AI: Representation and Problem Solving

Local Search

Instructors: Pat Virtue & Stephanie Rosenthal

Slide credits: CMU AI, http://ai.berkeley.edu

Learning Objectives

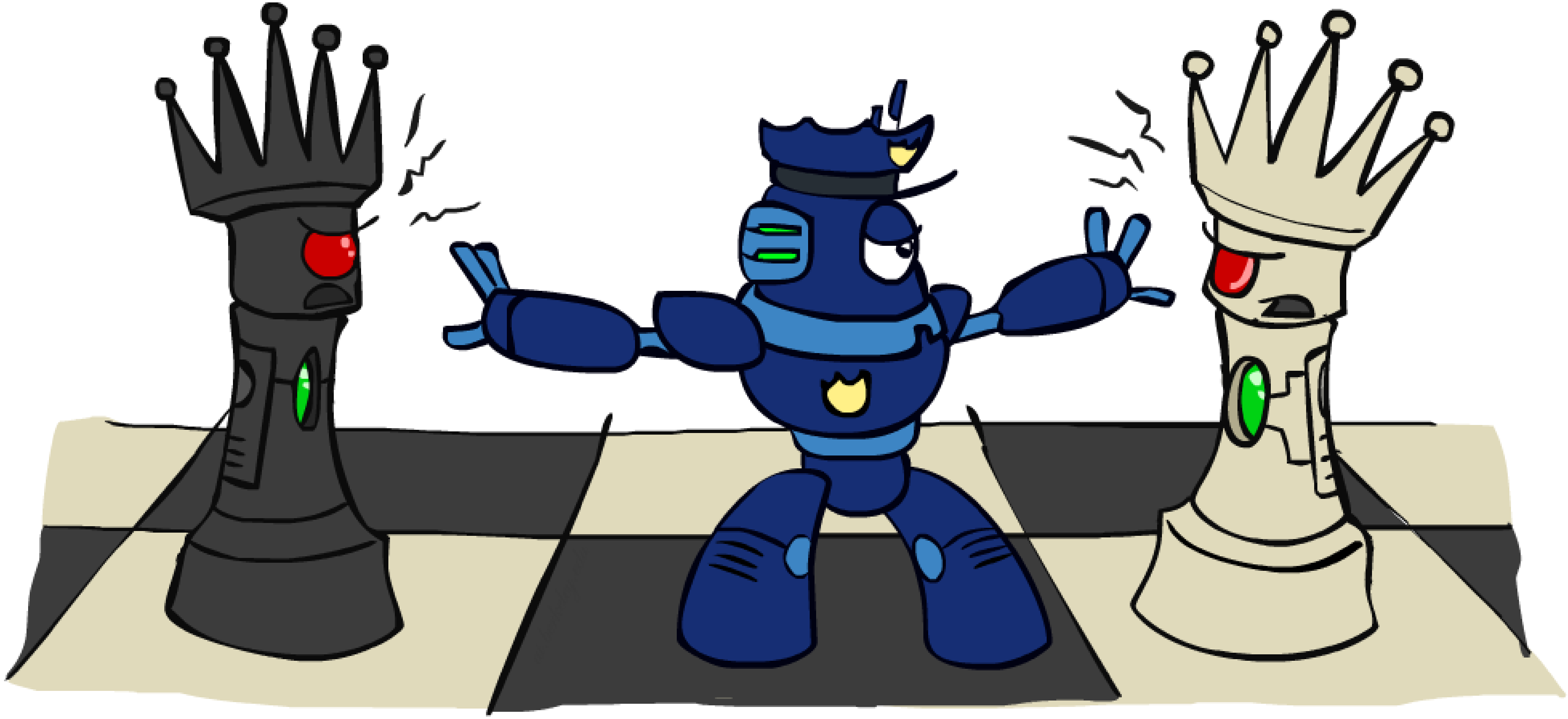
- Describe and implement the following local search algorithms
  - Iterative improvement algorithm with min-conflict heuristic for CSPs
  - Hill Climbing (Greedy Local Search)
  - Random Walk
  - Simulated Annealing
  - Beam Search
  - Genetic Algorithm
- Identify completeness and optimality of local search algorithms
- Compare different local search algorithms as well as contrast with classical search algorithms
- Select appropriate local search algorithms for real-world problems

Local Search

- Can be applied to identification problems (e.g., CSPs), as well as some planning and optimization problems

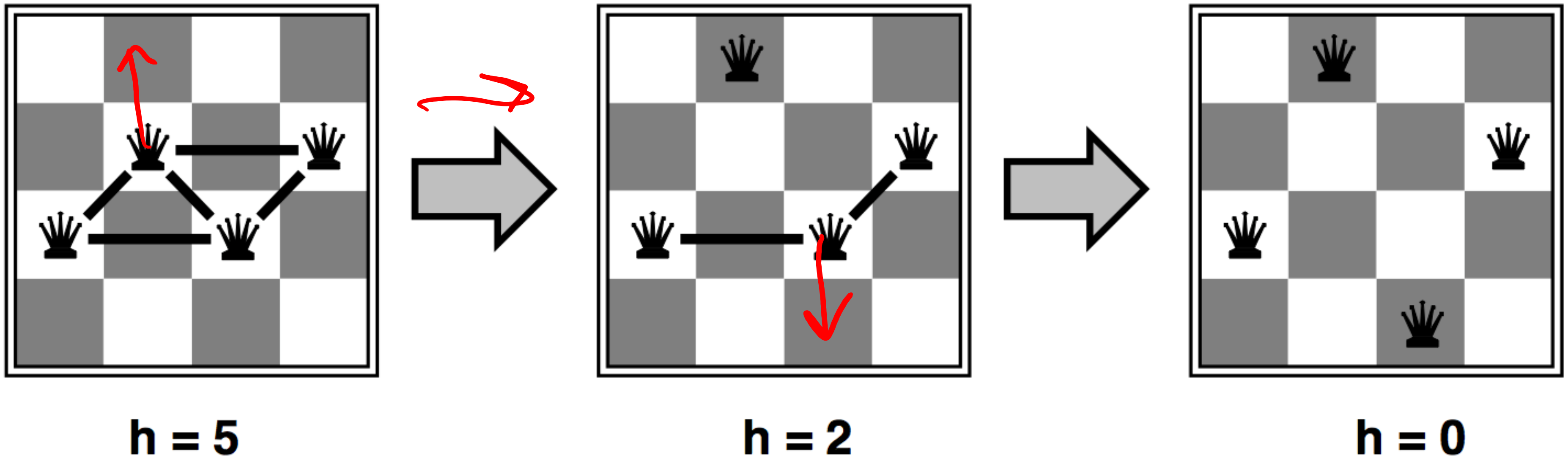
- Typically use a complete-state formulation, e.g., all variables assigned in a CSP (may not satisfy all the constraints)

Iterative Improvement for CSPs



Iterative Improvement for CSPs

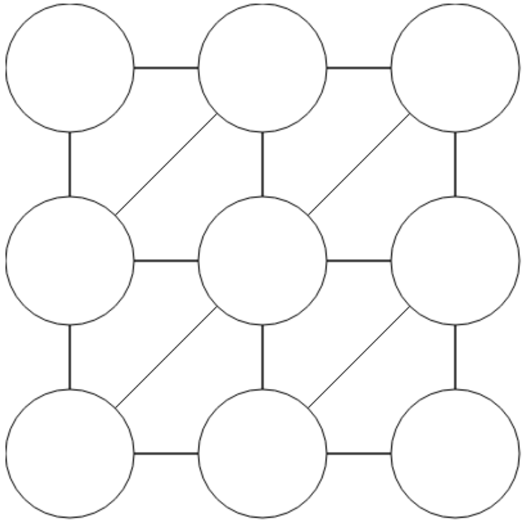
- Start with an arbitrary assignment, iteratively *reassign* variable values
- While not solved,
  - Variable selection: randomly select a conflicted variable
  - Value selection with min-conflicts heuristic h: Choose a value that violates the fewest constraints (break tie randomly)
- For n-Queens: Variables $x_i \in \{1..n\}$; Constraints $x_i \neq x_j, \left|x_i - x_j\right| \neq |i - j|, \forall i \neq j$



Demo – n-Queens

[Demo: n-queens – iterative improvement (L5D1)]

Demo – Graph Coloring

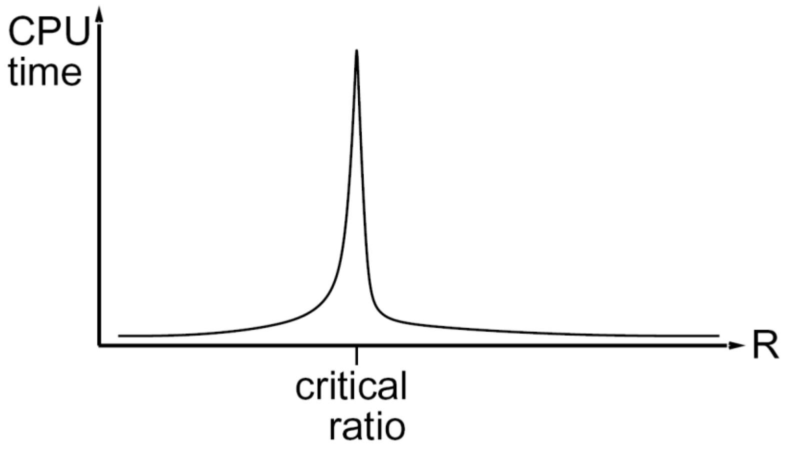


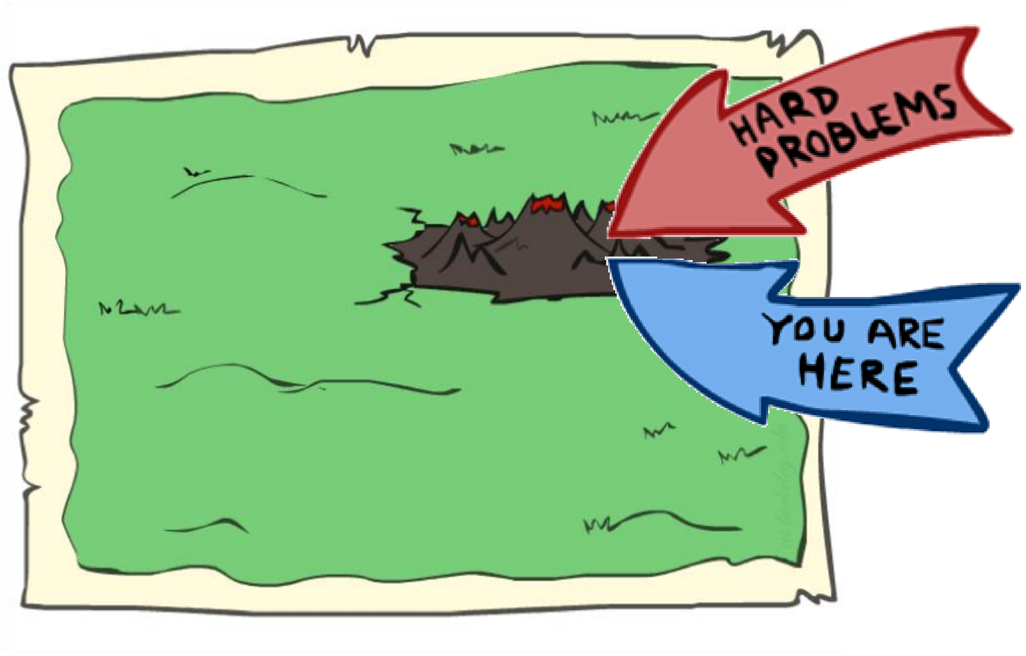
Iterative Improvement for CSPs

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!

- Same for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$

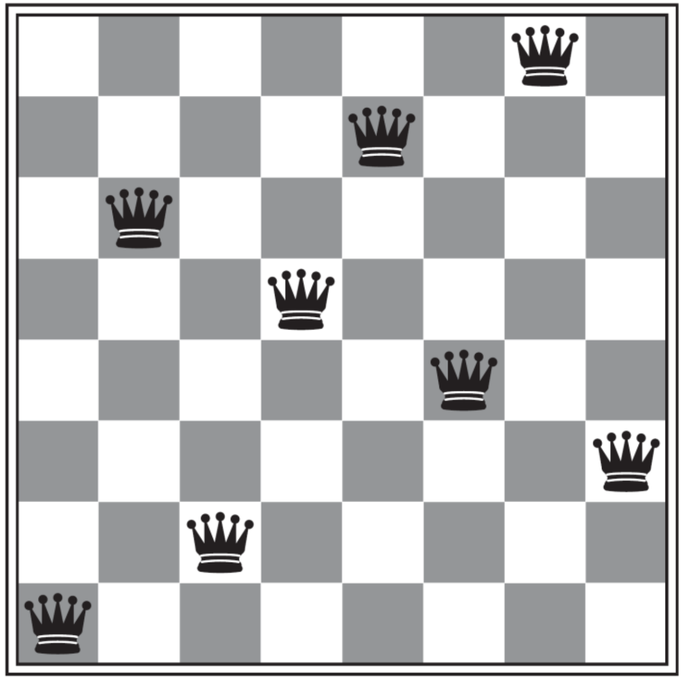




Local Search

- A local search algorithm is…
  - Complete if it always finds a goal if one exists
  - Optimal if it always finds a global minimum/maximum

$h = 1$

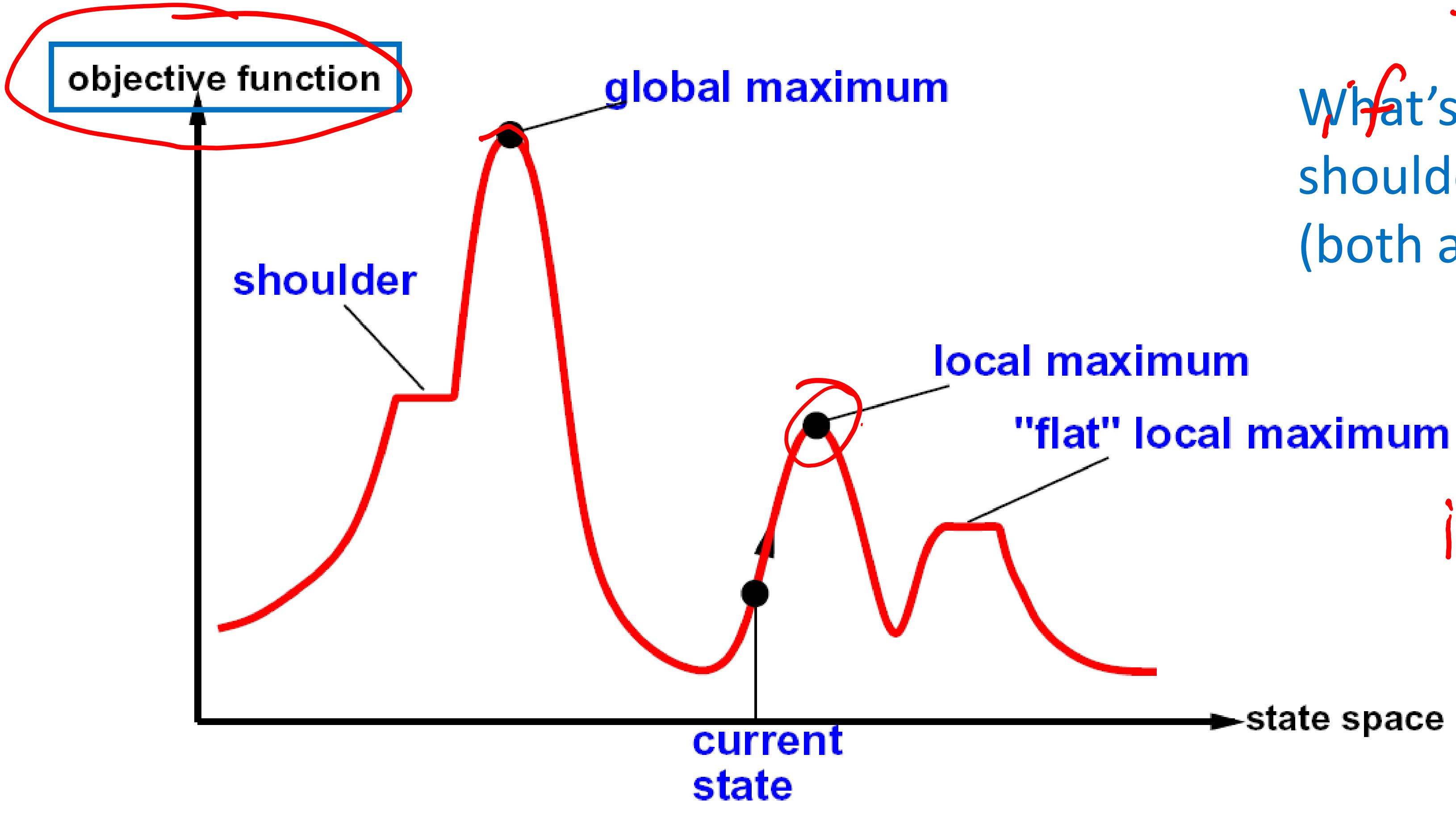


Is Iterative Improvement for CSPs complete?

No! May get stuck in a local optima

State-Space Landscape

In identification problems, could be a function measuring how close you are to a valid solution, e.g., $-1 \times$ #conflicts in n-Queens/CSP



What's the difference between shoulder and flat local maximum (both are plateaux)?

s is local maximum if $h(s) \geq h(s')$ for s' in children(s)

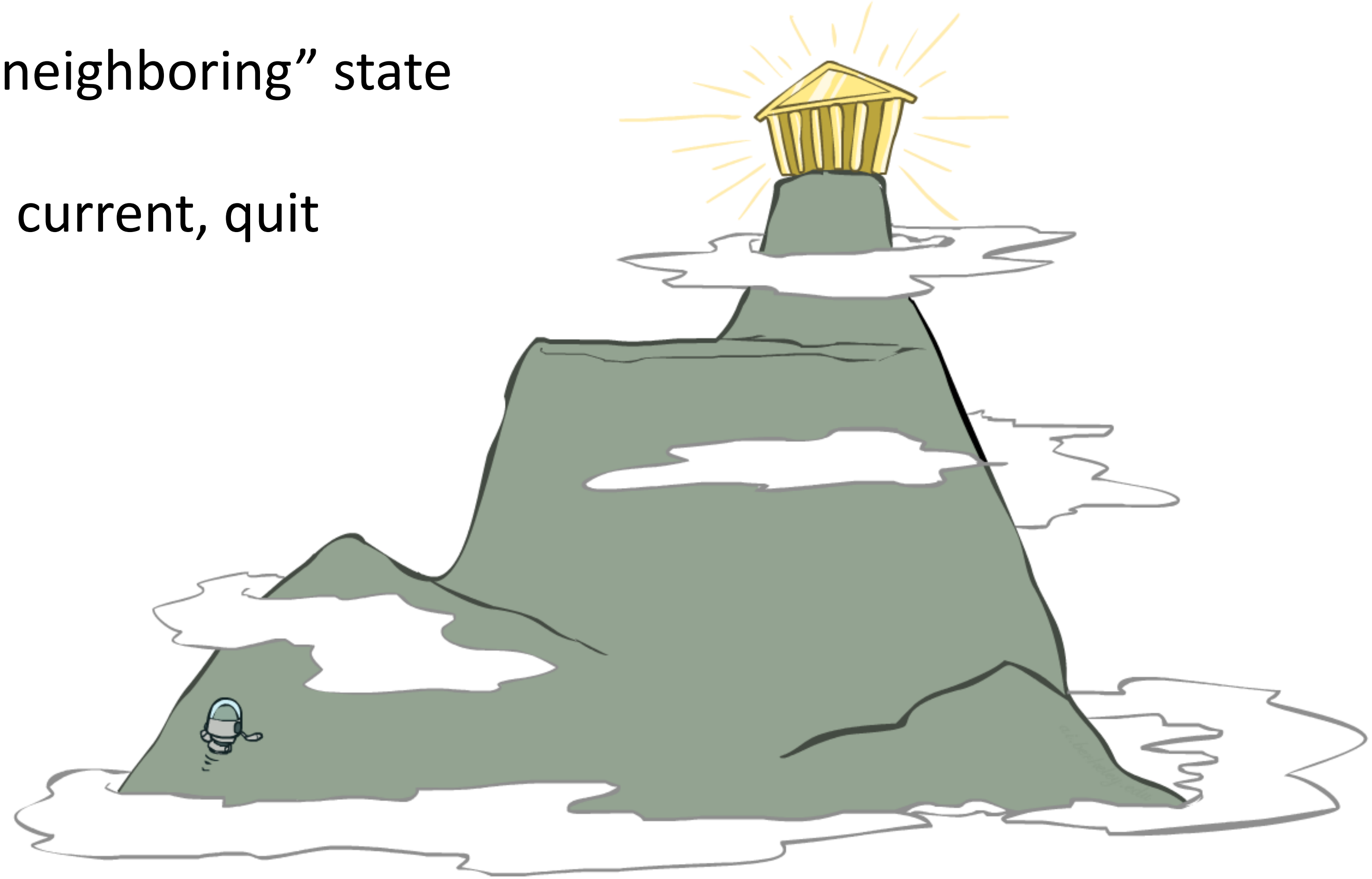
if $h(s) \geq h(s')$ for all s'

Hill Climbing (Greedy Local Search)

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best "neighboring" state (successor state)
  - If no neighbors better than current, quit

Complete? No!

Optimal? No!



Hill Climbing (Greedy Local Search)

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*

How to apply Hill Climbing to n-Queens? How is it different from Iterative Improvement?

Define a state as a board with n queens on it, one in each column

Define a successor (neighbor) of a state as one that is generated by moving a single queen to another square in the same column How many successors?

Hill Climbing (Greedy Local Search)

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*

What if there is a tie?

Typically break ties randomly

What if we do not stop here? Make a sideway move if "="

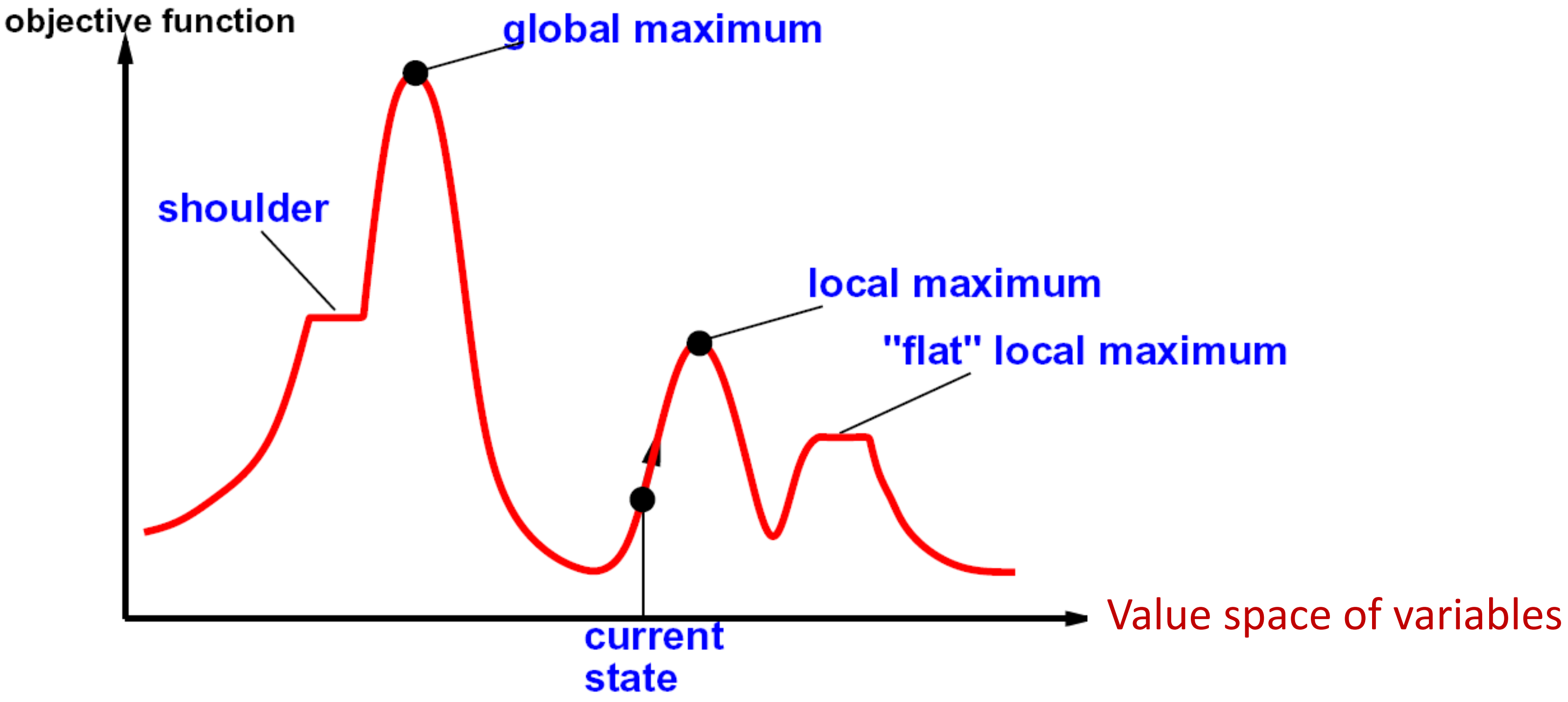
- In 8-Queens, steepest-ascent hill climbing solves 14% of problem instances
  - Takes 4 steps on average when it succeeds, and 3 steps when it fails
- When allow for ≤100 consecutive sideway moves, solves 94% of problem instances
  - Takes 21 steps on average when it succeeds, and 64 steps when it fails

Variants of Hill Climbing

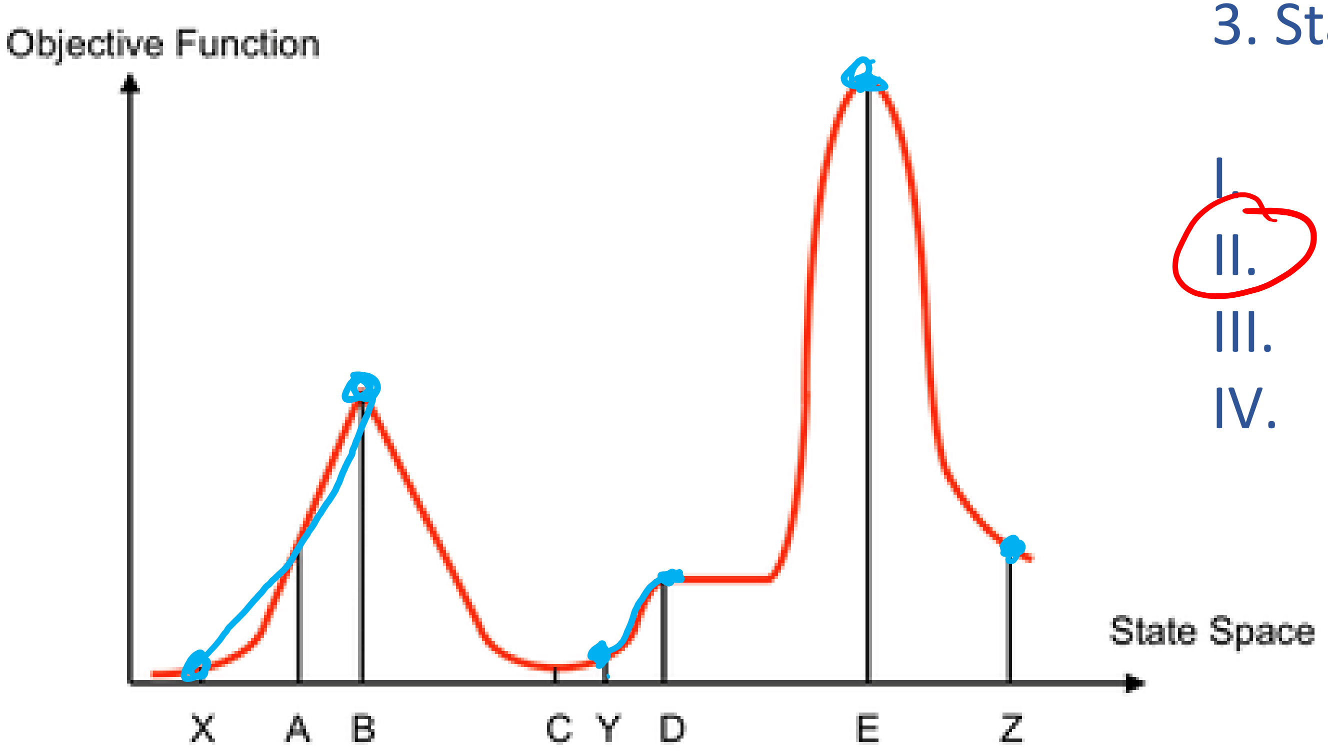
- Random-restart hill climbing
  - "If at first you don't succeed, try, try again."
  - Complete!
  - What kind of landscape will random-restarts hill climbing work the best?
- Stochastic hill climbing
  - Choose randomly from the uphill moves, with probability dependent on the "steepness" (i.e., amount of improvement)
  - Converge slower than steepest ascent, but may find better solutions
- First-choice hill climbing
  - Generate successors randomly (one by one) until a better one is found
  - Suitable when there are too many successors to enumerate

Variants of Hill Climbing

- What if variables are continuous, e.g. find $x \in [0,1]$ that maximizes $f(x)$?
  - Gradient ascent
    - Use gradient to find best direction
    - Use the magnitude of the gradient to determine how big a step you move



Piazza Poll 1: Hill Climbing



1. Starting from X, where do you end up?
2. Starting from Y, where do you end up?
3. Starting from Z, where do you end up?

I. $X \to A, Y \to D, Z \to E$

II. $X \to B, Y \to D, Z \to E$

III. $X \to X, Y \to C, Z \to Z$

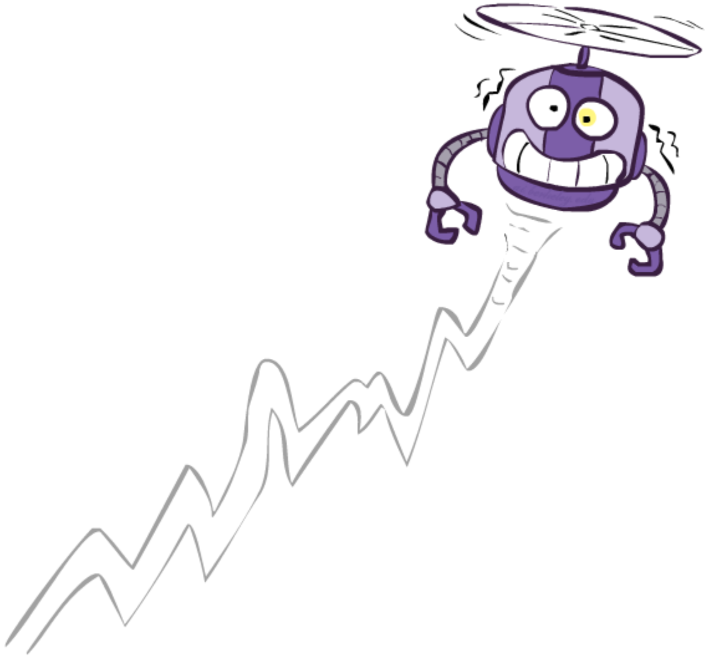
IV. I don't know

Random Walk

- Uniformly randomly choose a neighbor to move to

- Complete but inefficient!

Simulated Annealing



- Combines random walk and hill climbing
- Complete and efficient
- Inspired by statistical physics
- Annealing – Metallurgy
  - Heating metal to high temperature then cooling
  - Reaching low energy state
- Simulated Annealing – Local Search
  - Allow for downhill moves and make them rarer as time goes on
  - Escape local maxima and reach global maxima

Simulated Annealing

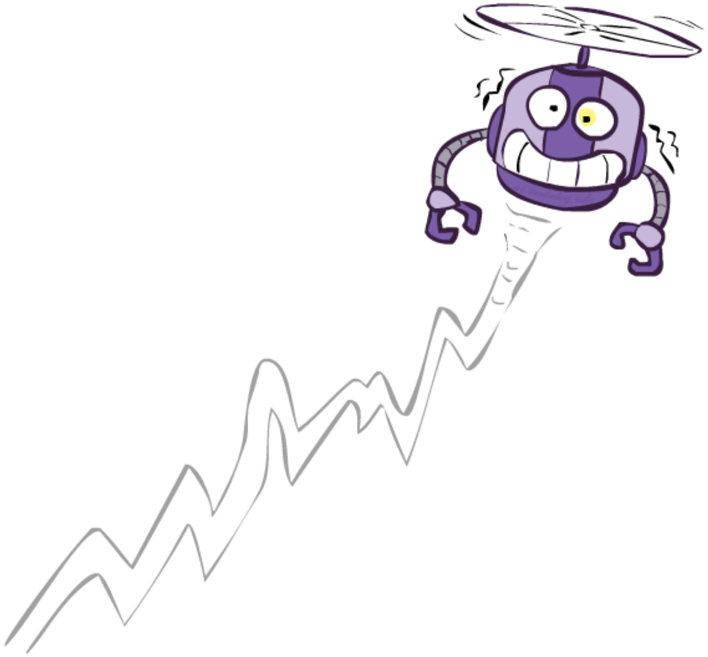
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
     T ← schedule(t)
     if T = 0 then return current
     next ← a randomly selected successor of current
     ΔE ← next.VALUE – current.VALUE
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

Control the change of temperature T (↓ over time)

Almost the same as hill climbing except for a *random* successor

Unlike hill climbing, move downhill with some prob.

Simulated Annealing



- $\mathbb{P}[\text{move downhill}] = e^{\Delta E/T}$
  - Bad moves are more likely to be allowed when T is high (at the beginning of the algorithm)
  - Worse moves are less likely to be allowed
- Stationary distribution: $p(x) \propto e^{\frac{E(x)}{kT}}$
- Guarantee: If T decreased slowly enough, will converge to optimal state!
- But! In reality, the more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row

Local Beam Search

- Keep track of k states
- In each iteration
  - Generate all successors of all k states
  - Only retain the best k successors among them all

How is this different from K local searches with different initial states in parallel?

The searches ***communicate***! “Come over here, the grass is greener!”

Analogous to evolution / natural selection!

Limitations and Variants of Local Beam Search

- Suffer from a lack of diversity; Quickly concentrated in a small region of the state space

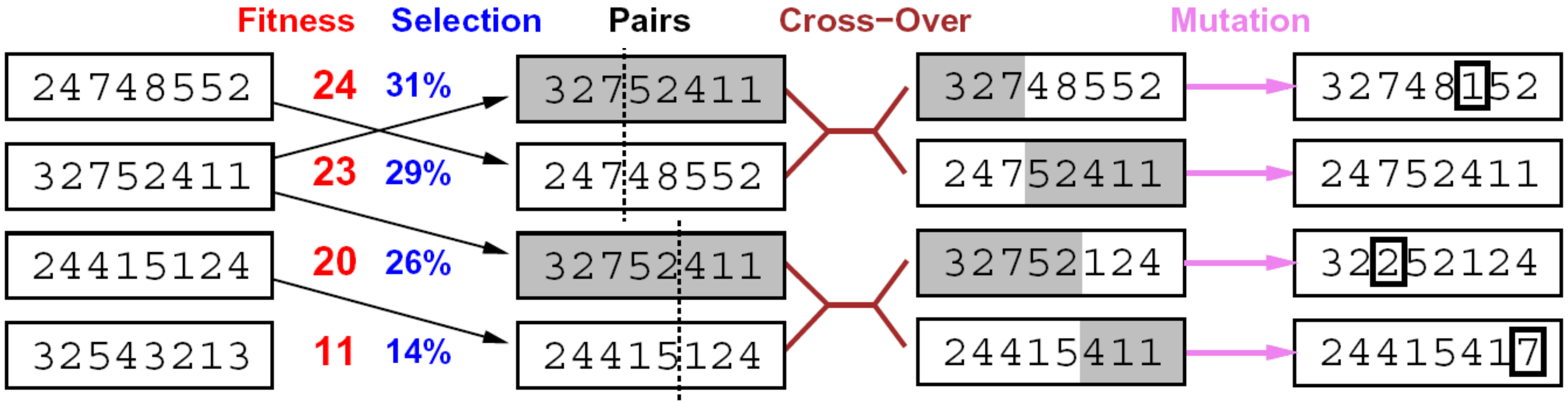
- Variant: Stochastic beam search
  - Randomly choose k successors (offsprings) of a state (organism) population according to its objective value (fitness)

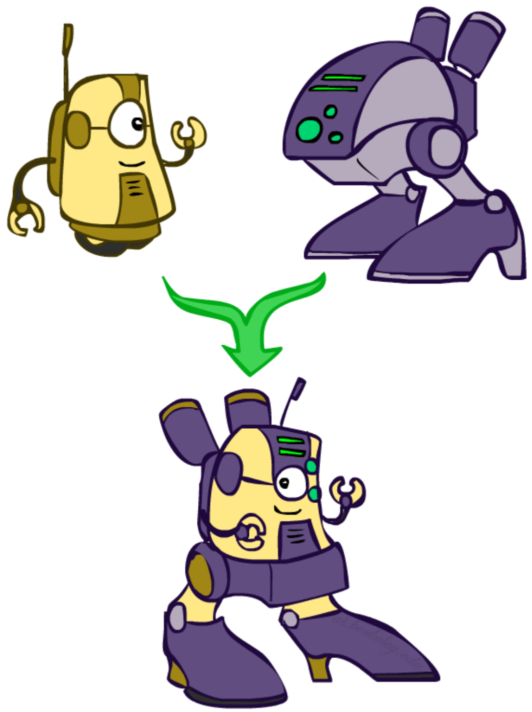
Genetic Algorithms

- Inspired by evolutionary biology
  - Nature provides an objective function (reproductive fitness) that Darwinian evolution could be seen as attempting to optimize

- A variant of stochastic beam search
  - Successors are generated by combining two parent states instead of modifying a single state (sexual reproduction rather than asexual reproduction)

Genetic Algorithms for 8-Queens

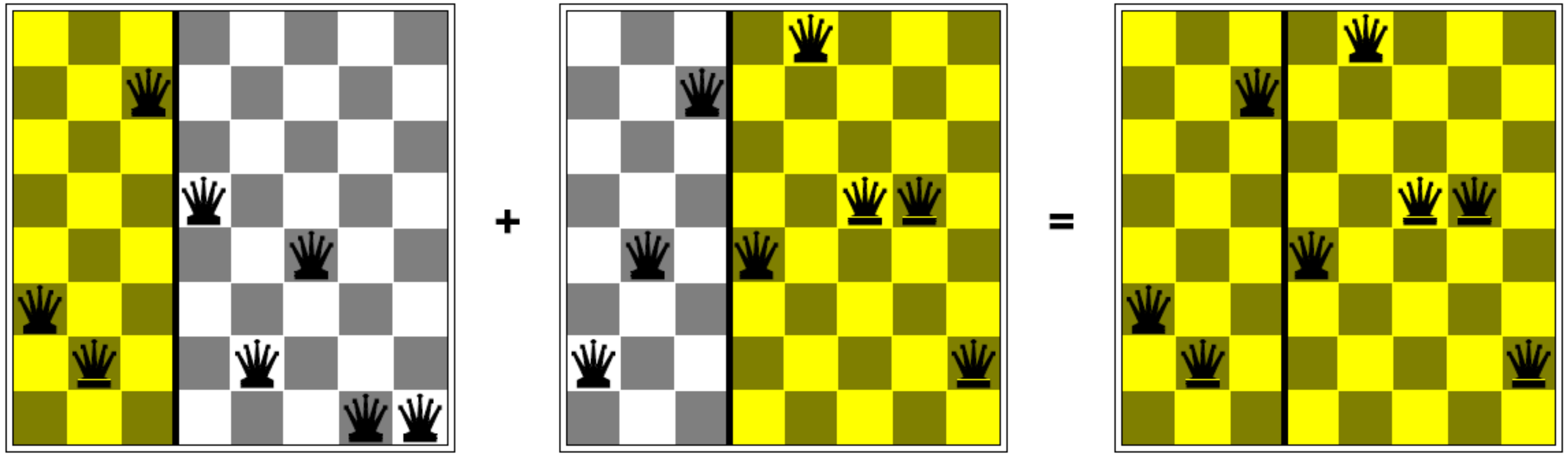




- State Representation: 8-digit string, each digit in $\{1..8\}$
- Fitness Function: #Nonattacking pairs
- Selection: Select k individuals randomly with probability proportional to their fitness value (random selection with replacement)
- Crossover: For each pair, choose a crossover point $\in \{1..7\}$, generate two offsprings by crossing over the parent strings
- Mutation (With some prob.): Choose a digit and change it to a different value in $\{1..8\}$

What if k is an odd number?

Genetic Algorithms for 8-Queens



- Why does crossover make sense here?
- Would crossover work well without a selection operator?

Genetic Algorithms

- Start with a population of k individuals (states)
- In each iteration
  - Apply a fitness function to each individual in the current population
  - Apply a selection operator to select k pairs of parents
  - Generate k offsprings by applying a crossover operator on the parents
  - For each offspring, apply a mutation operation with a (usually small) independent probability

- For a specific problem, need to design these functions and operators
- Successful use of genetic algorithms require careful engineering of the state representation!

Genetic Algorithms

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

How is this different from the illustrated procedure on 8-Queens?

Exercise: Traveling Salesman Problem

- Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?
- Input: $c_{ij}, \forall i, j \in \{0, \dots, n-1\}$
- Output: A ordered sequence $\{v_0, v_1, \dots, v_n\}$ with $v_0 = 0$, $v_n = 0$ and all other indices show up exactly once

- Question: How to apply Local Search algorithms to this problem?

Summary: Local Search

- Maintain a constant number of current nodes or states, and move to “neighbors” or generate “offsprings” in each iteration
  - Do not maintain a search tree or multiple paths
  - Typically do not retain the path to the node
- Advantages
  - Use little memory
  - Can potentially solve large-scale problems or get a reasonable (suboptimal or almost feasible) solution

Learning Objectives

- Describe and implement the following local search algorithms
  - Iterative improvement algorithm with min-conflict heuristic for CSPs
  - Hill Climbing (Greedy Local Search)
  - Random Walk
  - Simulated Annealing
  - Beam Search
  - Genetic Algorithm
- Identify completeness and optimality of local search algorithms
- Compare different local search algorithms as well as contrast with classical search algorithms
- Select appropriate local search algorithms for real-world problems