Last update: February 2, 2010

Local search algorithms

CMSC 421: Chapter 4, Sections 3–4

Iterative improvement algorithms

In many optimization problems, the **path** to a goal is irrelevant;
the goal state itself is the solution

Then state space = a set of goal states
find one that satisfies constraints (e.g., no two classes at same time)
or, find **optimal** one (e.g., highest possible value, least possible cost)

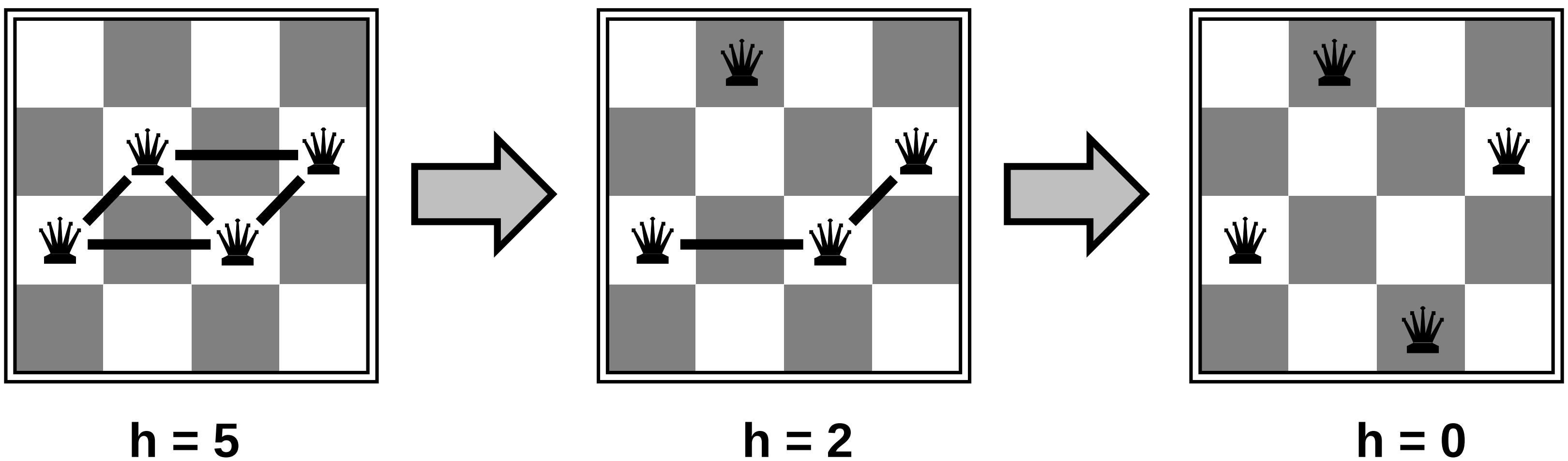
In such cases, can use *iterative improvement* algorithms;
keep a single "current" state, try to improve it

◇ Constant space
◇ Suitable for online as well as offline search

Example: the n-Queens Problem

◇ Put n queens on an $n \times n$ chessboard
◇ No two queens on the same row, column, or diagonal

Iterative improvement:
Start with one queen in each column
move a queen to reduce number of conflicts



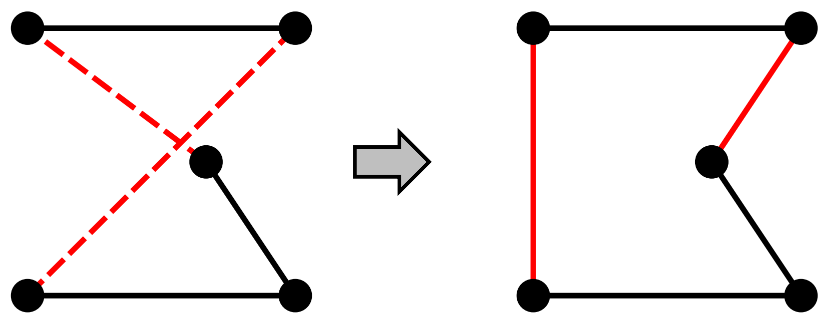
Even for very large n (e.g., $n = 1\ million$),
this usually finds a solution almost instantly

Example: Traveling Salesperson Problem

- Given a *complete* graph (edges between all pairs of nodes)
- A *tour* is a cycle that visits every node exactly once
- Find a least-cost *tour* (simple cycle that visits each city exactly once)

Iterative improvement:

Start with any tour, perform pairwise exchanges



Variants of this approach get within 1% of optimal very quickly with thousands of cities

Outline

- Hill-climbing
- Simulated annealing
- ~~Genetic algorithms (briefly)~~
- ~~Local search in continuous spaces (very briefly)~~

Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in thick fog with amnesia"

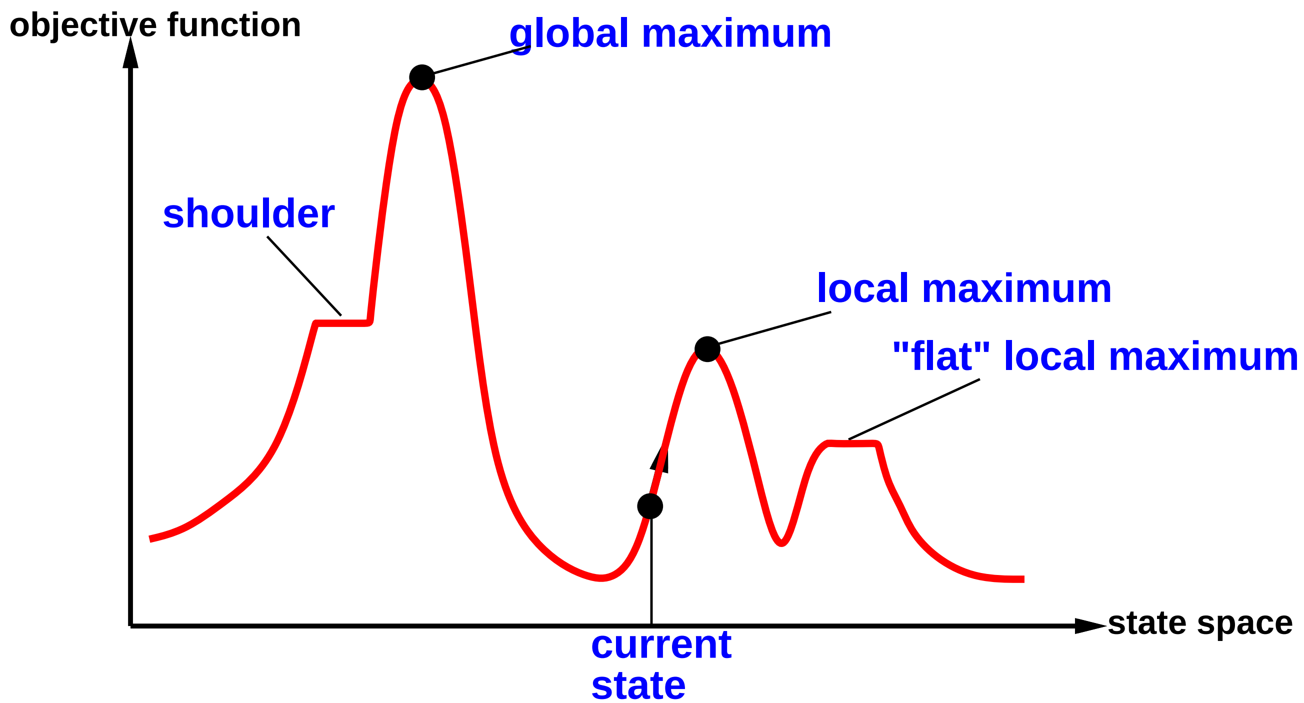
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

At each step, move to a neighbor of higher value
in hopes of getting to a solution having the highest possible value

Can easily modify this for problems where we want to minimize rather than maximize

Hill-climbing, continued

Useful to consider *state space landscape*



Random-restart hill climbing: repeat with randomly chosen starting points

Russell & Norvig say it's trivially complete; they're almost right

If finitely many local maxima, then $\lim_{\text{restarts}\to\infty} P(\text{complete}) = 1$

Simulated annealing

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

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

A simple example

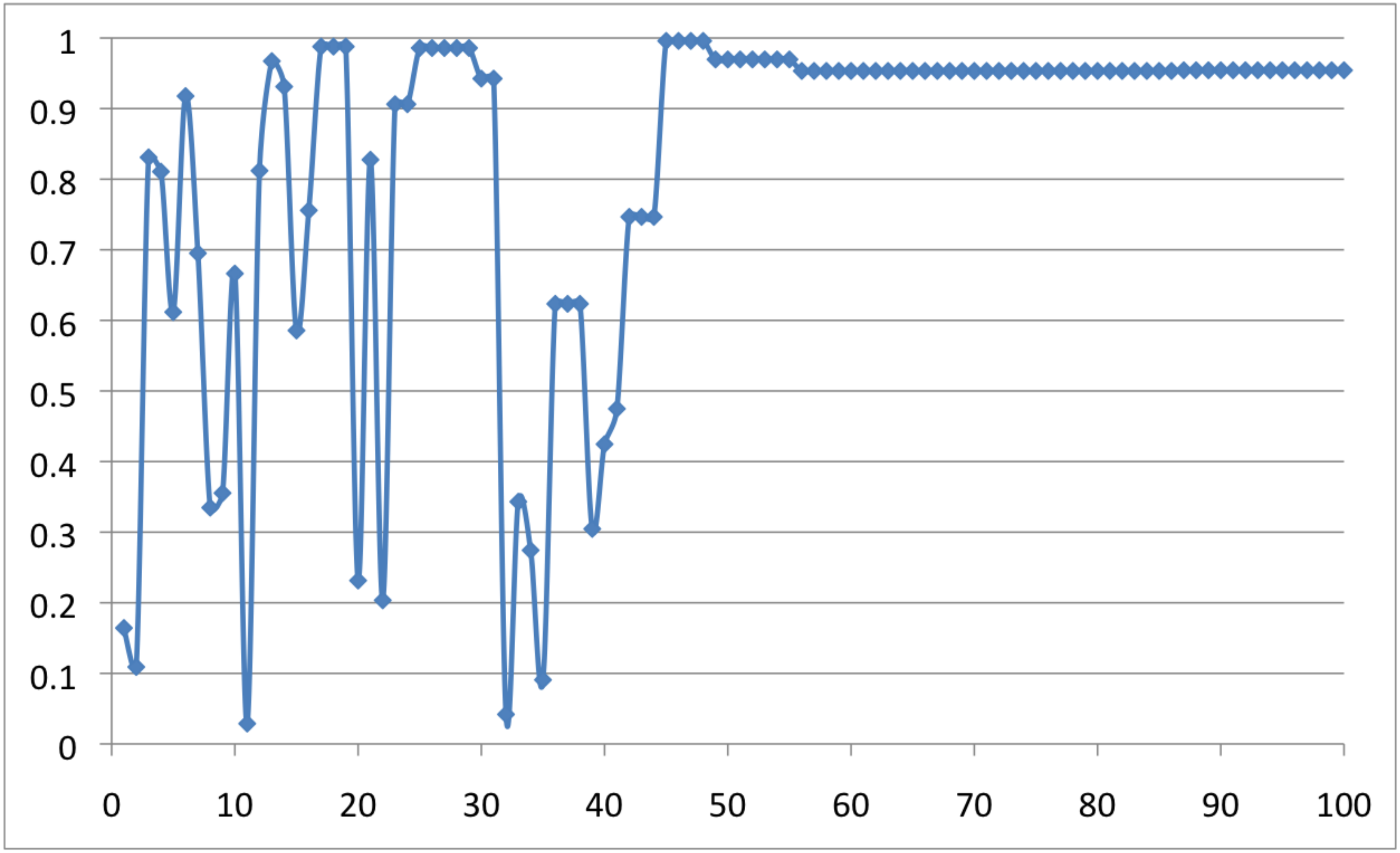
Each state is a number $x \in [0, 1]$, initial state is 0, all states are neighbors, $\textsc{Value}(x) = x^2$, 100 iterations, $schedule[i] = 10 \times 0.9^i$

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  $current \leftarrow \textsc{Make-Node}(\textsc{Initial-State}[problem])$
  for $i \leftarrow$ 1 **to** ∞ **do**
    $T \leftarrow schedule[i]$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow \textsc{Value}[next] - \textsc{Value}[current]$
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else with probability $e^{\Delta E/T}$, set $current \leftarrow next$

Simple example, continued

100 iterations, each state is a number $x \in [0,1]$, initial state is $x = 0$, $\textsc{Value}(x) = x^2$, all states are neighbors, $schedule[i] = 10 \times 0.9^i$



Properties of simulated annealing

At fixed "temperature" T, probability of being in any given state x reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

for every state x other than x^* and for small T,

$$p(x^*)/p(x) = e^{\frac{E(x^*)}{kT}}/e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$$

From this it can be shown that
if we decrease T slowly enough, $\mathrm{Pr}[\text{reach } x^*]$ approaches 1

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

Local beam search

function BEAM-SEARCH(*problem*, k) **returns** a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution **then** return it
        else select the k best successors

Not the same as k parallel searches
Searches that find good states will recruit other searches to join them

Problem: often all k states end up on same local hill

Stochastic beam search:
    choose k successors randomly, biased towards good ones

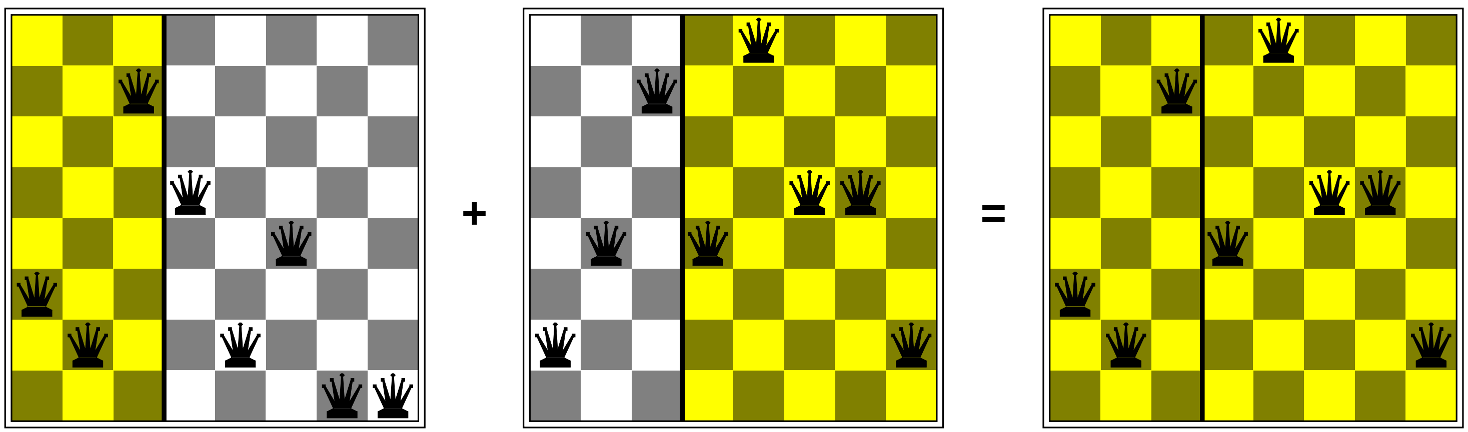
Close analogy to natural selection

Genetic algorithms

Genetic algorithms
$=$ stochastic local beam search $+$ generate successors from **pairs** of states

Each state should be a string of characters;
Substrings should be meaningful components

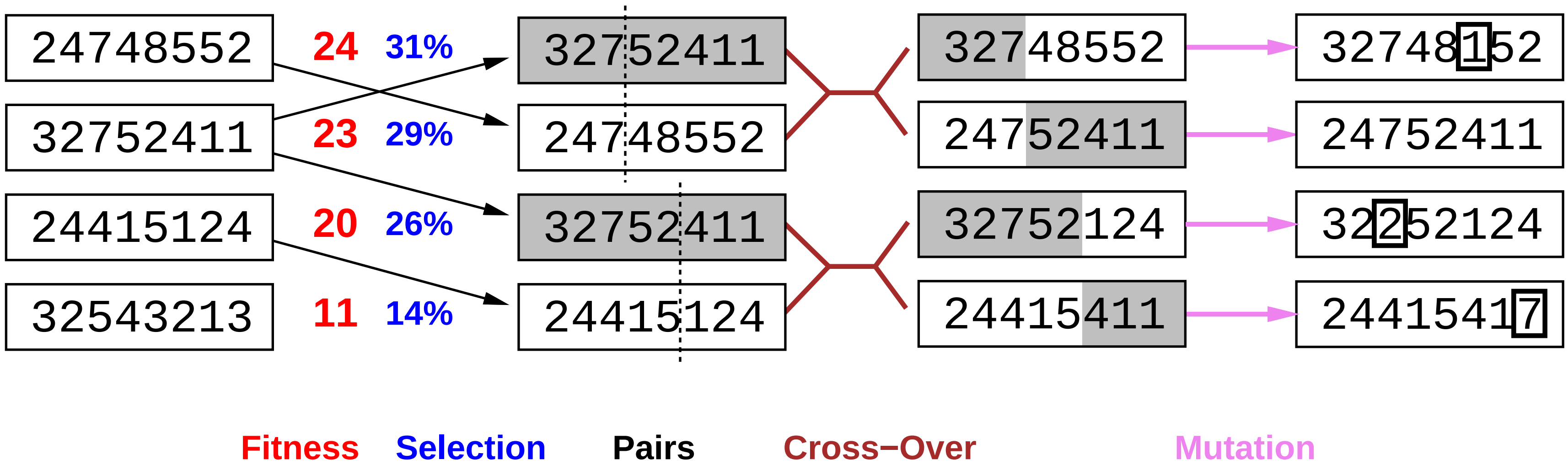
Example: n-queens problem
i'th character $=$ row where i'th queen is located



$\boxed{672}$ 47588 752 $\boxed{51447}$ $\boxed{672}\boxed{51447}$

Genetic algorithms

Genetic algorithms
= stochastic local beam search + generate successors from **pairs** of states

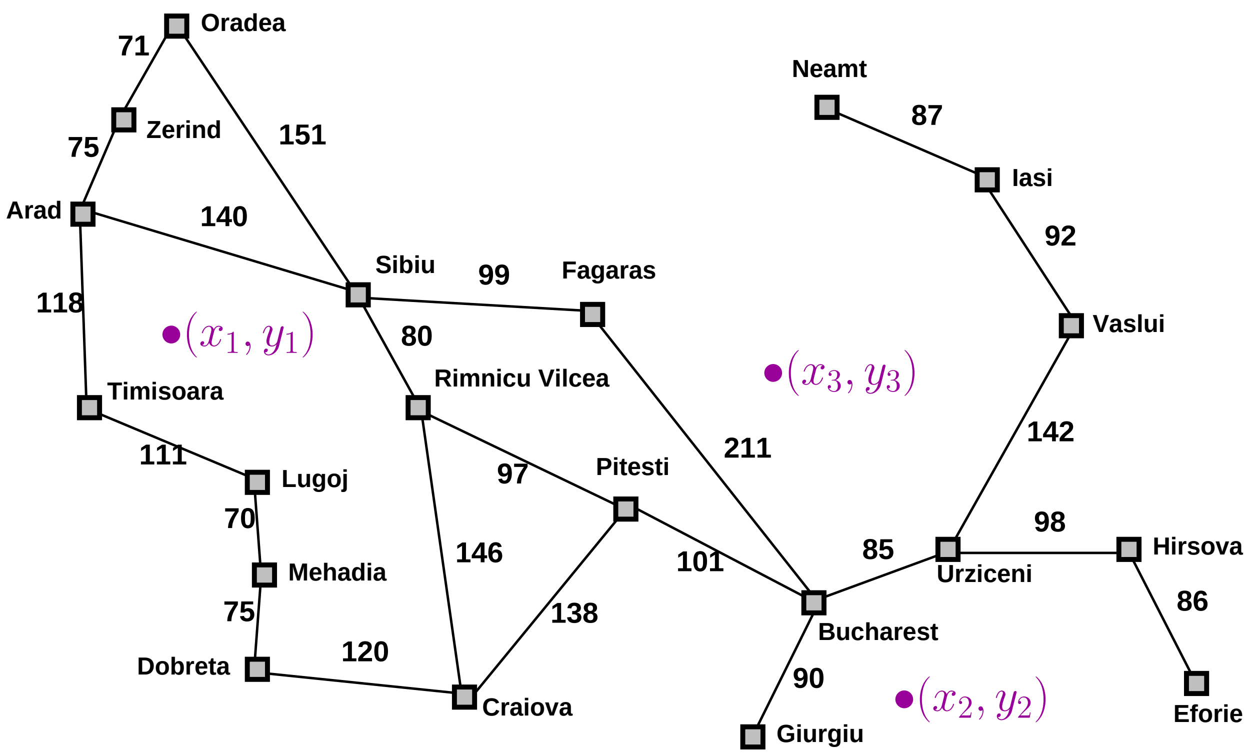


Genetic algorithms $\neq$ biological evolution
for example, real genes encode replication machinery

Hill-climbing in continuous state spaces

Suppose we want to put three airports in Romania – what locations?

- ♢ 6-D state space defined by (x_1, y_2), (x_2, y_2), (x_3, y_3)
- ♢ Objective function $f(x_1, y_2, x_2, y_2, x_3, y_3)$ measures desirability, e.g., sum of squared distances from each city to nearest airport



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

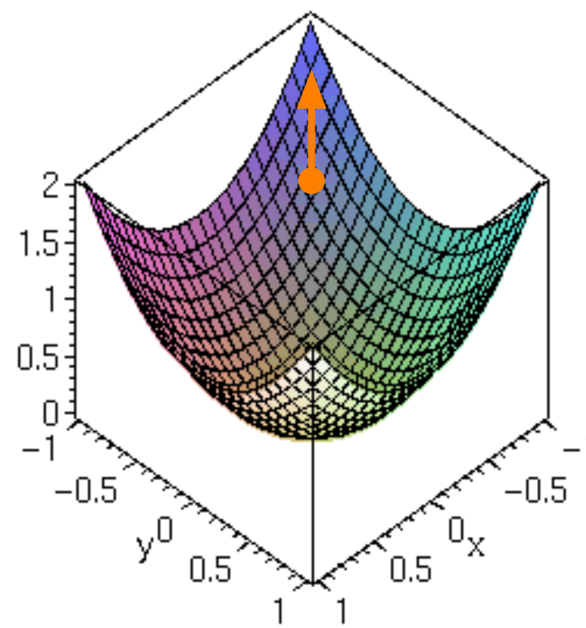
Hill-climbing in continuous state spaces

A technique from numerical analysis:

Given a surface $z = f(x, y)$, and a point (x, y), a **gradient** is a vector

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

The vector points in the direction of the steepest slope, and its length is proportional to the slope.



Gradient methods compute ∇f and use it to increase/reduce f,

e.g., by $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$

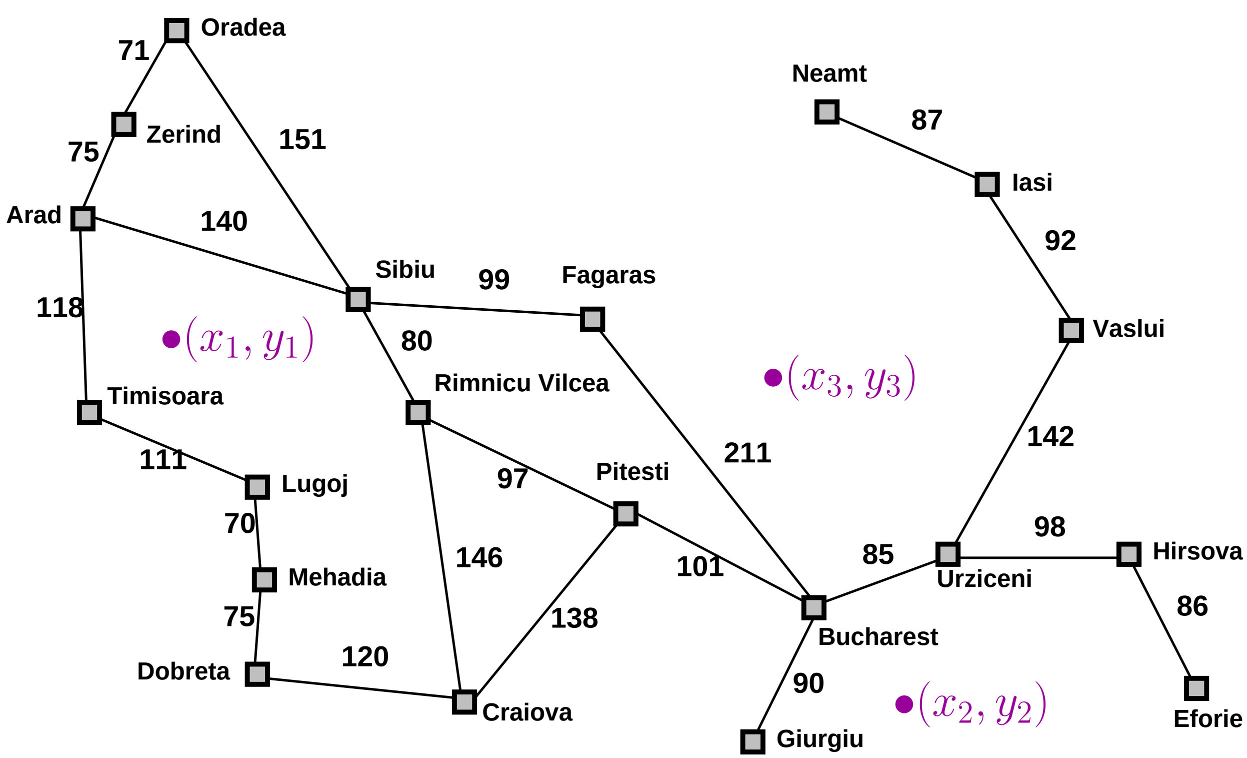
If $\nabla f = 0$ then you've reached a local maximum/minimum

Hill-climbing in continuous state spaces

Suppose we want to put three airports in Romania – what locations?

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

Look for $x_1, y_1, x_2, y_2, x_3, y_3$ such that $\nabla f(x_1, y_1, x_2, y_2, x_3, y_3) = 0$



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

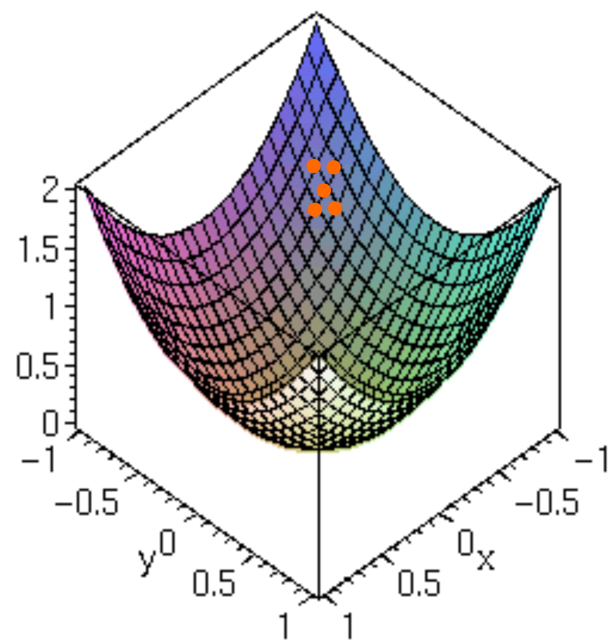
Continuous state spaces, continued

Sometimes can solve for $\nabla f(\mathbf{x}) = 0$ exactly (e.g., with one city)

Newton–Raphson (1664, 1690) iterates $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x})$ to solve $\nabla f(\mathbf{x}) = 0$, where $\mathbf{H}_{ij} = \partial^2 f / \partial x_i \partial x_j$

Discretization methods turn continuous space into discrete space

e.g., *empirical gradient* considers $\pm\delta$ change in each coordinate



Homework

Problems 4.1,
4.2,
4.9 (but you don't need to suggest a way to calculate it)
4.11,
4.12

10 points each, 50 points total

Due in one week