

# Artificial Intelligence

## Local Search

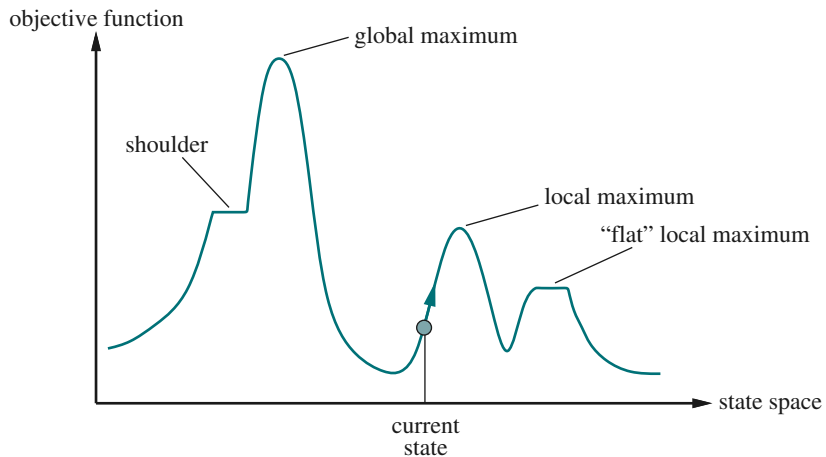
Christopher Simpkins

# Local Search

If you don't care about the path to a goal state, you can use **local search**.

- ▶ Search neighbors of current state, moving to best neighbor.
- ▶ Track only current state.
- ▶ Uses very little memory.
- ▶ Can find reasonable solutions in large or infinite state spaces.
- ▶ Often used for **optimization** problems – finding states that maximize or minimize an **objective function**.

# State Space Landscape



# Hill-Climbing Search

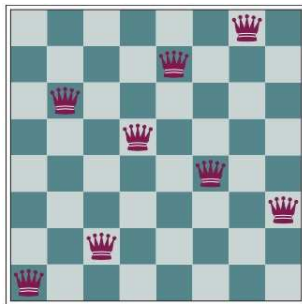
**function** HILL-CLIMBING(*problem*) **returns** a state that is a local maximum  
    *current*  $\leftarrow$  *problem*.INITIAL  
    **while** *true* **do**  
        *neighbor*  $\leftarrow$  a highest-valued successor state of *current*  
        **if** VALUE(*neighbor*)  $\leq$  VALUE(*current*) **then return** *current*  
        *current*  $\leftarrow$  *neighbor*

- ▶ Also known as **greedy local search**

# The 8 Queens Problem

**Complete-state formulation:** row position for each of 8 columns, e.g., (a) below is

$\langle 1, 6, 2, 5, 7, 4, 8, 3 \rangle$



(a)

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	16
17	16	18	15	14	16	16	16
18	14	15	15	14	16	16	16
14	14	13	17	12	14	12	18

(b)

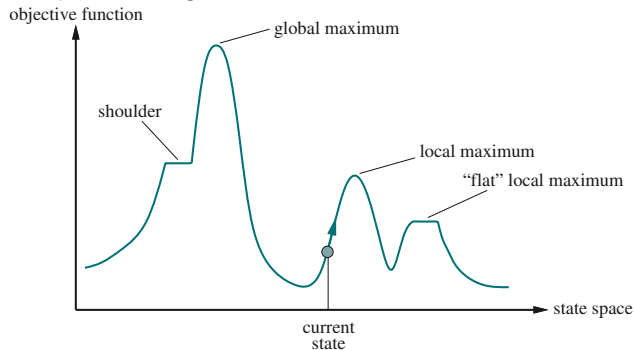
- ▶ Action: move a single queen to new row within column. Each state has  $8 \cdot 7 = 56$  successor states.
- ▶ Possible heuristic: number of pairs of attacking queens (even if blocked). (b) above has  $h = 17$ .

▶ Useful to remember:  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

# Disadvantages of Hill-Climbing

Susceptible to getting stuck in:

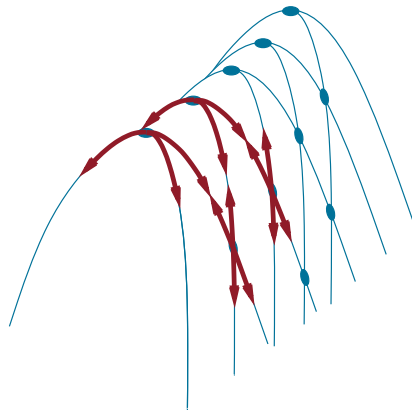
- ▶ local maxima
- ▶ ridges – sequences of local maxima
- ▶ plateaus, e.g., flat local maxima or shoulders.



How to fix:

- ▶ Allow "sideways" moves
- ▶ Stochastic hill climbing chooses randomly from uphill moves. Stochastic beam search does this with  $k$  states in parallel.
- ▶ Random restart hill climbing restarts from multiple

Grid of states superimposed on ridge rising from left to right.



# Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state  
  current  $\leftarrow$  problem.INITIAL  
  for  $t = 1$  to  $\infty$  do  
     $T \leftarrow$  schedule( $t$ )  
    if  $T = 0$  then return current  
    next  $\leftarrow$  a randomly selected successor of current  
     $\Delta E \leftarrow$  VALUE(current) – VALUE(next)  
    if  $\Delta E > 0$  then current  $\leftarrow$  next  
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```

- ▶ Based on metallurgy – gradually cool metal to reach low-energy crystalline state.
- ▶ Intuition: think of gradient descent instead of gradient ascent – multiple shallow valleys, one deepest valley. Shake ball out of shallow valleys into deepest valley.
- ▶ Similar to hill climbing, but picks a random move and
  - ▶ accepts it if its better,
  - ▶ if not better, accept with probability  $< 1$ .
- ▶ Probability of accepting a worse move depends on:
  - ▶ how much worse the move is,  $\Delta E$ , and
  - ▶ the current “temperature,”  $T$ .

If  $T$  decreases sufficiently slowly, then the Boltzman distribution,  $e^{\frac{\Delta E}{T}}$ , ensures that all the probability is concentrated on the global maxima, so the algorithm finds a global maximum with probability approaching 1.

# Evolutionary (Genetic) Algorithms

A kind of **local beam search**: tracking  $k$  states instead of just one.

Elements of genetic algorithms:

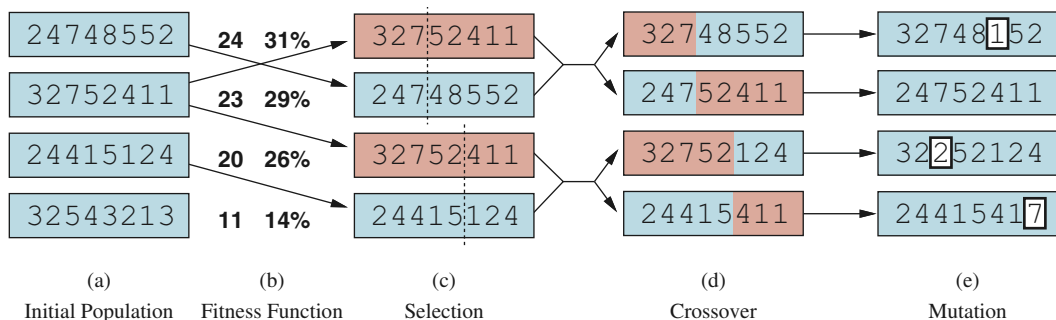
- ▶ Fitness function.
- ▶ Population size.
- ▶ Candidate representation:
  - ▶ Typically a string (vector) over a finite alphabet.
  - ▶ **Evolution strategies**: sequence of real numbers.
  - ▶ **Genetic programming**: computer programs.
- ▶ Mixing number,  $\rho$ : number of “parents” from which to generate new candidates. When  $\rho = 1$ , stochastic beam search.
- ▶ Selection process for choosing “parents.”
- ▶ Recombination procedure.
- ▶ Mutation rate.
- ▶ Composition of next generation.
  - ▶ Elitism: choose top-scoring candidates.
  - ▶ Culling: eliminate bottom-scoring candidates.

# A Genetic Algorithm

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights  $\leftarrow$  WEIGHTED-BY(population, fitness)
    population2  $\leftarrow$  empty list
    for i = 1 to SIZE(population) do
      parent1, parent2  $\leftarrow$  WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child  $\leftarrow$  REPRODUCE(parent1, parent2)
      if (small random probability) then child  $\leftarrow$  MUTATE(child)
      add child to population2
    population  $\leftarrow$  population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness
```

```
function REPRODUCE(parent1, parent2) returns an individual
  n  $\leftarrow$  LENGTH(parent1)
  c  $\leftarrow$  random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

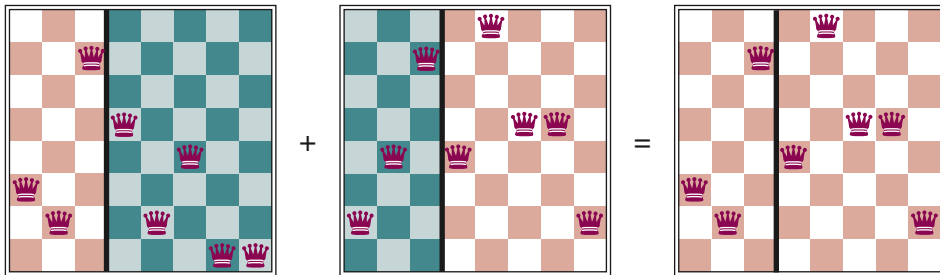
# Genetic Algorithm on 8-Queens Problem



1. Population is generated in (a).
2. Fitness function is applied to population, which is then ranked by fitness score.
3. Highest-scoring candidates are selected for reproduction in (c).
4. Crossover operation is applied to candidates in (c) to produce "children" in (d).
5. In (e) "offspring" are randomly chosen for mutation. For each chosen candidate, a "gene" is randomly chosen, then that gene is assigned a random "mutated" value.

## Crossover in the 8-queens Problem

Here is a pictorial illustration of the crossover operation in the 8-queens problem:



Is the random crossover operation depicted here meaningful for the 8-queens problem?

# Genetic Algorithms and Biological Evolution

Genetic algorithms borrow the language of biological evolution for marketing purposes, but are far more simplistic than biological evolution. My takes:

- ▶ Genetic algorithms are just stochastic beam search with “sexual” successor generation and “mutation.”
- ▶ If there is no meaningful crossover operation, genetic algorithms are just random walks in the state space graph.

There is an interesting connection between biological evolution and AI, in particular learning.

- ▶ Learning is adaptation. With experience an agent adapts to a task, getting better at the task.
- ▶ Biological evolution can be seen as a learning process whereby specieses “learn” to perform better in their environments.
- ▶ The Baldwin effect: immutable traits vs. online learning ability.
  - ▶ Plasticity, or the ability to learn, allows a species to adapt to an environment for which it is ill-suited. E.g., building shelters, fire, etc. in cold regions.
  - ▶ Things that are harder, or impossible, to learn online must be encoded in the genome. E.g., the way our body uses the air it breathes or the sun.

# Continuous State Spaces – Airports In Romania



If each airport  $x_i$  is at location  $(x_i, y_i)$  and the set of cities closest to airport  $x_i$  is  $C_i$ , then

$$f(x) = f(x_1, y_1, x_2, y_2, x_3, y_3)$$

and we want to minimize

$$f(x) = \sum_{i=1}^3 \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

For a globally optimal solution, if the airports move “too much,” the sets  $C_i$  change. How to deal with that?

# Local Gradient Descent

The gradient of

$$f(\mathbf{x}) = \sum_{i=1}^3 \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

is

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

But that would only work for one airport. We can decompose it into three local problems:

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$$

$$\frac{\partial f}{\partial x_2} = 2 \sum_{c \in C_2} (x_2 - x_c)$$

$$\frac{\partial f}{\partial x_3} = 2 \sum_{c \in C_3} (x_3 - x_c)$$

$$\frac{\partial f}{\partial y_1} = 2 \sum_{c \in C_1} (y_1 - y_c)$$

$$\frac{\partial f}{\partial y_2} = 2 \sum_{c \in C_2} (y_2 - y_c)$$

$$\frac{\partial f}{\partial y_3} = 2 \sum_{c \in C_3} (y_3 - y_c)$$

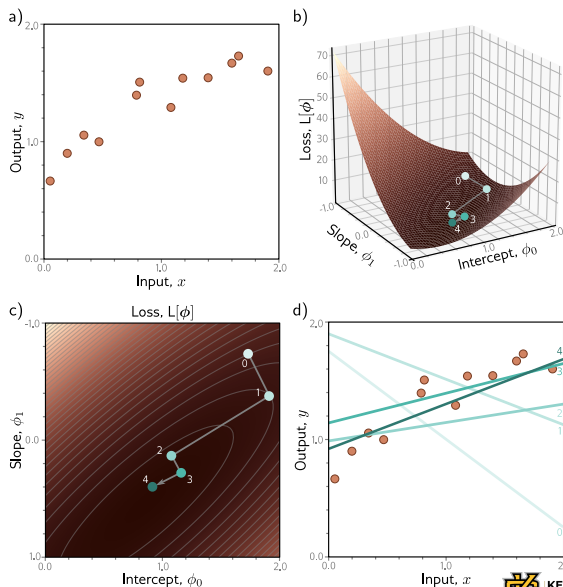
# Gradient Descent in Action

Given our 3 gradient expressions, we can use the update rule:

$$x \leftarrow x + \alpha \nabla f(x)$$

where  $\alpha$  is a **step size**, or learning rate.

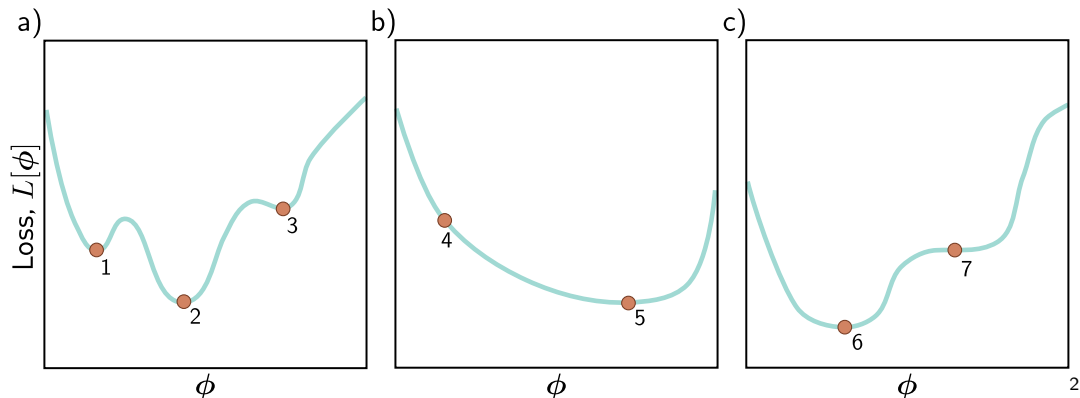
- ▶ What if  $\alpha$  is “too big?”
- ▶ What if  $\alpha$  is “too small?”



<sup>1</sup><https://udlbook.github.io/udlbook/>

# Continuous State Spaces and Convexity

A convex set is a set of points in which a line between any two points lies within the set. A convex function is a function for which the points above the function form a convex set.



There are mathematical properties of continuous spaces that rule out local minima. Take my deep learning class to learn about them!

<sup>2</sup><https://udlbook.github.io/udlbook/>

# Constrained Optimization via Linear Programming

$$\begin{aligned} &\text{maximize } 3x_1 + 2x_2 \\ &\text{subject to } -x_1 + 3x_2 \leq 12 \\ &\quad x_1 + x_2 \leq 8 \\ &\quad 2x_1 - x_2 \leq 10 \\ &\quad x_1, x_2 \geq 0 \end{aligned}$$

