Introduction to Artificial Intelligence
Problem Solving through Search
(Goal-based agents)
BFS, DFS, Greedy Search

Janyl Jumadinova
September 12-16, 2016
Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

**Formulate goal:** be in Bucharest

**Formulate problem:**

**states:** various cities

**actions:** drive between cities

**Find solution:** sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
Example: Romania
Tree search algorithms

Basic idea:
offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. *expanding* states)

```plaintext
function TREE-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
```
Tree search example
Tree search example
Tree search example
Implementation: states vs. nodes

A state is a (representation of) a physical configuration.
A node is a data structure constituting part of a search tree
includes parent, children, depth, path cost $g(x)$
States do not have parents, children, depth, or path cost!
Breadth-first search

Expand shallowest unexpanded node

- FIFO queue, i.e., new successors go at end
Breadth-first search

Expand shallowest unexpanded node
  ▶ FIFO queue, i.e., new successors go at end

```
A
B C
D E F G
```
Breadth-first search

Expand shallowest unexpanded node

- FIFO queue, i.e., new successors go at end
Breadth-first search

Expand shallowest unexpanded node

- FIFO queue, i.e., new successors go at end
Breadth-first search
Breadth-first search

Zerind  Sibiu  Timisoara
Breadth-first search
Breadth-first search

Arad
Oradea
Rimnicu
Vilcea
Fagaras
Arad
Lugoj
Arad
Oradea
Zerind
Sibiu
Timisoara
Arad
### Breadth-first search

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>110</td>
<td>.11 milliseconds</td>
<td>107 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11,110</td>
<td>11 milliseconds</td>
<td>10.6 megabytes</td>
</tr>
<tr>
<td>6</td>
<td>$10^6$</td>
<td>1.1 seconds</td>
<td>1 gigabyte</td>
</tr>
<tr>
<td>8</td>
<td>$10^8$</td>
<td>2 minutes</td>
<td>103 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>$10^{10}$</td>
<td>3 hours</td>
<td>10 terabytes</td>
</tr>
<tr>
<td>12</td>
<td>$10^{12}$</td>
<td>13 days</td>
<td>1 petabyte</td>
</tr>
<tr>
<td>14</td>
<td>$10^{14}$</td>
<td>3.5 years</td>
<td>99 petabytes</td>
</tr>
<tr>
<td>16</td>
<td>$10^{16}$</td>
<td>350 years</td>
<td>10 exabytes</td>
</tr>
</tbody>
</table>
Depth-first search

Expand deepest unexpanded node

- LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node
  ▶ LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node
  ▶ LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

- LIFO queue, i.e., put successors at front
Best-first search

**Idea**: use an *evaluation function* for each node – estimate of “desirability”
Best-first search

**Idea:** use an *evaluation function* for each node – estimate of “desirability”

⇒ Expand most desirable unexpanded node
**Best-first search**

**Idea:** use an *evaluation function* for each node – estimate of “desirability”

⇒ Expand most desirable unexpanded node

**Implementation:**
a queue sorted in decreasing order of desirability
**Best-first search**

**Idea:** use an evaluation function for each node – estimate of “desirability”

⇒ Expand most desirable unexpanded node

**Implementation:**
a queue sorted in decreasing order of desirability

**Special cases:** greedy search, $A^*$ search
Romania with step costs in km

<table>
<thead>
<tr>
<th>City</th>
<th>Step Costs to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>71</td>
</tr>
<tr>
<td>Ursiceni</td>
<td>151</td>
</tr>
<tr>
<td>Hirsova</td>
<td>140</td>
</tr>
<tr>
<td>Eforie</td>
<td>160</td>
</tr>
<tr>
<td>Neamt</td>
<td>111</td>
</tr>
<tr>
<td>Oradea</td>
<td>120</td>
</tr>
<tr>
<td>Zerind</td>
<td>118</td>
</tr>
<tr>
<td>Arad</td>
<td>140</td>
</tr>
<tr>
<td>Timisoara</td>
<td>146</td>
</tr>
<tr>
<td>Sibiu</td>
<td>144</td>
</tr>
<tr>
<td>Fagaras</td>
<td>146</td>
</tr>
<tr>
<td>Neamt</td>
<td>87</td>
</tr>
<tr>
<td>Iasi</td>
<td>92</td>
</tr>
<tr>
<td>Vaslui</td>
<td>142</td>
</tr>
<tr>
<td>Pitesti</td>
<td>211</td>
</tr>
<tr>
<td>RimniculVilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>101</td>
</tr>
<tr>
<td>Timisoara</td>
<td>85</td>
</tr>
<tr>
<td>Urziceni</td>
<td>98</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>86</td>
</tr>
<tr>
<td>Eforie</td>
<td>90</td>
</tr>
</tbody>
</table>

Straight-line distance to Bucharest

- **Arad**: 366
- **Bucharest**: 0
- **Craiova**: 160
- **Dobret**: 242
- **Eforie**: 161
- **Fagaras**: 178
- **Giurgiu**: 77
- **Hirsova**: 151
- **Iasi**: 226
- **Lugoj**: 244
- **Mehadia**: 241
- **Neamt**: 234
- **Oradea**: 380
- **Pitesti**: 98
- **RimniculVilcea**: 193
- **Sibiu**: 253
- **Timisoara**: 329
- **Urziceni**: 80
- **Vaslui**: 199
- **Zerind**: 374
Greedy search

Evaluation function $h(n)$ (heuristic)

= estimate of cost from $n$ to the closest goal

Example: $h(n) =$ straight-line distance from $n$ to Bucharest
Greedy search

Evaluation function $h(n)$ (heuristic)

$= \text{estimate of cost from } n \text{ to the closest goal}$

Example: $h(n) = \text{straight-line distance from } n \text{ to Bucharest}$

- Greedy search expands the node that appears to be closest to goal
Greedy Search
Greedy Search

Diagram:

- Arad
  - Sibiu 253
  - Timisoara 329
  - Zerind 374
Greedy Search

Albania

Kosovo

Macedonia

North Macedonia
Greedy Search
Greedy search

- Can get stuck in loops, e.g.,
  Iasi → Neamt → Iasi → Neamt →
  Complete in finite space with repeated-state checking
- A good heuristic can give dramatic improvement in search cost
http://aispace.org/search/

Explore Best first search (explanation from Alspace: http://www.aispace.org/deduction/help/tut3.shtml)
A* search

Idea:
avoid expanding paths that are already expensive
A* search

Idea:

Avoid expanding paths that are already expensive

- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach $n$
- $h(n)$ = estimated cost to goal from $n$
- $f(n)$ = estimated total cost of path through $n$ to goal

Romania with step costs
Example: Romania
A* Search

Arad
366=0+366
A* Search

Sibiu
393=140+253

Timisoara
447=118+329

Zerind
449=75+374
A* Search

- Arad
- Fagaras
- Oradea
- Sibiu
- Timisoara
- Rimnicu Vilcea
- Zerind

Distances:

- Arad to Sibiu: 646 = 280 + 366
- Arad to Oradea: 415 = 239 + 176
- Arad to Timisoara: 447 = 118 + 329
- Zerind to Arad: 449 = 75 + 374
- Arad to Fagaras: 671 = 291 + 380
- Arad to Rimnicu Vilcea: 413 = 220 + 193

Graph representation:

- Arad is the starting point.
- Sibiu, Timisoara, Zerind are destinations.
- Distances are calculated using the formula: distance = path cost.
A* Search

Diagram showing the A* search algorithm with distances between cities.

- Arad
- Fagaras
- Oradea
- Rimnicu Vilcea
- Craiova
- Pitesti
- Sibiu
- Timisoara
- Zerind

Distances:
- 447 = 118 + 329
- 449 = 75 + 374
- 646 = 280 + 366
- 415 = 239 + 176
- 526 = 366 + 160
- 553 = 300 + 253
- 671 = 291 + 380

36/38
A* Search

```
A* Search

Arad
Sibiu
Timisoara
Zerind

Arad
Fagaras
Oradea
Rimnicu Vilcea
Sibiu
Bucharest
Craiova
Pitesti
Sibiu

447=118+329
646=280+366
591=338+253
450=450+0
526=366+160
553=300+253
417=317+100
671=291+380
```
A* Search

Arad
- Sibiu: 646 = 280 + 366
- Fagaras
- Oradea: 671 = 291 + 380
- Rimnicu Vilcea

Sibiu
- Arad: 591 = 338 + 253
- Bucharest: 450 = 450 + 0

Arad
- Timisoara: 447 = 118 + 329

Timisoara
- Bucharest: 591 = 338 + 253
- Pitesti: 526 = 366 + 160

Pitesti
- Sibiu: 553 = 300 + 253
- Craiova: 526 = 366 + 160

Craiova
- Bucharest: 615 = 455 + 160
- Fagaras

Fagaras

Zerind
- Sibiu: 449 = 75 + 374

Zerind