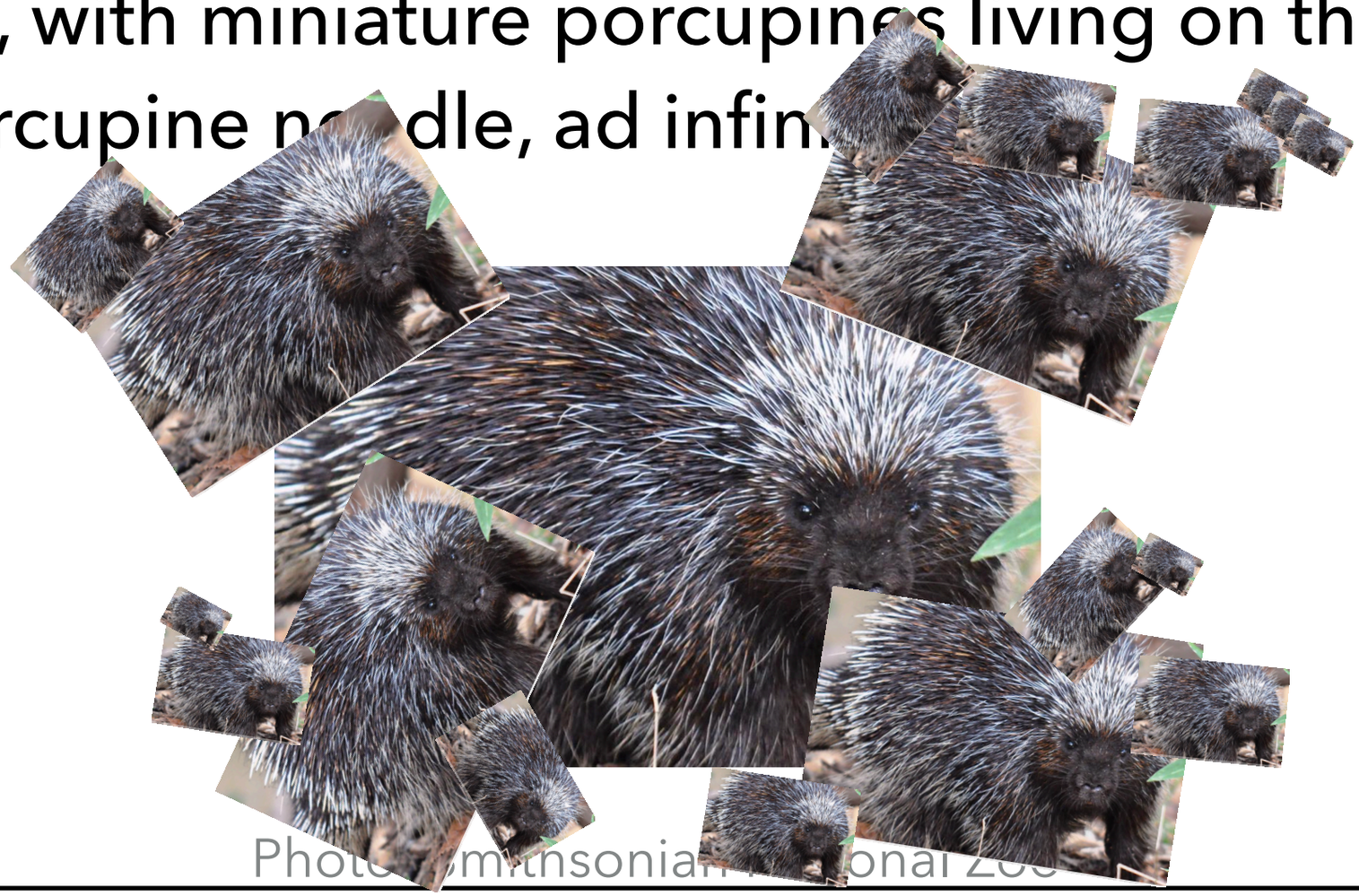
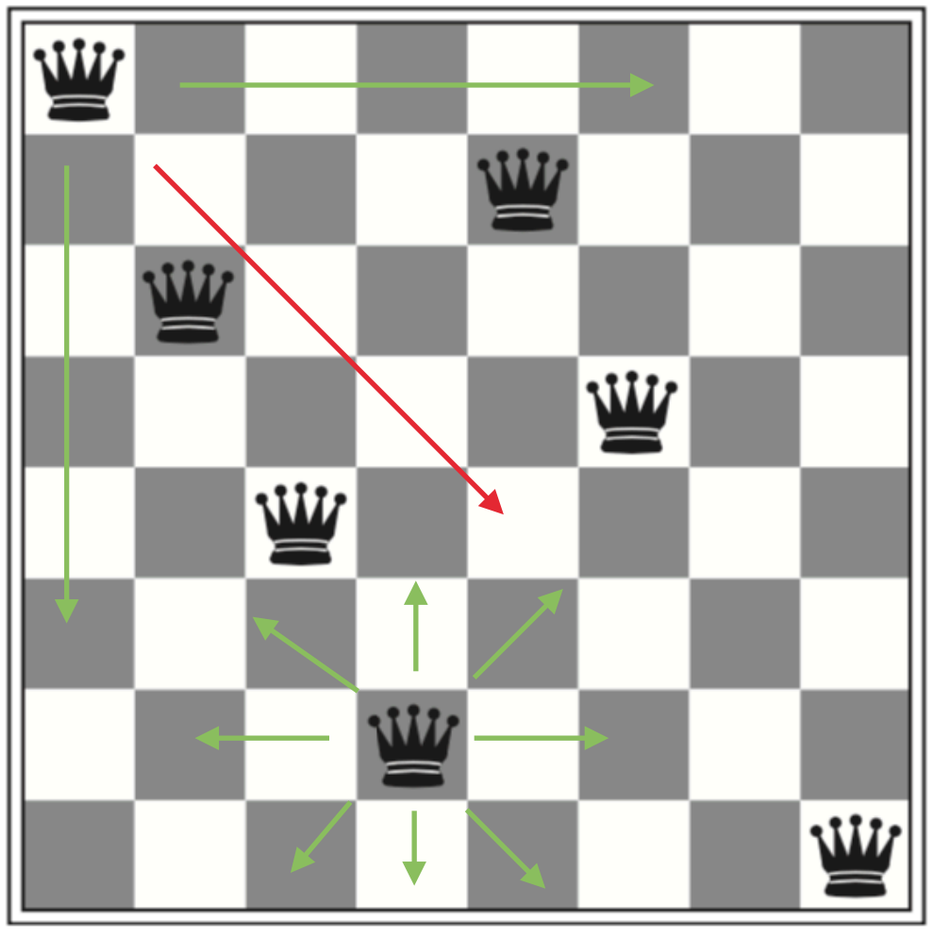


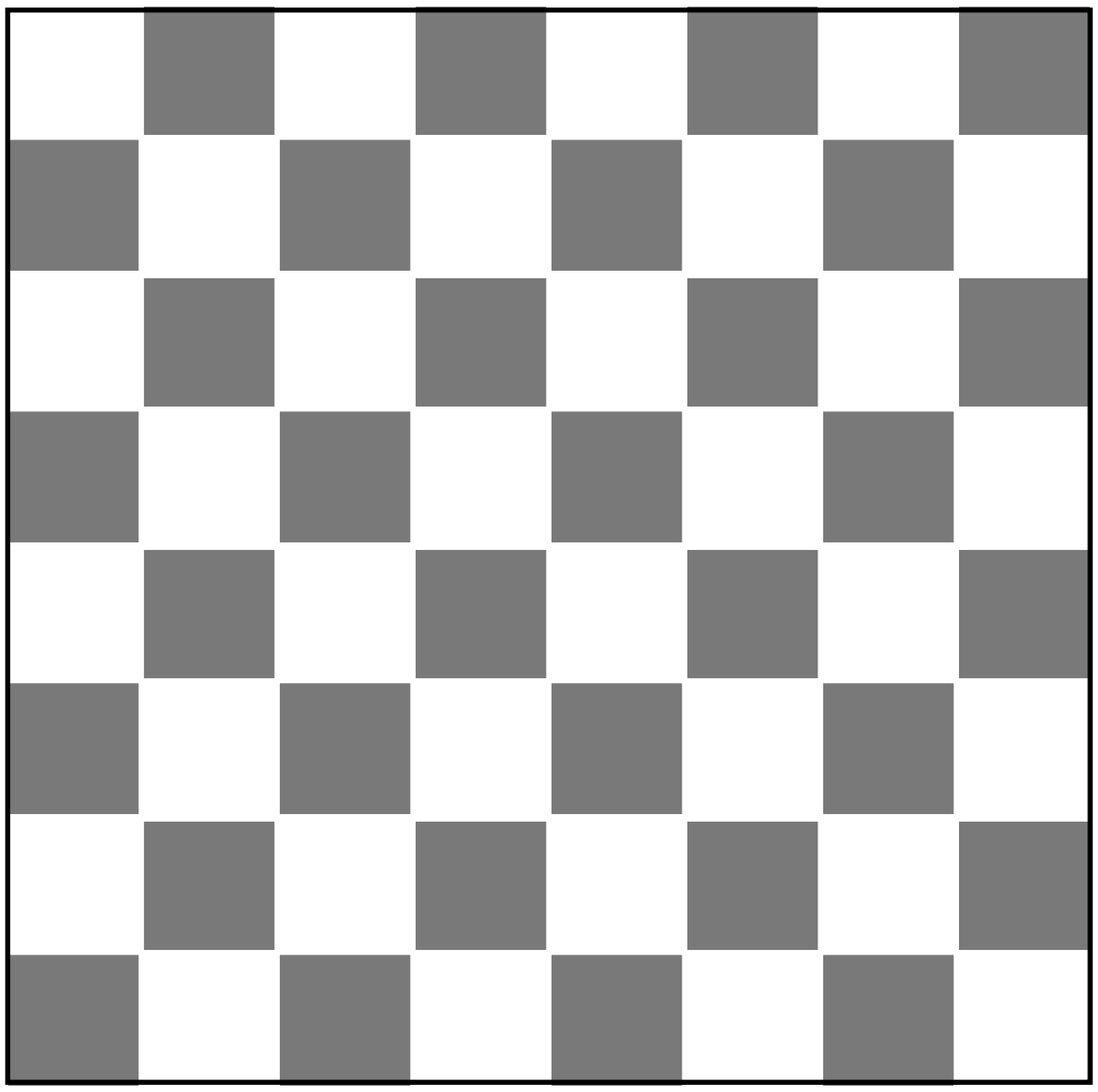
Photo: Smithsonian National Zoo

FAMILIAR PROBLEM: THE N-QUEENS PUZZLE

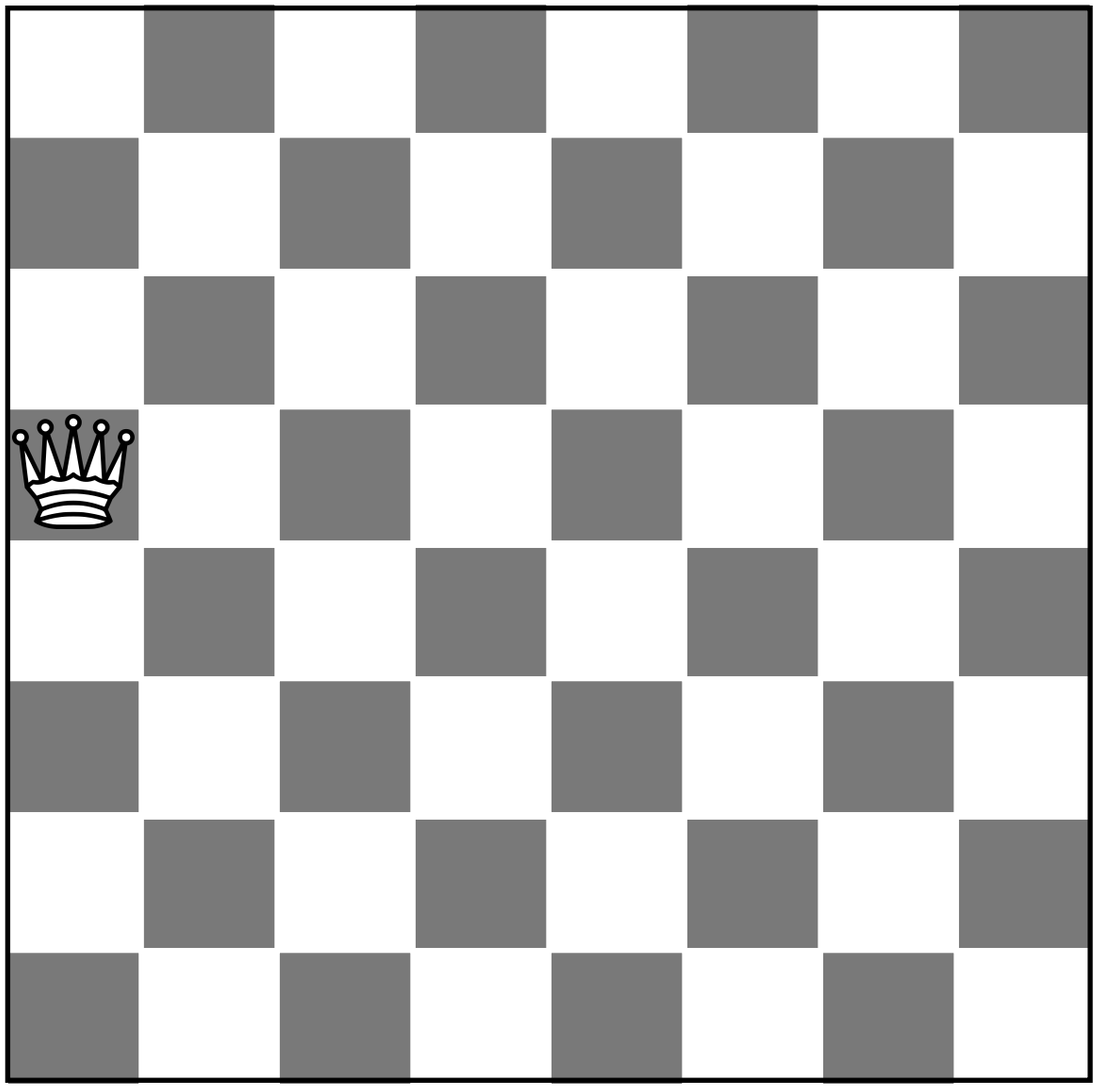
- Objective:
- Place N queens on an NxN chessboard, so that no two queens can attack each other.



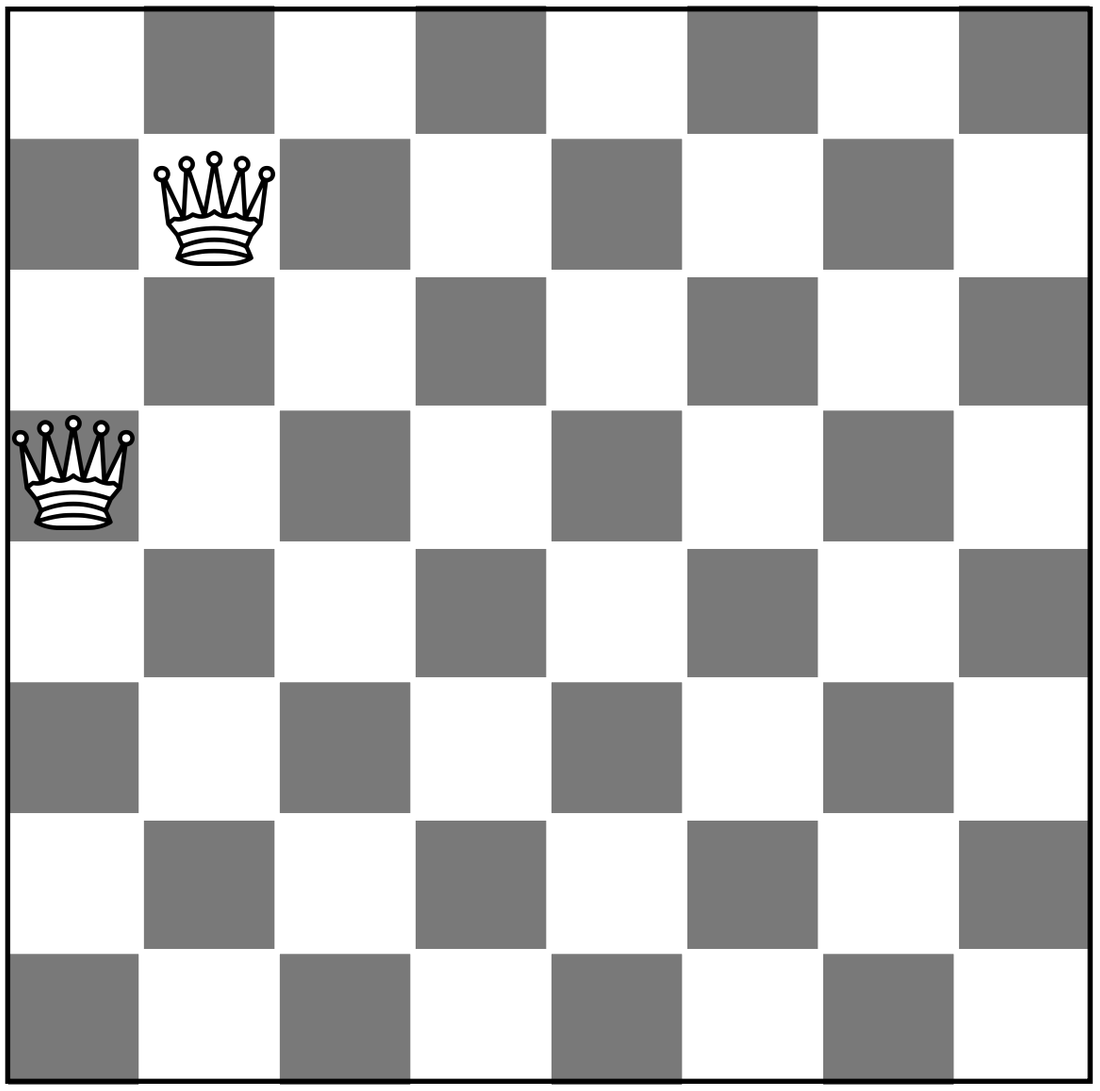
HILLCLIMBING FOR N-QUEENS



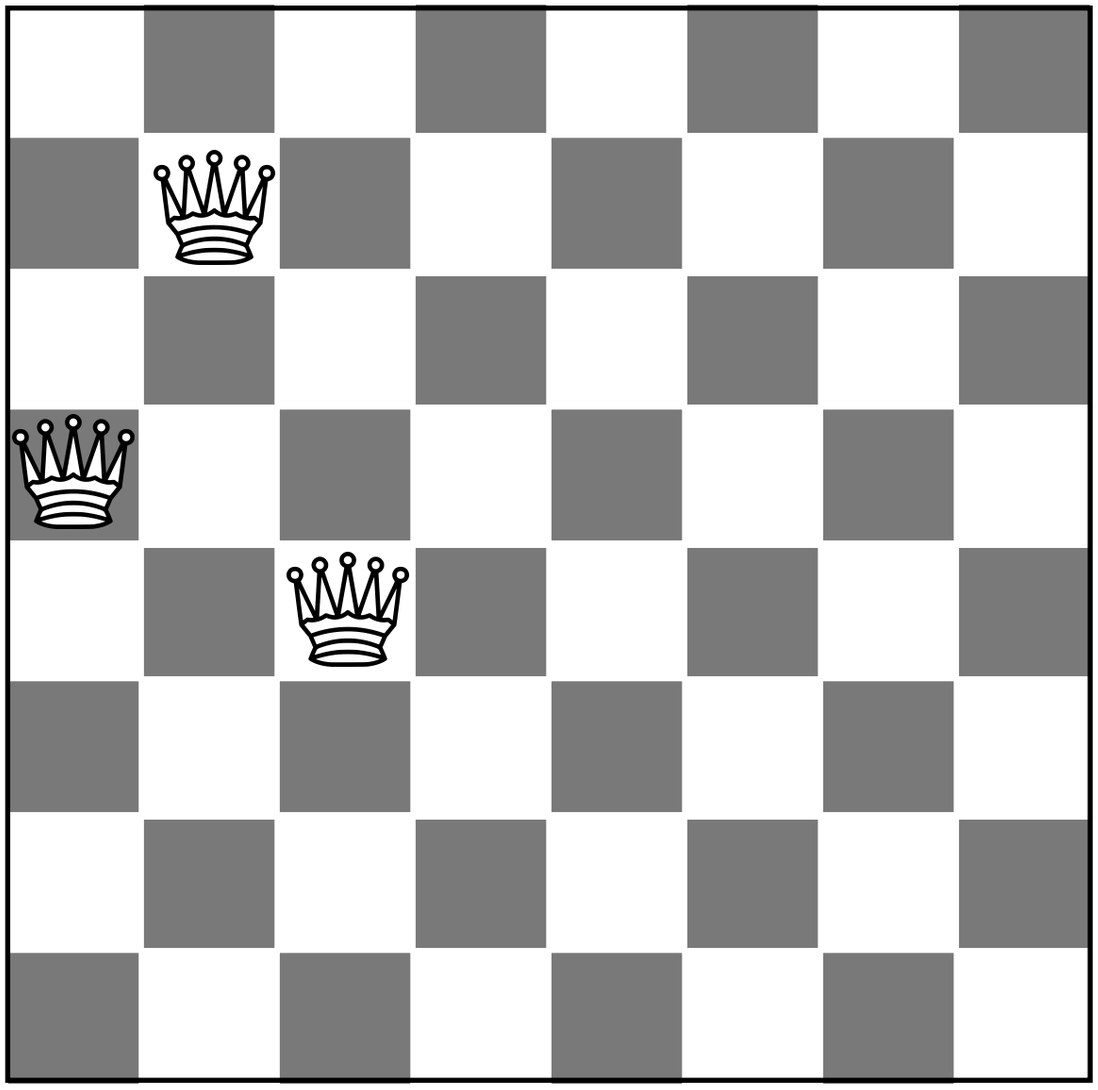
HILLCLIMBING FOR N-QUEENS



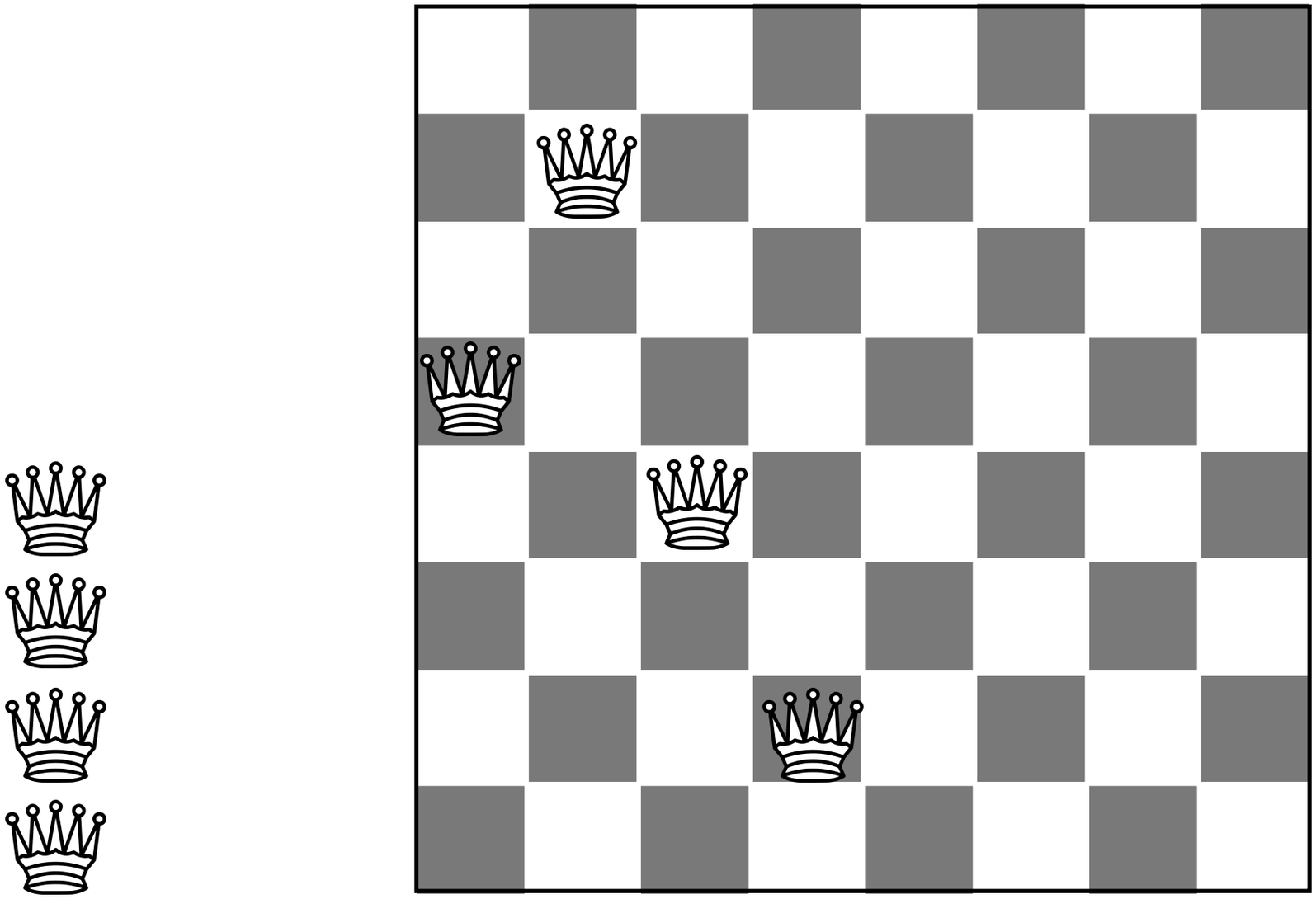
HILLCLIMBING FOR N-QUEENS



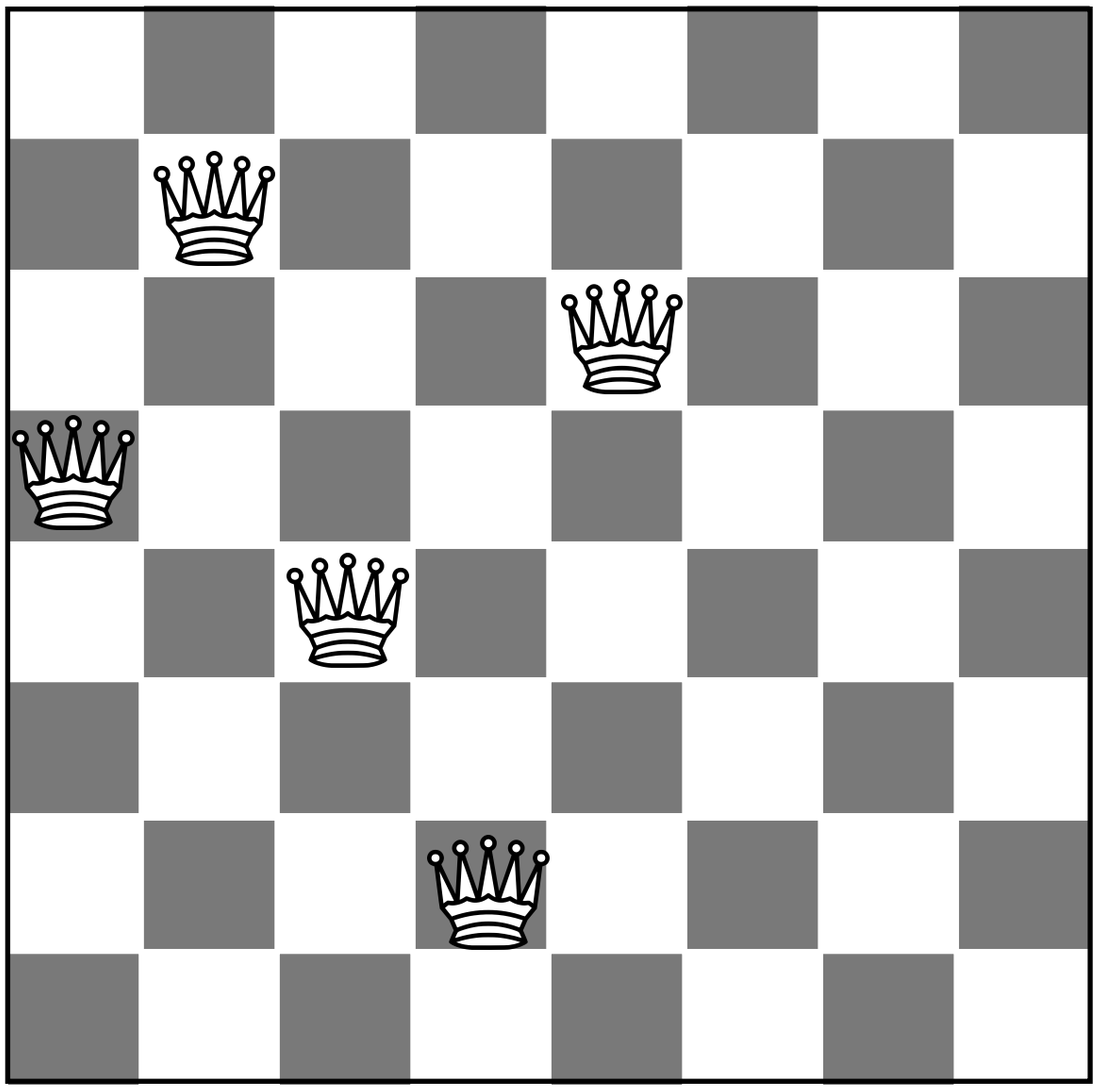
HILLCLIMBING FOR N-QUEENS



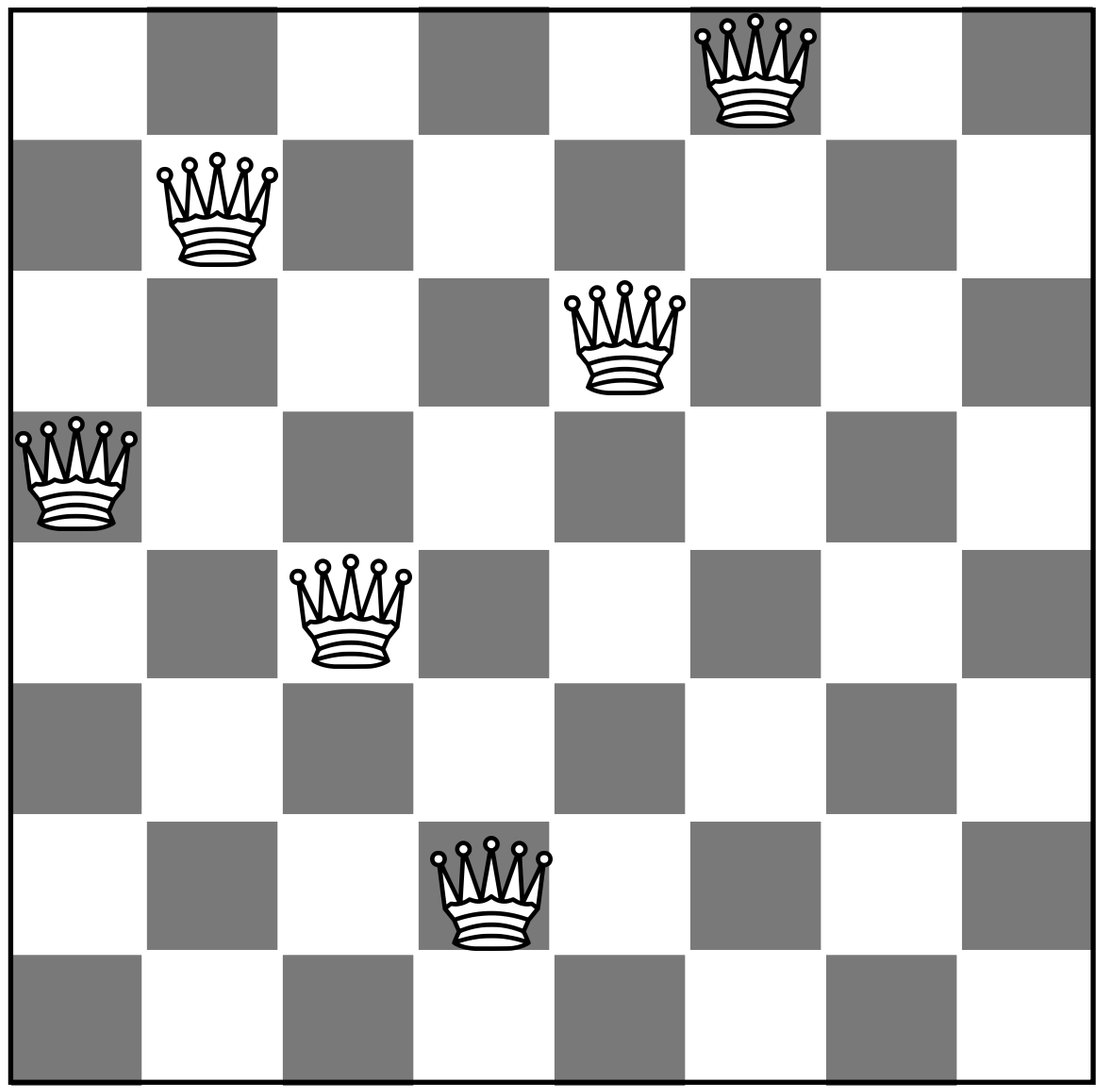
HILLCLIMBING FOR N-QUEENS



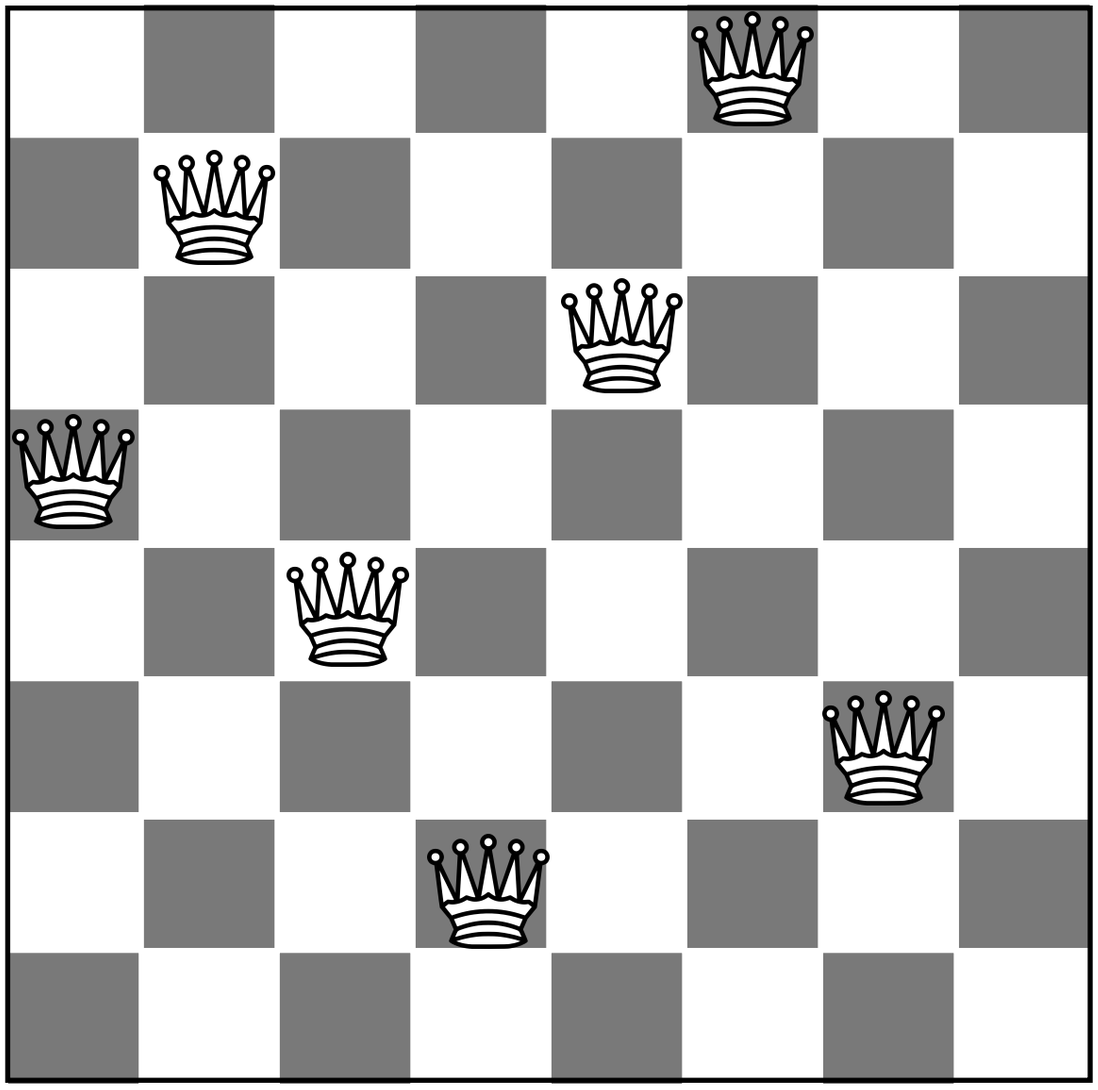
HILLCLIMBING FOR N-QUEENS



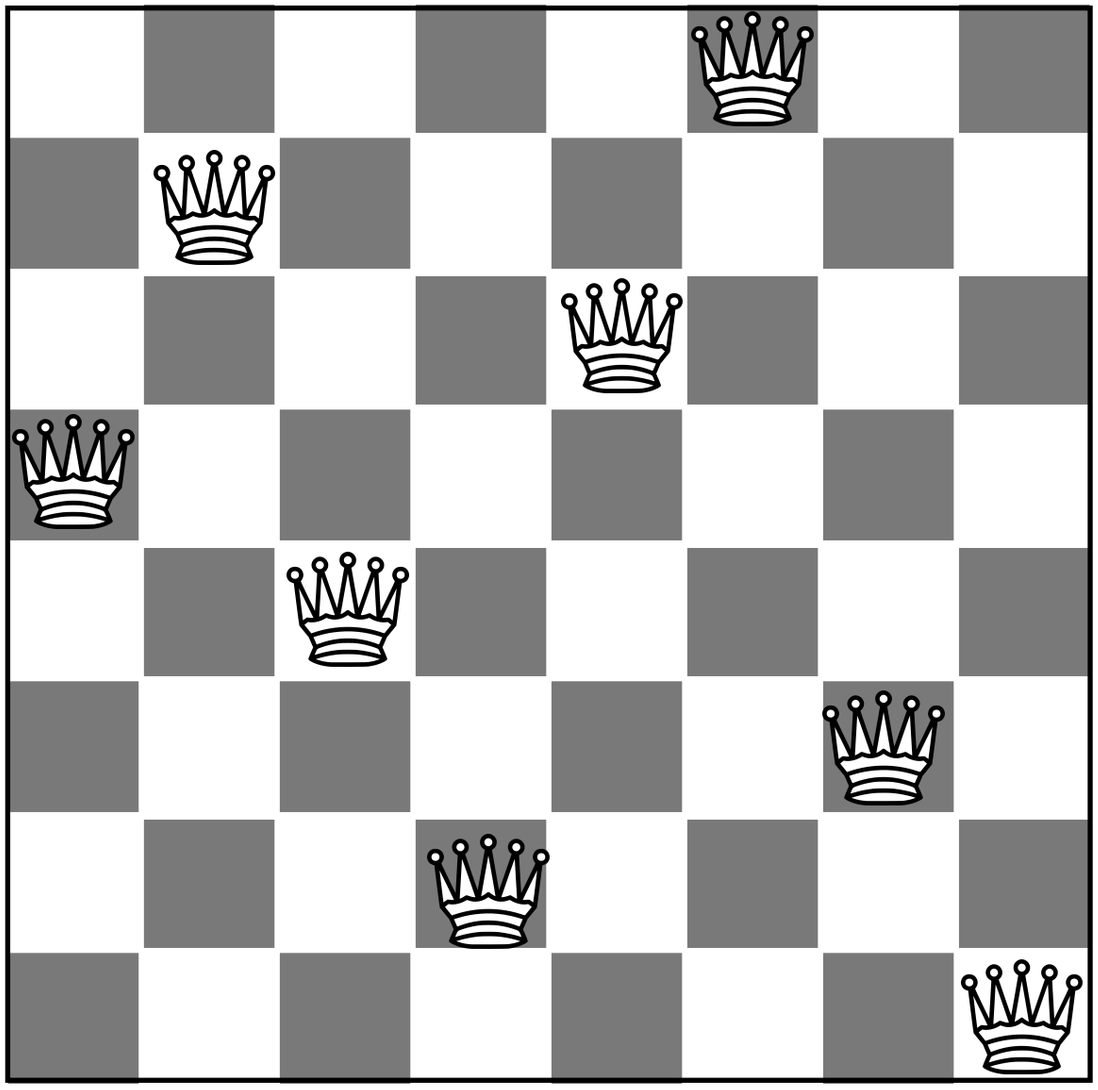
HILLCLIMBING FOR N-QUEENS



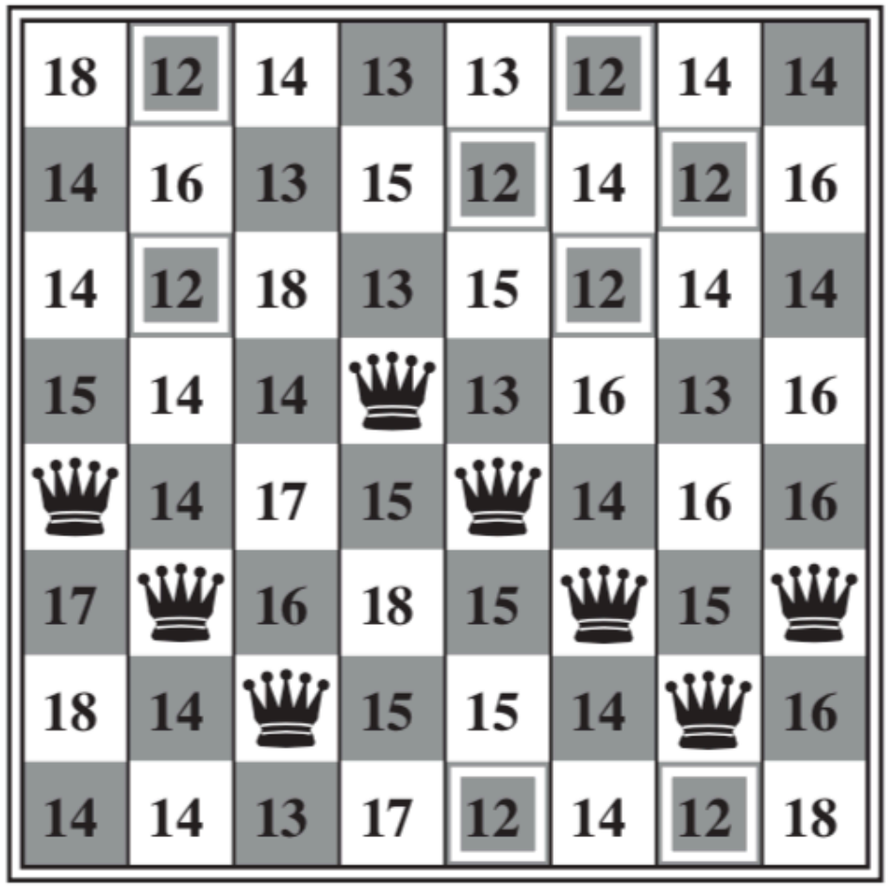
HILLCLIMBING FOR N-QUEENS



HILLCLIMBING FOR N-QUEENS

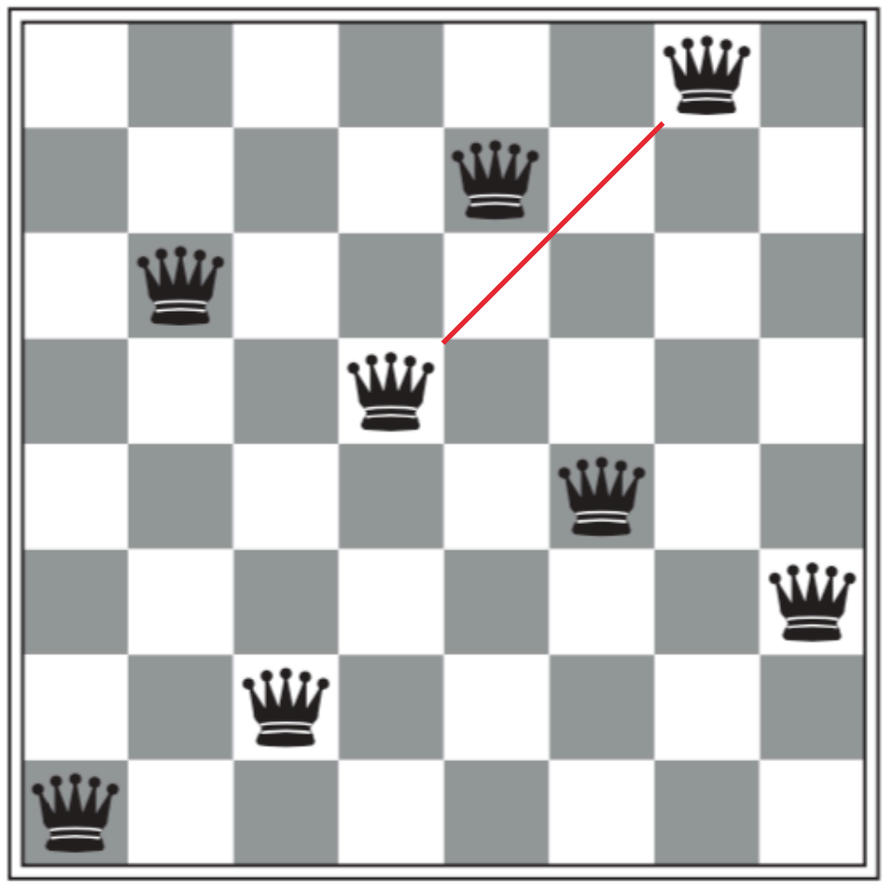


EXAMPLE: HILLCLIMBING FOR N-QUEENS



Current: h(n) = 17

Best successor: h(n′) = 12



Current: h(n) = 1

Best successor: h(n′) = 1

Local Optimum

AIMA, Fig 4.3, p123

NORVIG'S 8-QUEENS SIMULATION

- "Starting from a randomly generated 8-queens state, steepest-ascent hill climbing gets stuck 86% of the time, solving only 14% of problem instances."

AIMA, p123

NORVIG'S 8-QUEENS SIMULATION

- "It works quickly, taking just 4 steps on average when it succeeds and 3 when it gets stuck—not bad for a state space with 8^8 ≈ 17 million states."

AIMA, p123

HILLCLIMBING

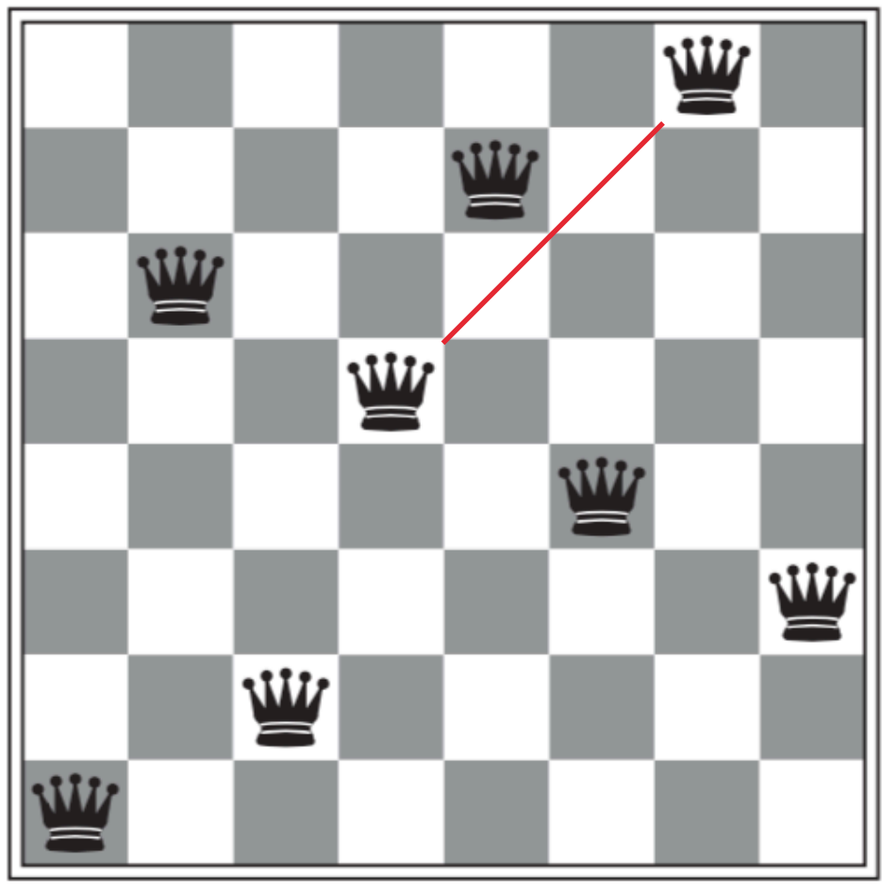
- So, hillclimbing gets stuck pretty often.

HILLCLIMBING: TIPS AND TRICKS

- Two very powerful tricks to improve hillclimbing:
  - Sideways Moves
  - Random Restart

HILLCLIMBING — SIDEWAYS MOVES

- At the first local optimum, basic hillclimbing gets stuck.
- Sideways moves are where the score is equivalent, but the state is different.



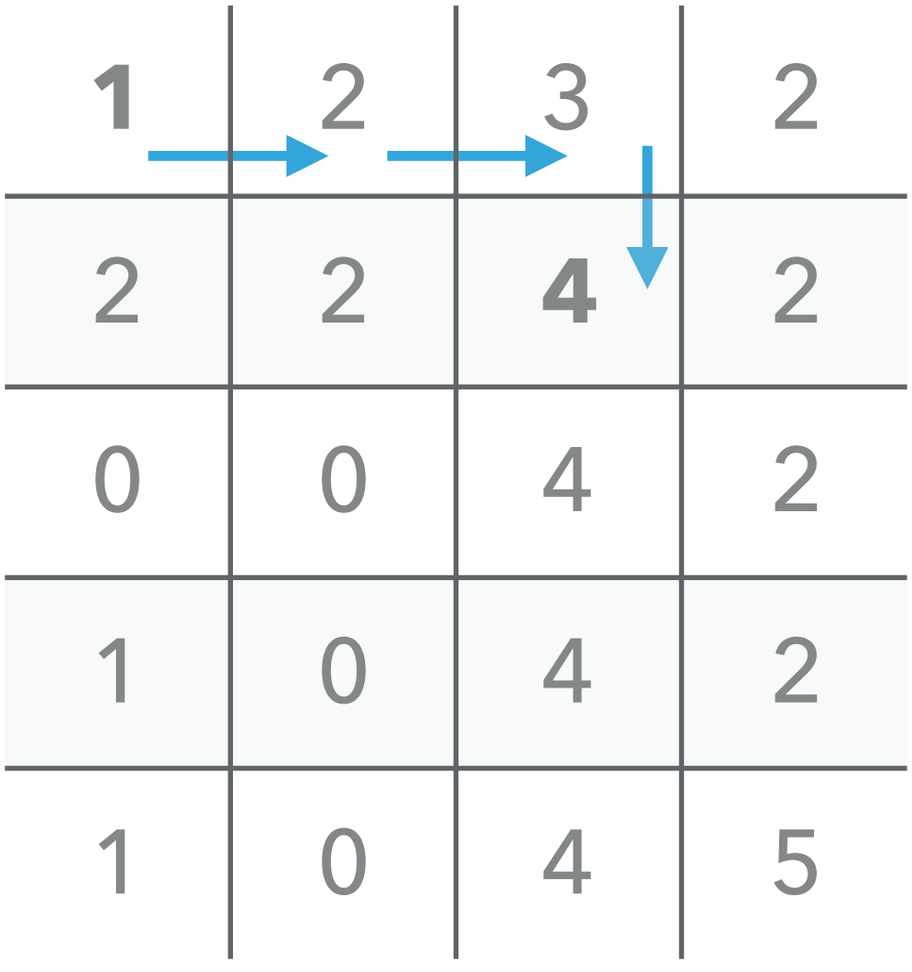
Current: h(n) = 1

Best successor: h(n') = 1

Local Optimum

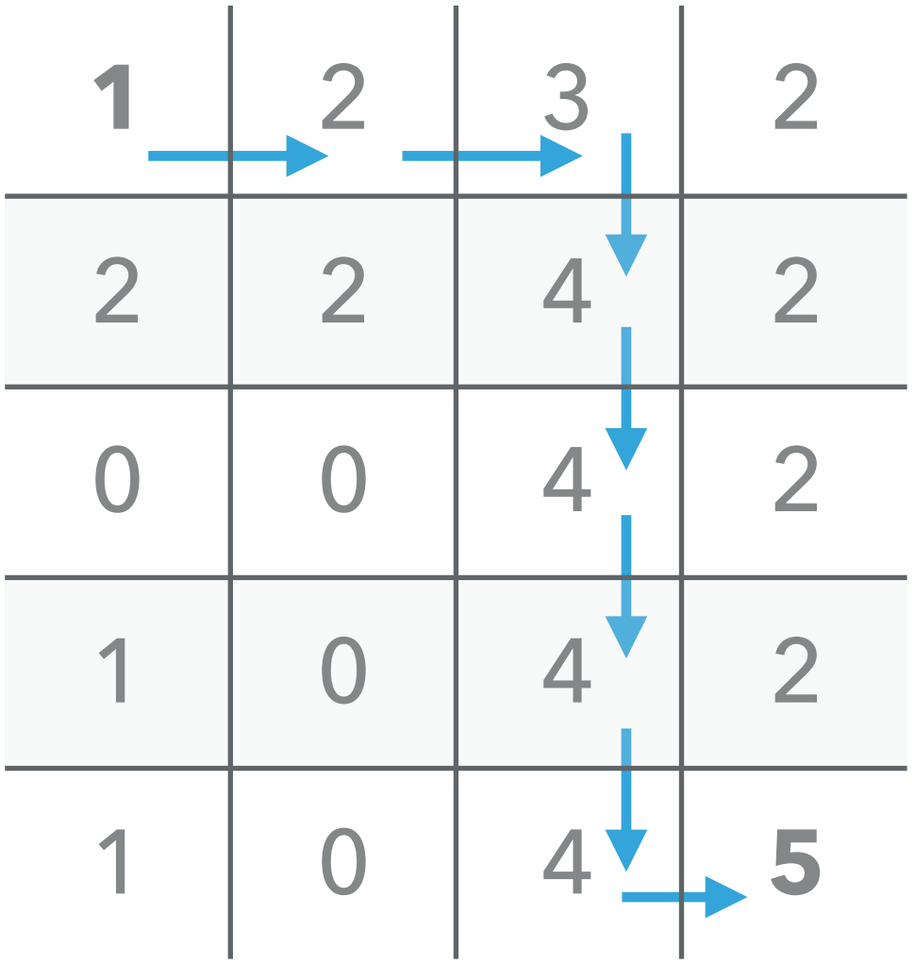
HILLCLIMBING

- "Basic" algorithm:
  - random init to upper left
  - stops at first local optimum – 4



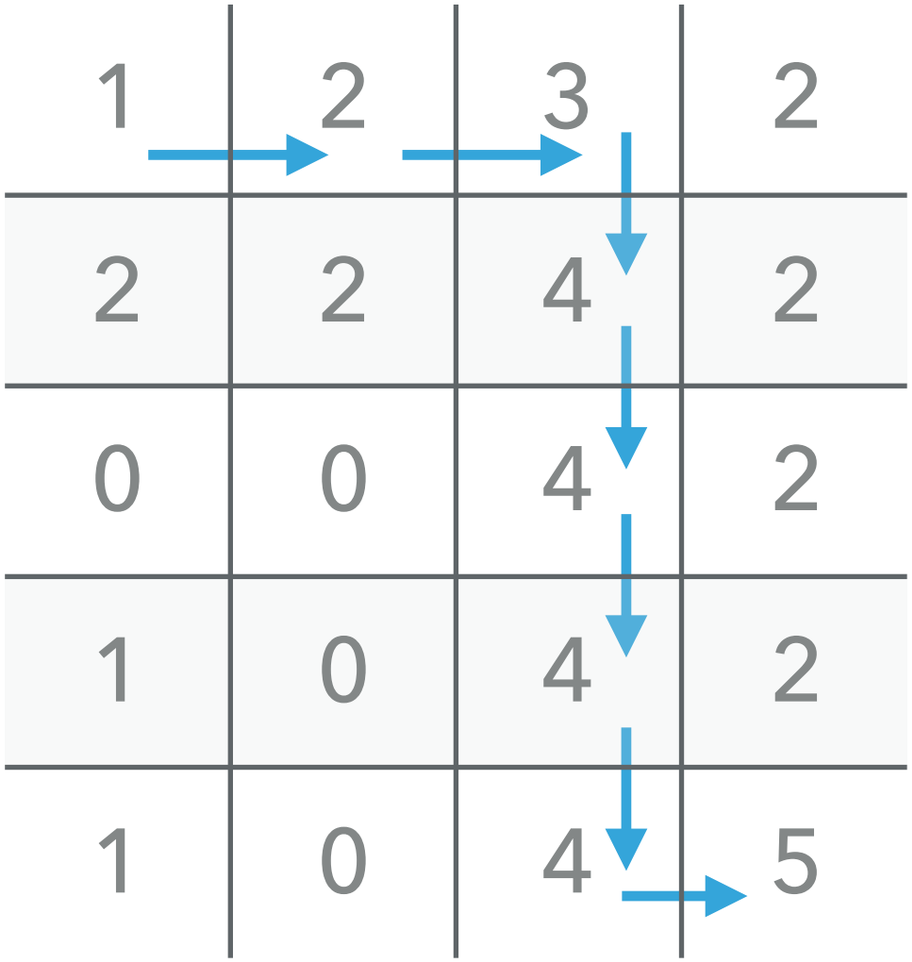
HILLCLIMBING: SIDEWAYS MOVES

- Sideways moves let us explore the equivalent valued states.
- Reach new optimum: 5



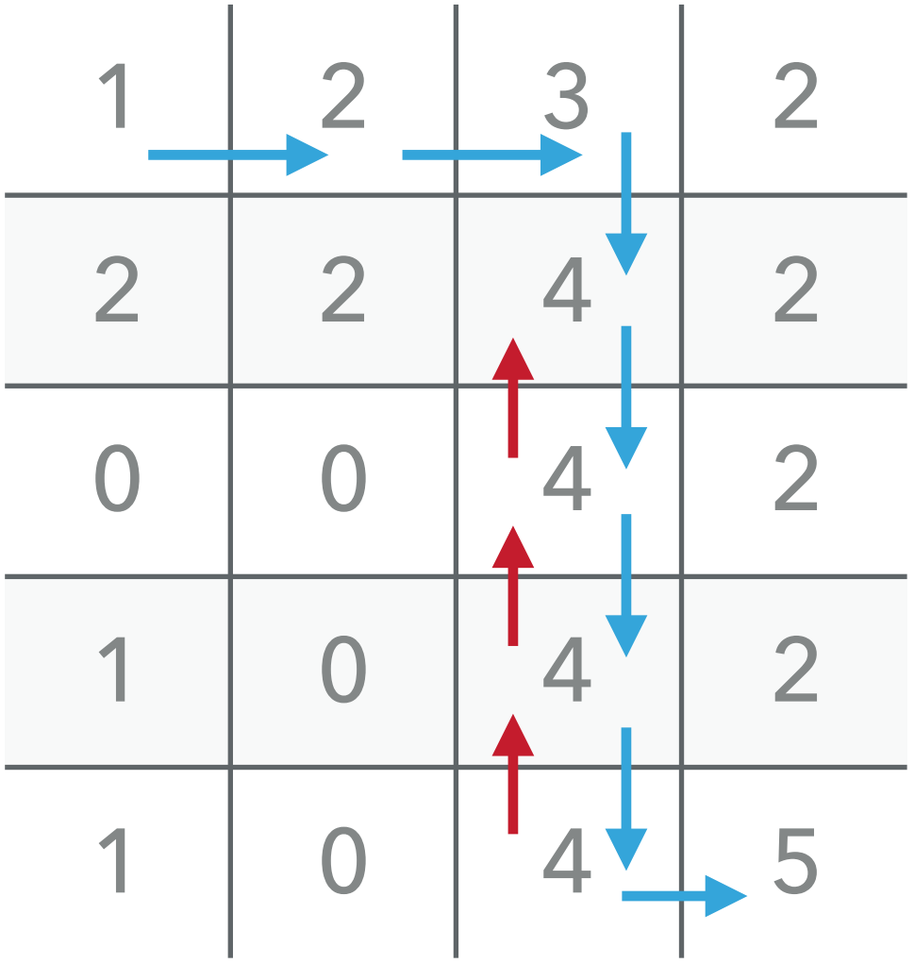
HILLCLIMBING: SIDEWAYS MOVES

- What if you are unlucky?



HILLCLIMBING: SIDEWAYS MOVES

- What if you are unlucky?
- Sideways moves can wander around aimlessly…



HILLCLIMBING: SIDEWAYS MOVES

- What if you are **super** unlucky?

1	2	3	2
2	2	4	2
0	0	4	2
1	0	4	2
1	0	4	5

HILLCLIMBING: SIDEWAYS MOVES

- What if you are **super** unlucky?

1	2	3	2
2	2	4	2
0	0	4	2
1	0	4	2
1	0	4	5

AIMA'S 8-QUEENS SIMULATION

- Basic hillclimbing – successful in 14% of initializations.
  - Avg success in 4 steps. Avg stuck in 3 steps.
- Allowing 100 consecutive sideways pushes it to 94%.
  - Avg success now 21 steps.
  - Avg failure now 64.

- The hill-climbing algorithms described so far are incomplete—they often fail to find a goal when one exists because they can get stuck on local maxima (if we get super unlucky).

If at first you don't succeed, try, try again.

HILLCLIMBING: RANDOM RESTART

- Random restart – If you get stuck, restart from a new random initialization.
- Trivially complete.
- Expected number of restarts is 1/Pr(success).

- For 8-queens instances with no sideways moves allowed, p ≈ 0.14, so we need roughly 7 iterations to find a goal (6 failures and 1 success).

- The expected number of steps is the cost of one successful iteration plus (1−p)/p times the cost of failure, or roughly 22 steps in all.

- When we allow sideways moves, 1/0.94 ≈ 1.06 iterations are needed on average and (1 × 21) + (0.06/0.94) × 64 ≈ 25 steps.

HILLCLIMBING: RANDOM RESTART

- Without sideways moves:
  - E(# steps to solve) = 22.42
- With up to 100 sideways move:
  - E(# steps to solve) = 24.78
- But 24.78 > 22.42!
- Sometimes it is better to restart often, other times it might be better to allow more sideways moves. The true behavior depends on your problem space!

HILLCLIMBING: SIDEWAYS MOVES

- What if you are **super** unlucky?

1	2	3	2
2	2	4	2
0	0	4	2
1	0	4	2
1	0	4	5

- For 8-queens instances with no sideways moves allowed, p ≈ 0.14, so we need roughly 7 iterations to find a goal (6 failures and 1 success).

- The expected number of steps is the cost of one successful iteration plus (1−p)/p times the cost of failure, or roughly 22 steps in all.

- When we allow sideways moves, 1/0.94 ≈ 1.06 iterations are needed on average and (1 × 21) + (0.06/0.94) × 64 ≈ 25 steps.

- For 8-queens instances with no sideways moves allowed, p ≈ 0.14, so we need roughly 7 iterations to find a goal (6 failures and 1 success).

- The expected number of steps is the cost of one successful iteration plus (1−p)/p times the cost of failure, or roughly 22 steps in all.

- When we allow sideways moves, 1/0.94 ≈ 1.06 iterations are needed on average and (1 × 21) + (0.06/0.94) × 64 ≈ 25 steps.

SEGUE — FROM HILLCLIMBING TO ANNEALING

- A hill-climbing algorithm that never makes "downhill" moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum.
- In contrast, a purely random walk—that is, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient.
- Therefore, it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness. **Simulated annealing** is such an algorithm.

SIMULATED ANNEALING

Image from Wikimedia Commons. This file was provided to Wikimedia Commons by the Deutsche Fotothek of the Saxon State Library / State and University Library Dresden (SLUB)

- In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state.

SIMULATED ANNEALING

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

AIMA, Fig 4.5, p126

SIMULATED ANNEALING (WALL OF TEXT)

- The innermost loop of the simulated-annealing algorithm (Figure 4.5) is quite similar to hill climbing. Instead of picking the best move, however, it picks a **random move**.
- If the move improves the situation, it is always accepted. Otherwise, the algorithm accepts the move with some probability less than 1. The probability decreases exponentially with the "badness" of the move—the amount ΔE by which the evaluation is worsened.
- The probability also decreases as the "temperature" T goes down: "bad" moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases. If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1.
- Original citation: Kirkpatrick et al 1983.

BEAM SEARCH

- Beam Search begins with k randomly generated states. At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts. Otherwise, it selects the k best successors from the complete list and repeats.

BEAM SEARCH

- How does it compare to running multiple random-restart hillclimbing searches in parallel?
- In beam search, useful information is passed among the parallel search threads. In effect, the states that generate the best successors say to the others, “Come over here, the grass is greener!”

BEAM SEARCH

- Beam search can suffer from a lack of diversity among the k states.
- Stochastic beam search chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value.
- Stochastic beam search bears some resemblance to the process of natural selection, whereby the "successors" (offspring) of a "state" (organism) populate the next generation according to its "value" (fitness). ...

GENETIC ALGORITHMS

- A genetic algorithm (or GA) is a variant of stochastic beam search in which successor states are generated by combining two parent states rather than by modifying a single state.

- Like beam searches, GAs begin with a set of k randomly generated states, called the population.
- Each state is rated by the objective function, or (in GA terminology) the fitness function. A fitness function should return higher values for better states.
- A crossover point is chosen randomly from the positions in the string, and a new state is created by combining two parents.

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

AIMA, Fig 4.8, p129

GENETIC ALGORITHMS: AN OVERVIEW

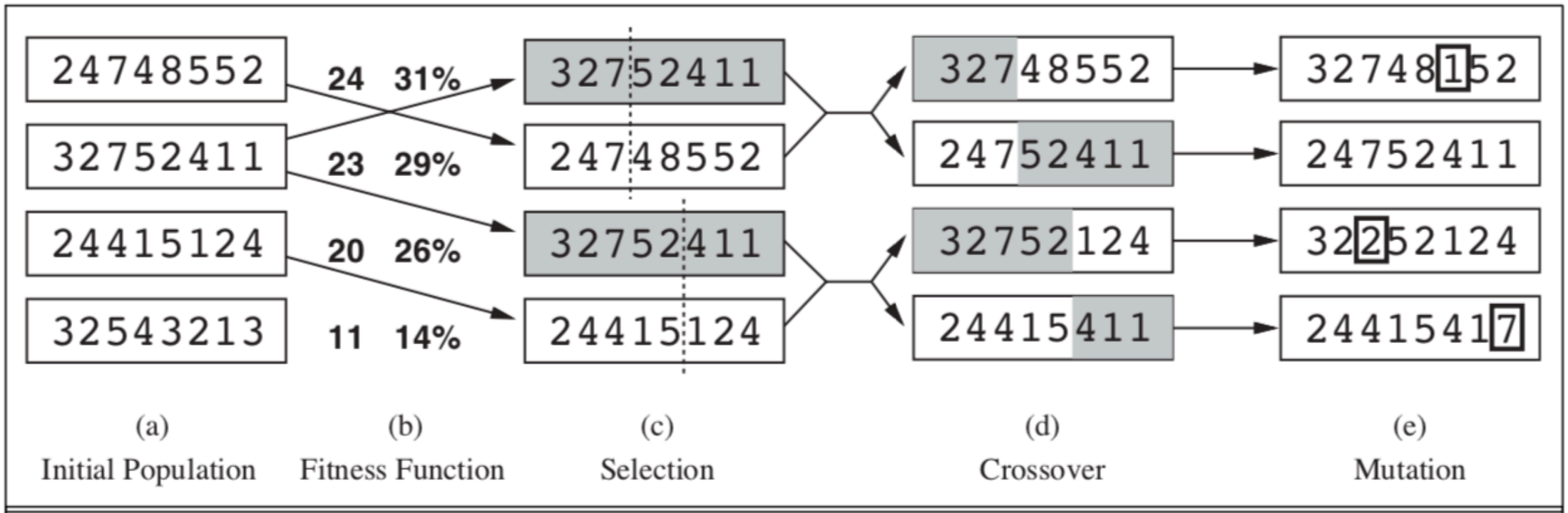


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

AIMA, p127

GENETIC ALGORITHMS: CROSSOVER

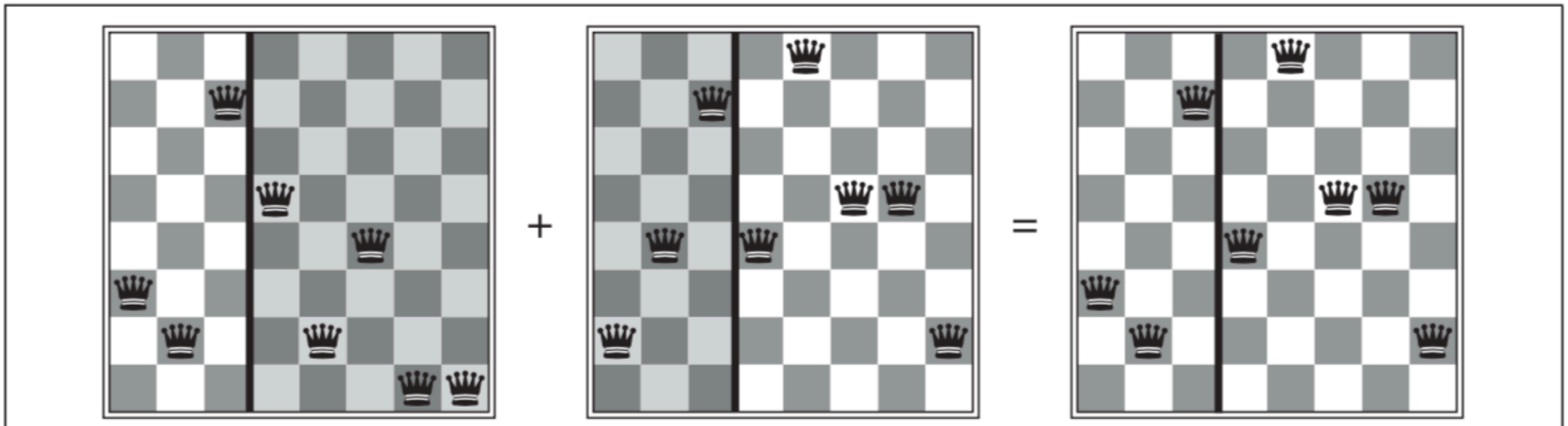


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The shaded columns are lost in the crossover step and the unshaded columns are retained.

AIMA, p127

THIS TIME: LOCAL SEARCH

- Pros – low memory requirements (O(1)), on-line, work in continuous spaces, applicable to optimization problems, can (usually) be applied to bigger problems than systematic search
- Cons – may return suboptimal solutions (get stuck in local minima) or no solution at all (but a little randomness helps limit these problems)
- Later – search in continuous spaces, newton's method, line search, gradient descent, and stochastic gradient descent

SUMMARY

- Online quiz on Monday – local search and CSPs (next)
- Local search can solve problems which are too hard for systematic search (such as some NP-hard ones).
- Key local search techniques:
  - Hillclimbing (with sideways moves and random restart)
  - Simulated Annealing
  - Local Beam Search
  - Genetic Algorithms