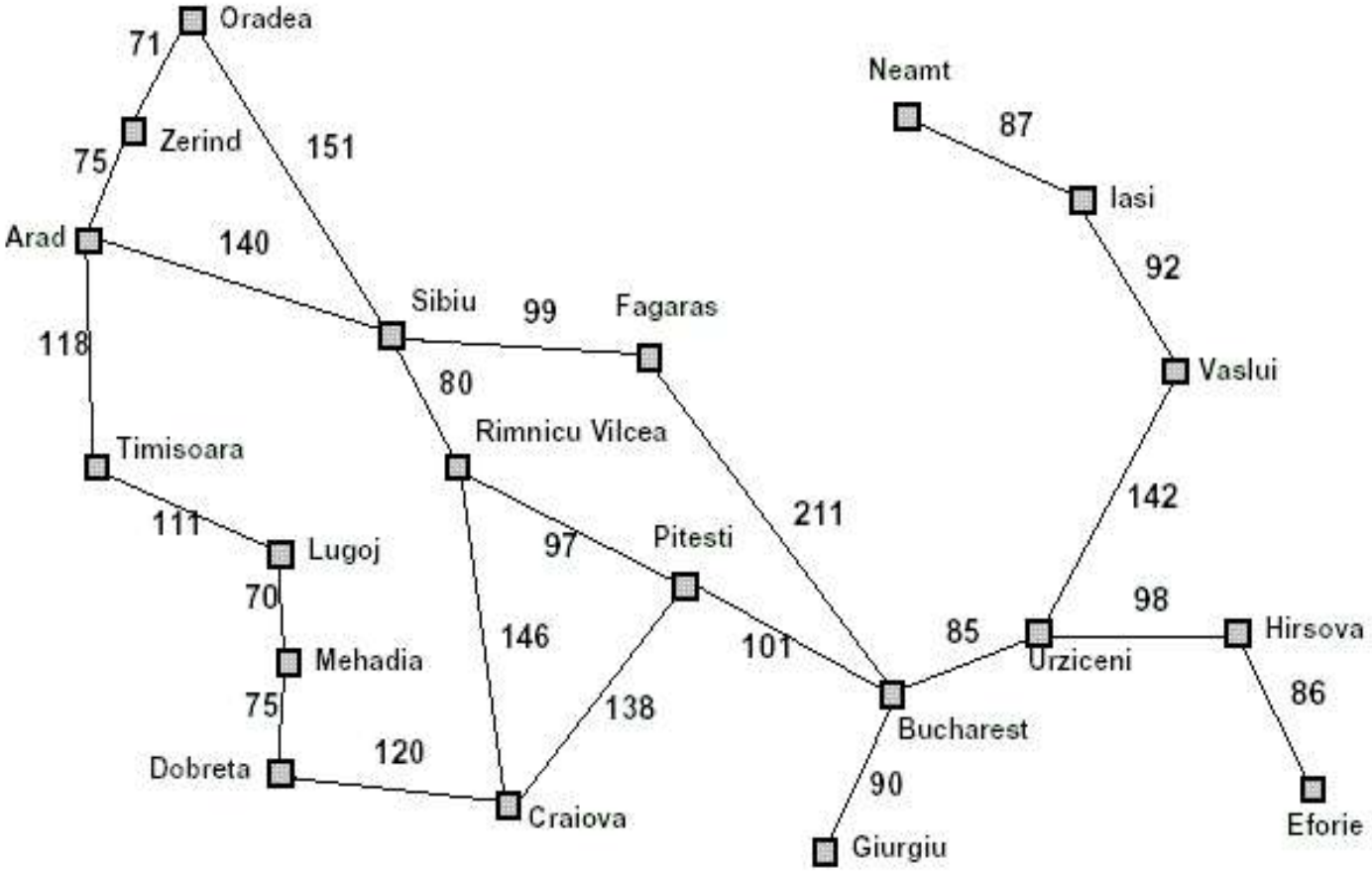
CSC421 Intro to Artificial Intelligence

UNIT 04:
Local Search

Review

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
  - Incomplete, not always optimal
- A* search expands lowest g + h
  - Complete and optimal
- Admissable heuristics can be derived from the exact solution of relaxed problems

Map with step costs and straight-line distances to goal



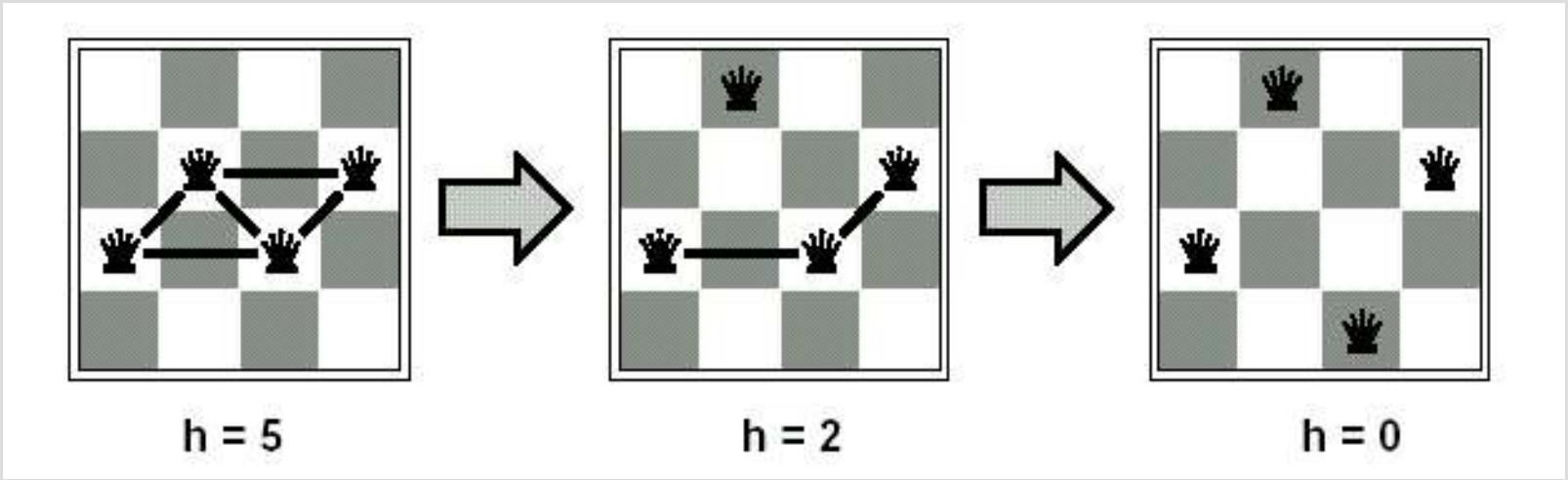
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Iterative improvement algorithms

- So far systematic exploration however …
- In many optimization problems, path is irrelevant; the goal state itself is the solution
- State space = set of "complete configurations"
  - TSP, Timetable, 8-queens
- Iterative improvement algorithms
  - Keep single current state, try to improve it
- Constant-space suitable for offline and online search

Example: n-queens

- Put n queens on an n by n board with no two queens on the same row, column, or diagonal
- Move a queen to reduce # of conflicts



Almost always solves n-queens problems almost instantaneously for very large n, e.g., n = 1 million

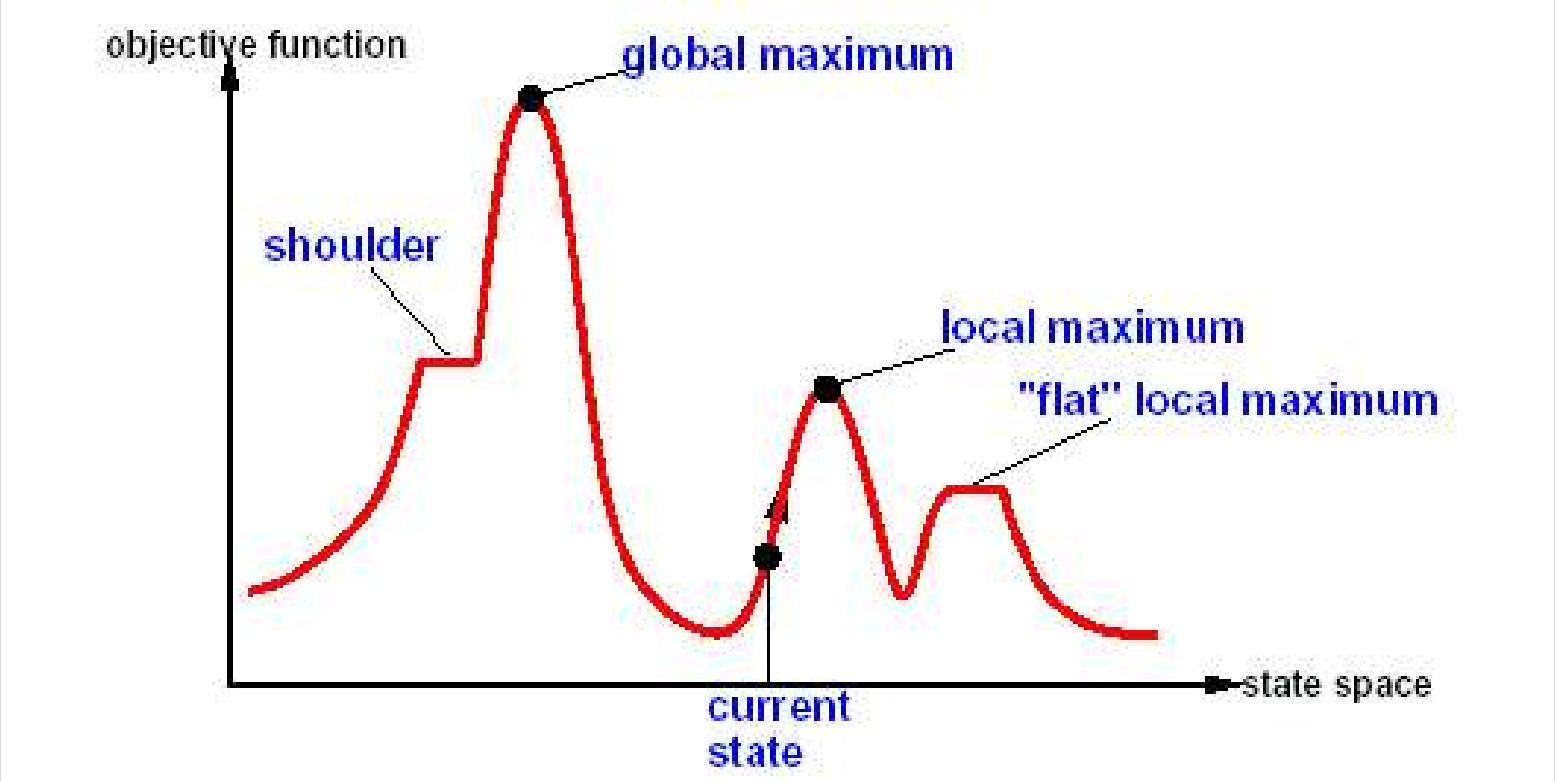
Hill climbing (or gradient descent)

- Like climbing mountain 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
```

Hill Climbing



Random restart hill climbing overcomes local maxima and is trivially complete
Random sideway moves – escape from shoulders, trap at “flats”

Simulated Annealing

Escape local maxima by allowing “bad moves” - decrease temperature (ammount of “moving” allowed”)

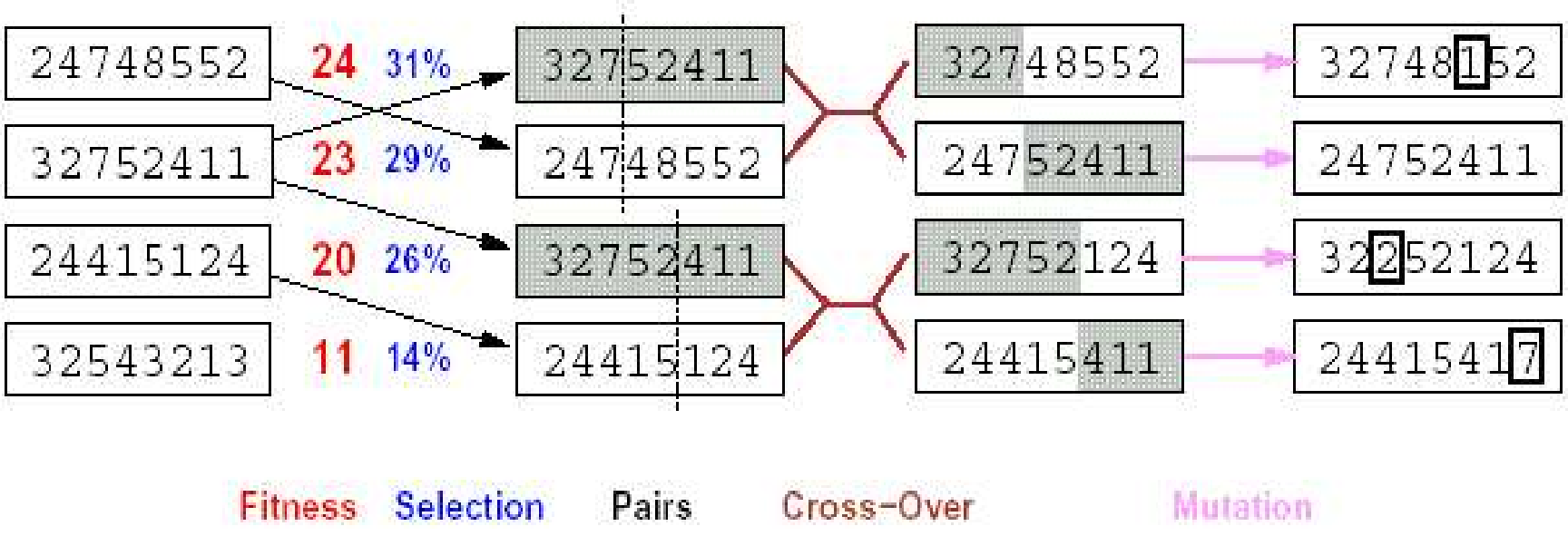
```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      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 current ← next only with probability e^(Δ E/T)
```

Local beam search

- Idea: keep k states instead of 1; choose top k of all their succesors
- Not the same as k searches run in parallel. Why ?
- Problem: quite often all k end up on same local hill
- Idea: choose k succesors randomly, biased toward good ones
- Observe the close analogy to natural selection

Genetic Algorithms

- Basically if stochastic beam search is asexual reproduction – genetic algorithms generate succesors from pairs of states



GAs continued

- GAs require states to be encoded as strings
- Cross-over helps iff substrings are meaningful component
  - From evolution: Good ears will still be good ears with a set of different legs
- GAs are not evolution (genes do not encode replication machinery)
- Main challenge to find representation
- Work well when
  - Good enough is ok
  - Few iterations can be afforded (for example user feedback)

Continuous State Spaces

- The “real” world
- Discretization
- Place 3 airports so that sum of sq. distances to all cities is minized

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

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

Follow the gradient

Typically numerically but sometimes:

$$\nabla f(\mathbf{x}) = 0$$

Constraint optimization harder: linear programming, quadratic programming

Online search problems

- Interleave computation & execution
- Exploration problems (robot placed on new planet go from A to B)
- Competitive ratio (can be infinite)
  - Actual cost compared to the cost of the path the agent would follow *if it "knew" the search space in advance*