

Last update: September 11, 2008

LOCAL SEARCH ALGORITHMS

CMSC 421: CHAPTER 4, SECTIONS 3–4

Outline

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

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 **optimal** one, e.g., TSP

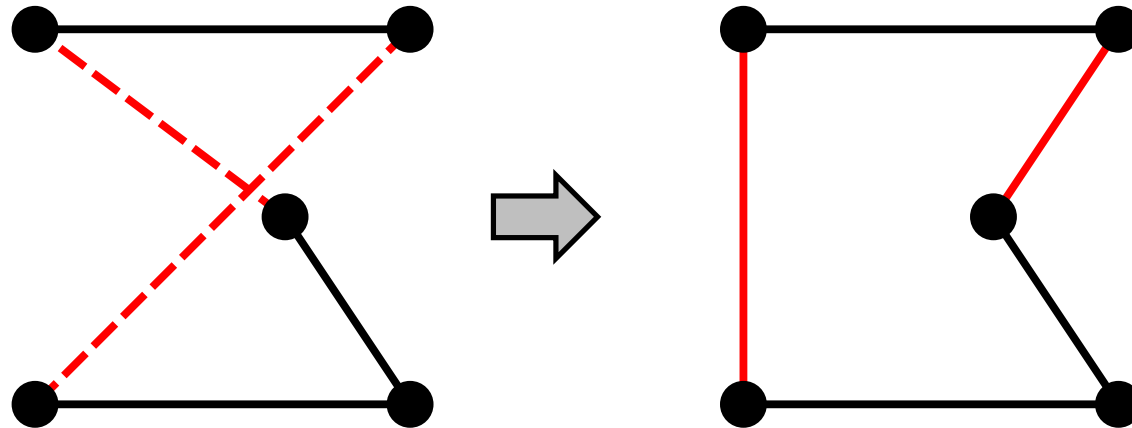
or, find one that satisfies constraints, e.g., timetable

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: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

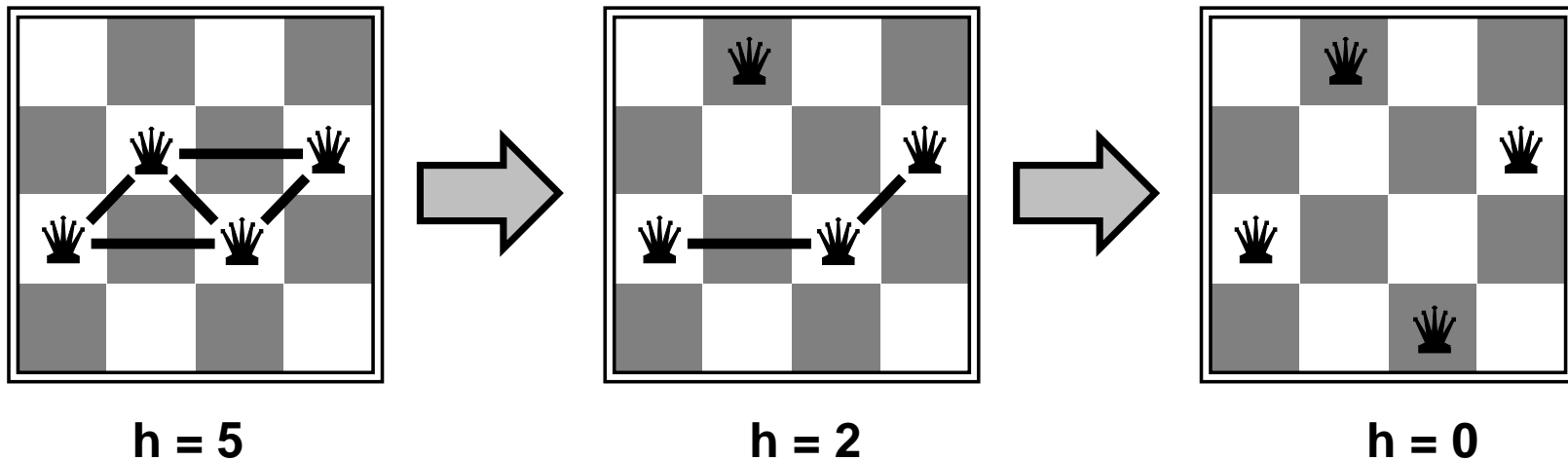


Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n -queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



Even for very large n (e.g., $n = 1$ million), usually solves n -queens problems almost instantly

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

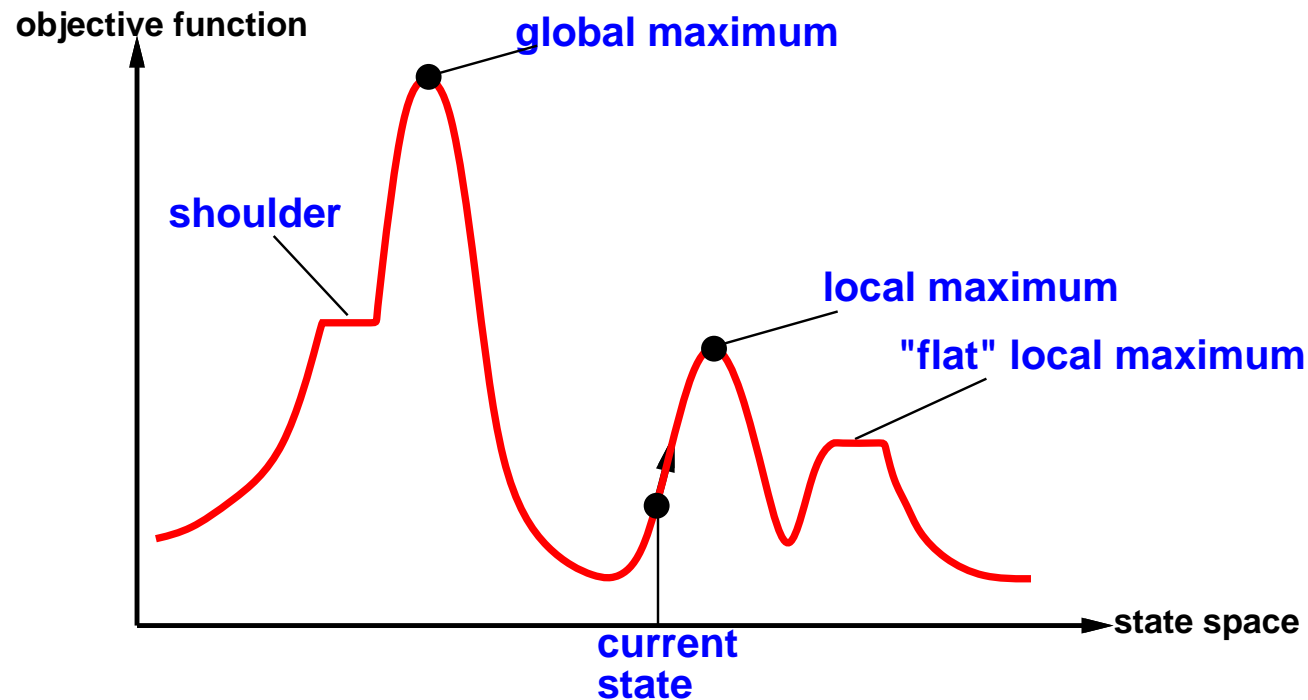
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
  end
```

At each step, move to a neighbor of higher value
in hopes of getting to an *optimal* solution (highest possible value)

Can easily modify this for problems where optimal means least possible value

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 $\lim_{\text{restarts} \rightarrow \infty} P(\text{complete}) = 1$

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
     $\Delta E$  ← VALUE[next] − VALUE[current]
    if  $\Delta E > 0$  then current ← next
    else with probability  $e^{\Delta E/T}$ , set current ← next
```

A very simple example

Each state x is a number in $[0, 1]$, and all states are neighbors

Want to find a state x that maximizes $\text{val}(x) = x^2$

Start with $x = 0$, and run for **iterations** number of time points

At each time point i , $T = \text{temp} * \text{factor}^i$

```
function anneal (temp,iterations)
  x = 0
  print x
  for i = 1 to iterations do
    temp = temp * 0.9
    xnew = random number between 0 and 1
    delta = xnew^2 - x^2
    if (delta > 0) or (random) <= e^(delta/temp))
      x = xnew
  print x
```

A very simple example

Results of calling `anneal(10,100)`. Read each column downward.

0.000000	0.987821	0.274338	0.969609	0.953317	0.953317
0.164227	0.987821	0.090741	0.969609	0.953317	0.953317
0.109036	0.987821	0.623421	0.969609	0.953317	0.954261
0.830933	0.231462	0.623421	0.969609	0.953317	0.954261
0.810696	0.827542	0.623421	0.969609	0.953317	0.954261
0.611756	0.203460	0.304669	0.953317	0.953317	0.954261
0.917702	0.906183	0.424640	0.953317	0.953317	0.954261
0.695078	0.906183	0.474800	0.953317	0.953317	0.954261
0.334728	0.985820	0.746596	0.953317	0.953317	0.954261
0.355373	0.985820	0.746596	0.953317	0.953317	0.954261
0.666291	0.985820	0.746596	0.953317	0.953317	0.954261
0.028830	0.985820	0.995854	0.953317	0.953317	0.954261
0.811972	0.985820	0.995854	0.953317	0.953317	0.954261
0.967231	0.942419	0.995854	0.953317	0.953317	0.954261
0.931109	0.942419	0.995854	0.953317	0.953317	0.954261
0.585659	0.041935	0.969609	0.953317	0.953317	0.954261
0.755799	0.343123	0.969609	0.953317	0.953317	

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, $\Pr[\text{reach } x^*]$ approaches 1

Is this necessarily an interesting guarantee?

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

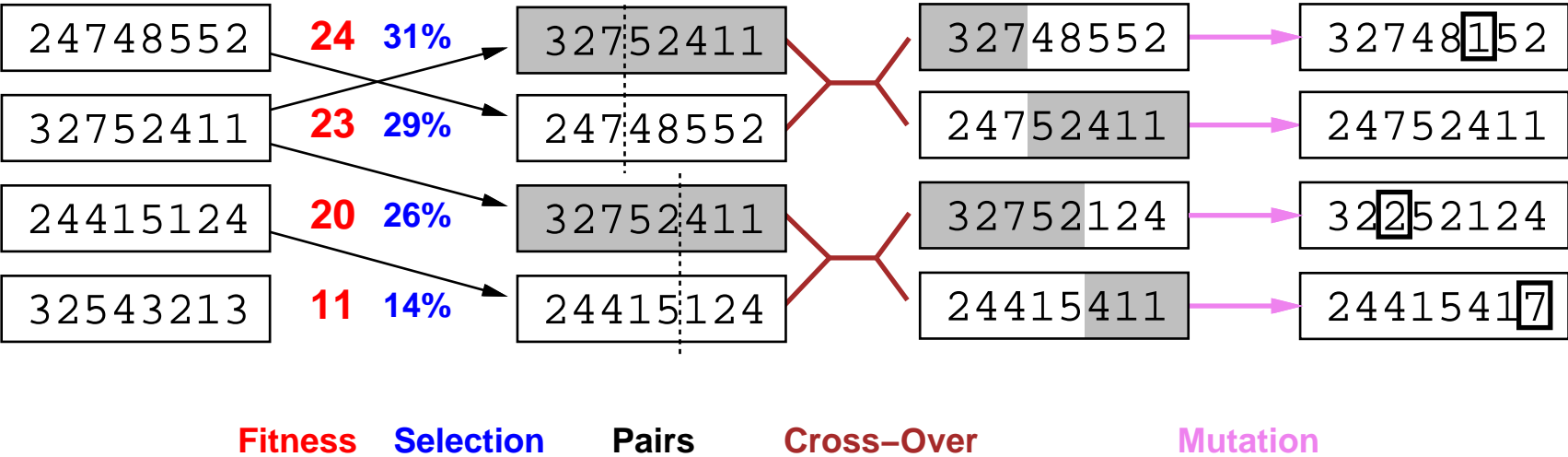
Stochastic beam search:

choose k successors randomly, biased towards good ones

Close analogy to natural selection

Genetic algorithms

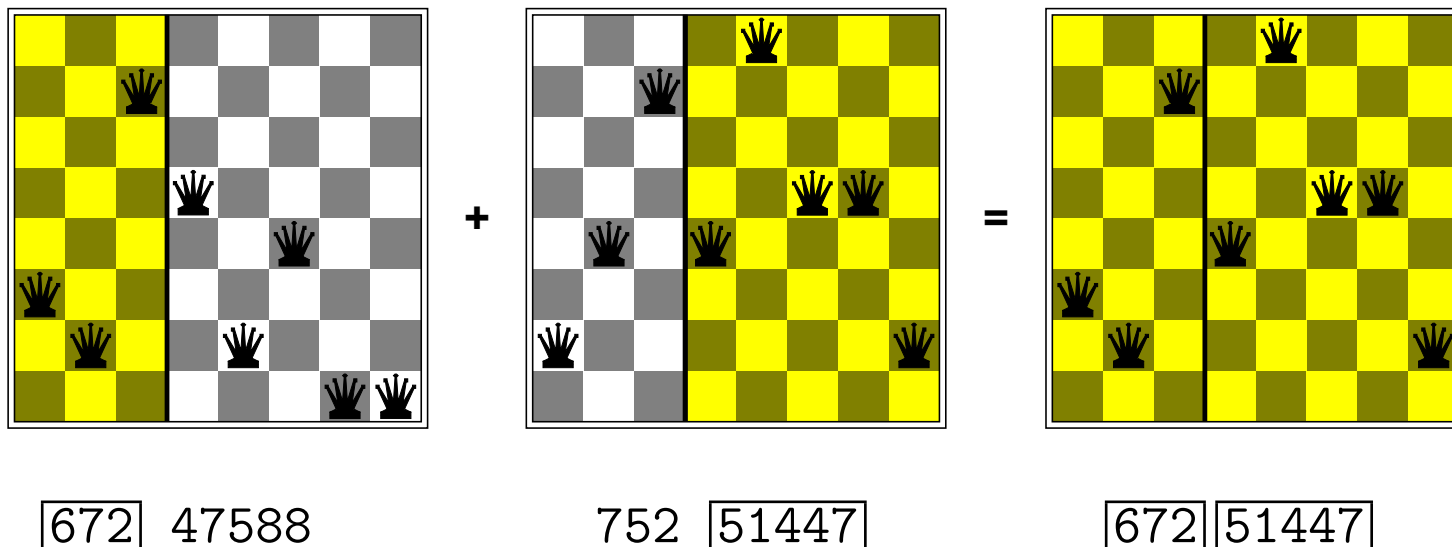
Genetic algorithms
 = stochastic local beam search + generate successors from **pairs** of states



Genetic algorithms contd.

GAs require states encoded as strings (GPs use programs)

Crossover helps **iff substrings are meaningful components**



GAs \neq evolution

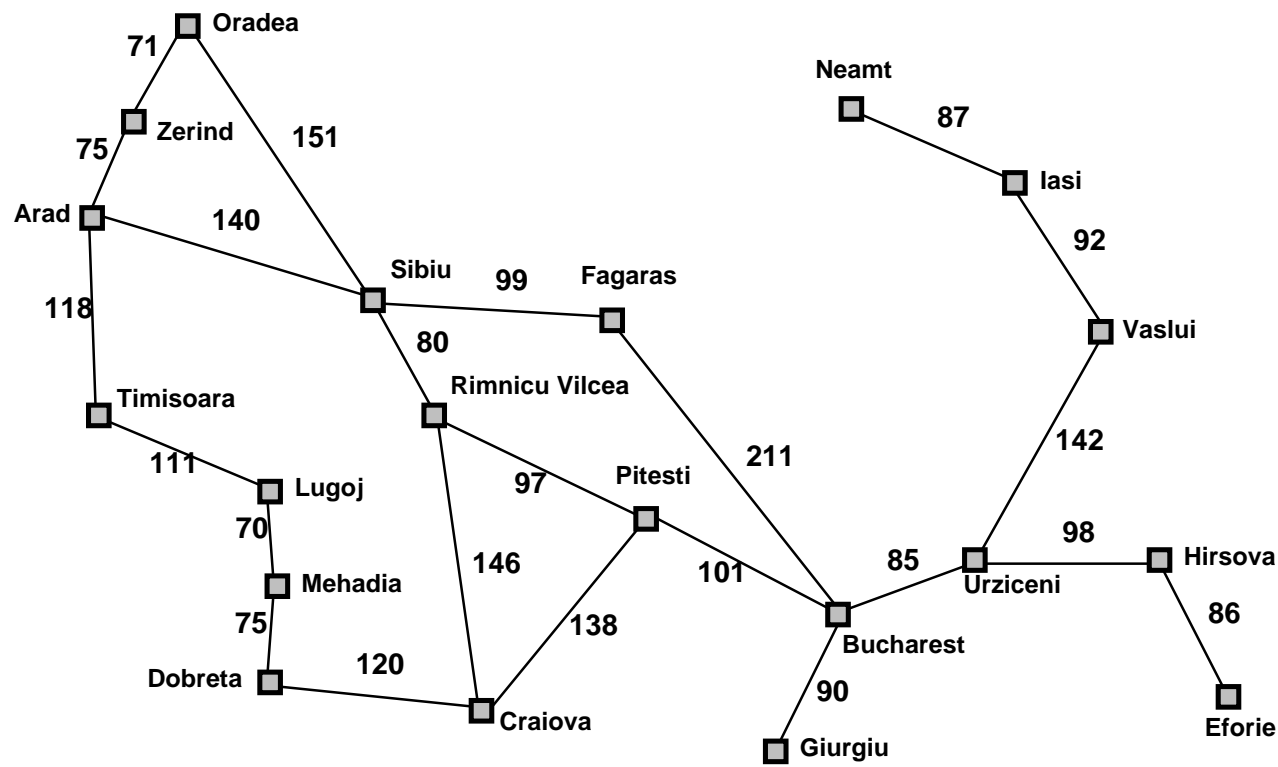
for example, real genes encode replication machinery

Hill-climbing in continuous state spaces

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

– 6-D state space defined by $(x_1, y_1), (x_2, y_2), (x_3, y_3)$

– objective function $f(x_1, y_1, x_2, y_2, x_3, y_3) =$
sum of squared distances from each city to nearest airport



City	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

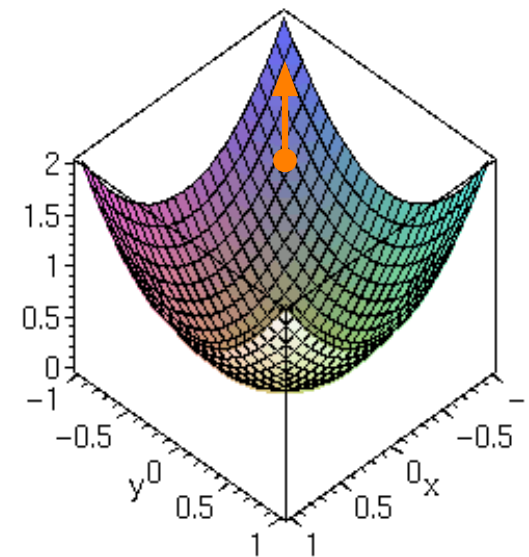
$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

that points in the direction of the steepest slope. The vector's length is proportional to the slope.

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

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

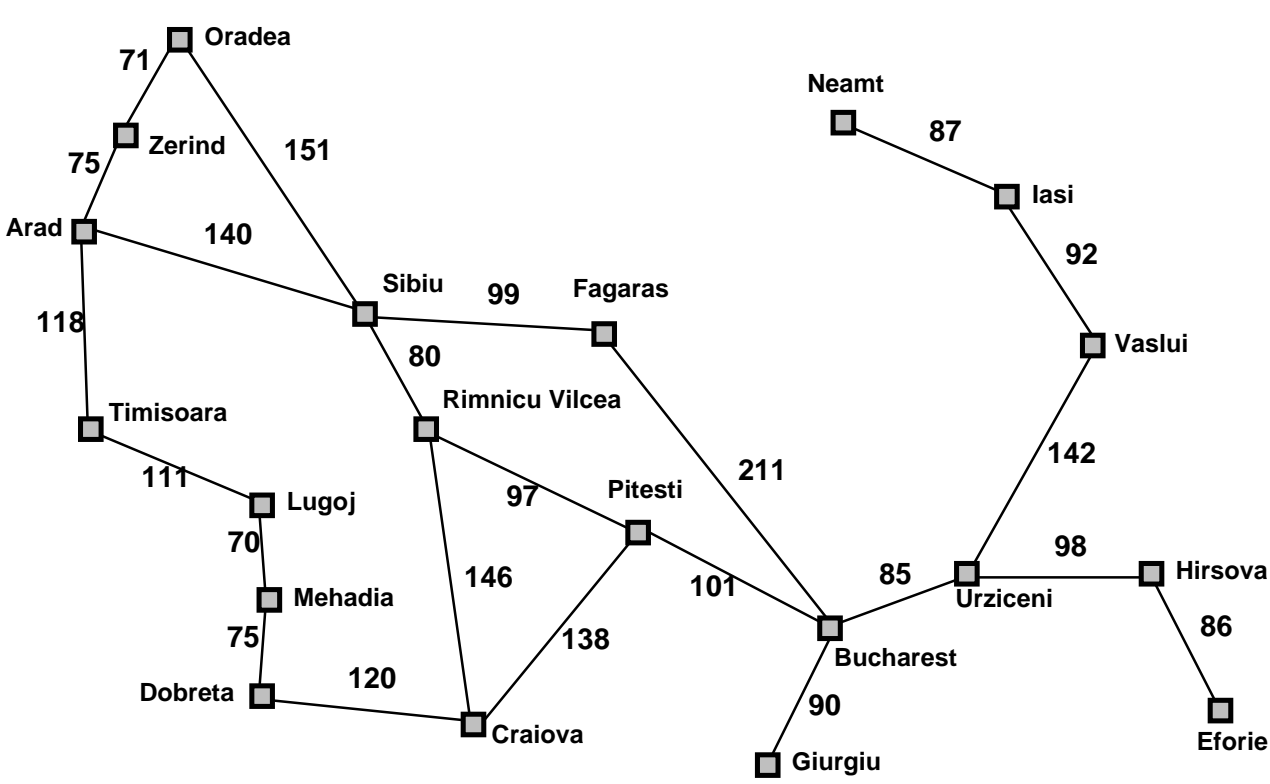
If $\nabla f = 0$ then you've reached a local maximum/minimum



Hill-climbing in continuous state spaces

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

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



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

