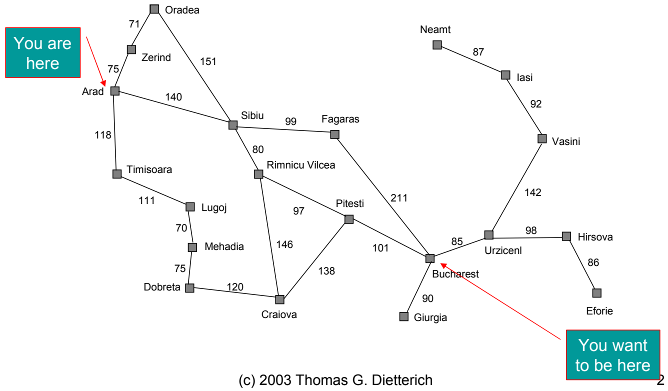
Search-Based Agents

- Appropriate in <u>Static Environments</u> where a model of the agent is known and the environment allows
  - prediction of the effects of actions
  - evaluation of goals or utilities of predicted states
- Environment can be partially-observable, stochastic, sequential, continuous, and even multi-agent, but it <u>must be static</u>!
- We will first study the deterministic, discrete, single-agent case.

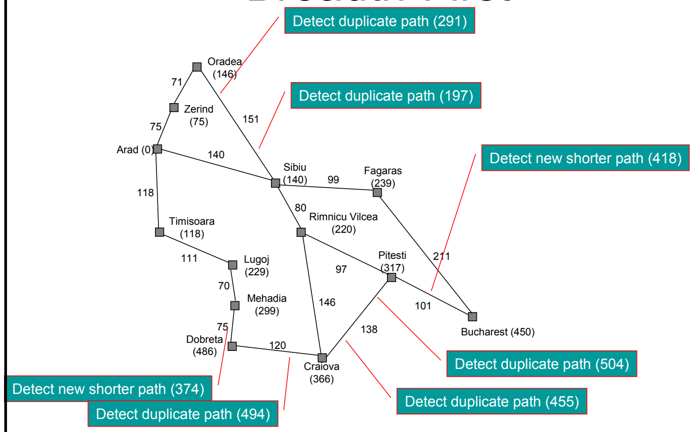
Computing Driving Directions

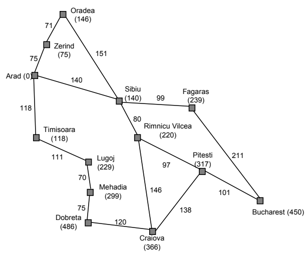


Search Algorithms

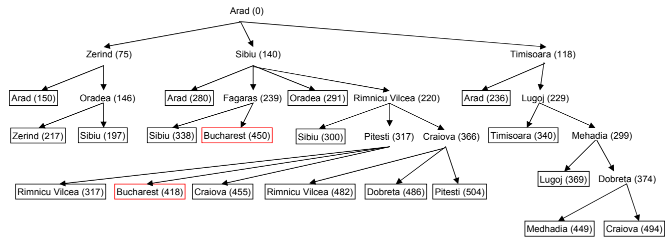
- Breadth-First
- Depth-First
- Uniform Cost
- A*
- Dijkstra's Algorithm

Breadth-First





Breadth-First



Formal Statement of Search Problems

- State Space: set of possible "mental" states
  - cities in Romania
- Initial State: state from which search begins
  - Arad
- Operators: simulated actions that take the agent from one mental state to another
  - traverse highway between two cities
- Goal Test:
  - Is current state Bucharest?

General Search Algorithm

```
function GENERAL-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
     if there are no candidates for expansion then return failure
     choose a leaf node for expansion according to strategy
     if the node contains a goal state then return the corresponding solution
     else expand the node and add the resulting nodes to the search tree
  end
```

- Strategy: first-in first-out queue (expand oldest leaf first)

Leaf Selection Strategies

- Breadth-First Search: oldest leaf (FIFO)
- Depth-First Search: youngest leaf (LIFO)
- Uniform Cost Search: cheapest leaf (Priority Queue)
- A* search: leaf with estimated shortest total path length $g(x) + h(x) = f(x)$
  - where $g(x)$ is length so far
  - and $h(x)$ is estimate of remaining length
  - (Priority Queue)

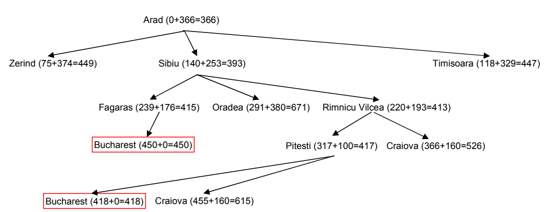
A* Search

- Let $h(x)$ be a “heuristic function” that gives an underestimate of the true distance between x and the goal state
  - Example: Euclidean distance
- Let $g(x)$ be the distance from the start to x, then $g(x) + h(x)$ is an <u>lower bound</u> on the length of the optimal path

Euclidean Distance Table

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

A* Search



All remaining leaves have $f(x) \geq 418$, so we know they cannot have shorter paths to Bucharest

Dijkstra’s Algorithm

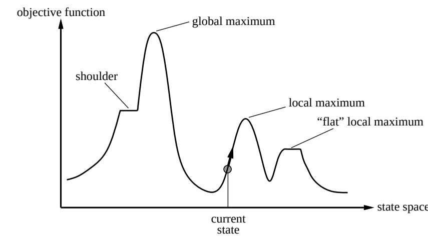
- Works backwards from the goal
- Each node keeps track of the shortest known path (and its length) to the goal
- Equivalent to uniform cost search starting at the goal
- No early stopping: finds shortest path from all nodes to the goal

Local Search Algorithms

- Keep a single current state *x*
- Repeat
  - Apply one or more operators to *x*
  - Evaluate the resulting states according to an Objective Function *J(x)*
  - Choose one of them to replace *x* (or decide not to replace *x* at all)
- Until time limit or stopping criterion

Hill Climbing

- Simple hill climbing: apply a randomly-chosen operator to the current state
- If resulting state is better, replace current state



- Steepest-Ascent Hill Climbing:
- Apply all operators to current state, keep state with the best value
- Stop when no successors state is better than current state

Gradient Ascent

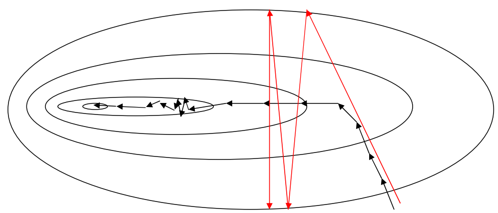
- In continuous state spaces, $x = (x_1, x_2, \ldots, x_n)$ is a vector of real values
- Continuous operator: $x := x + \Delta x$ for any arbitrary vector Δx (infinitely many operators!)
- Suppose J(*x*) is differentiable. Then we can compute the direction of steepest increase of J by the first derivative with respect to *x*, the gradient:

$$\nabla_x J(x) = \left(\frac{\partial J}{\partial x_1}, \frac{\partial J}{\partial x_2}, \cdots, \frac{\partial J}{\partial x_n} \right)$$

Gradient Descent Search

- Repeat
  - Compute Gradient ∇J
  - Update $x := x + \eta \nabla J$
- Until $\nabla J \approx 0$
- η is the "step size", and it must be chosen carefully
- Methods such as conjugate gradient and Newton's method choose η automatically

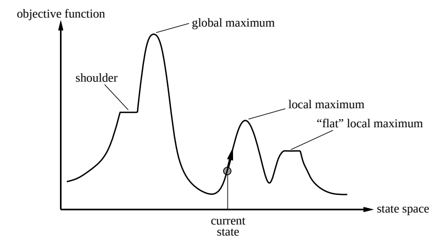
Visualizing Gradient Ascent



If η is too large, search may overshoot and miss the maximum or oscillate forever

Problems with Hill Climbing

- Local optima
- Flat regions
- Random restarts can give good results



Simulated Annealing

- T = 100 (or some large value)
- Repeat
  - Apply randomly-chosen operator to x to obtain x'.
  - Let $\Delta E = J(x') - J(x)$
  - If $\Delta E > 0$, switch to x'
  - Else switch to x' with probability
    - exp [$\Delta E/T$] (large negative steps are less likely)
    - T := 0.99 * T ("cool" T)
- Slowly decrease T ("anneal") to zero
- Stop when no changes have been accepted for many moves
- Idea: Accept "down hill" steps with some probability to help escape from local minima