Solving Problems using Search

Artificial Intelligence @ Allegheny College

Janyl Jumadinova

September 11, 2018
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).

```
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
(Search) Algorithm Evaluation

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?
<table>
<thead>
<tr>
<th>Name</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Yes*</td>
<td>Yes*</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>DFS</td>
<td>No</td>
<td>No</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>Greedy best-first</td>
<td>No</td>
<td>No</td>
<td>$O(b^m)$*</td>
<td>$O(b^m)$</td>
</tr>
<tr>
<td>A*</td>
<td>Yes*</td>
<td>Yes*</td>
<td>exp.</td>
<td>exp.</td>
</tr>
</tbody>
</table>

$b$ is the branching factor/maximum number of successors of any node, $d$ is the depth of the shallowest solution, $m$ is the maximum depth of the tree * indicates that there is a special case where this may not be true.
Example: Romania

Can use Tree Search Algorithms (BFS, DFS - Uninformed or Blind Search).

Special cases: greedy search, $A^*$ search (Informed or Heuristic Search).
Romania with step costs in km

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy search

Evaluation function $f(n) = h(n)$ (heuristic)
= estimate of cost from $n$ to the closest goal

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

Evaluation function \( f(n) = h(n) \) (heuristic)
\( = \) estimate of cost from \( n \) to the closest goal

Example: \( h(n) = \) straight-line distance from \( n \) to Bucharest.

- Greedy search expands the node that \textit{appears} to be closest to goal.
Greedy best-first search
Greedy best-first search

Search Example
Greedy best-first search
Greedy best-first search
Greedy search

- Can get stuck in loops, e.g. Iasi → Fagaras, Iasi → Neamt → Iasi → Neamt →
- Complete in finite space with repeated-state checking.
- A good heuristic can give dramatic improvement in search cost.
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
A* Search

Arad
366 = 0 + 366
A* Search

Solving Problems using Search
A* Search

{diagram showing cities and distances}

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

 Distances:
- Arad to Fagaras: 646 = 280 + 366
- Arad to Oradea: 671 = 291 + 380
- Arad to Rimnicu Vilcea: 447 = 118 + 329
- Zerind to Bucharest: 449 = 75 + 374
- Bucharest to Craiova: 615 = 414 + 193
- Craiova to Oradea: 526 = 366 + 160
- Oradea to Sibiu: 591 = 338 + 253
- Pitesti to Sibiu: 553 = 300 + 253
- Sibiu to Bucharest: 646 = 280 + 366
- Sibiu to Craiova: 615 = 414 + 193
- Sibiu to Oradea: 591 = 338 + 253
So far, addressed a single category of problems: observable, deterministic, known environments, solution is a sequence of actions.
Local Search

- So far, addressed a single category of problems: observable, deterministic, known environments, solution is a sequence of actions.
- Now, relax one of these assumptions: evaluate and modify one or more current states.
Local Search

- So far, addressed a single category of problems: observable, deterministic, known environments, solution is a sequence of actions.
- Now, relax one of these assumptions: evaluate and modify one or more current states.

Local search algorithms
- operate using a single current node (rather than multiple paths).
- generally move only to neighbors of that node.
Local Search

Benefits

1. use very little memory (usually a constant amount)
2. can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable
3. useful for solving pure optimization problems, where the aim is to find the best state according to an objective function
Local Search

State-space landscape: elevation is the objective/heuristic function, the goal is to find the global maximum.
Hill-Climbing

Or gradient ascent/descent
Also known as “Like climbing Everest in thick fog with amnesia”
Hill-Climbing

- A loop that continually moves in the direction of increasing value that is, uphill.
- It terminates when it reaches a “peak” where no neighbor has a higher value.
Hill-Climbing

- A loop that continually moves in the direction of increasing value that is, uphill.
- It terminates when it reaches a “peak” where no neighbor has a higher value.
- The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.
Hill-Climbing

- A loop that continually moves in the direction of increasing value that is, uphill.
- It terminates when it reaches a “peak” where no neighbor has a higher value.
- The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.
- Hill climbing does not look ahead beyond the immediate neighbors of the current state.
Hill-Climbing

\begin{center}
\begin{verbatim}
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
\end{verbatim}
\end{center}
Implementing Hill-Climbing: Environment Setup

- Download the program from the “class_materials” repository on GitHub.
- It has an implementation of the background.
- It also has definitions for the setup-patches and setup-turtles procedures:
Next, we need to define and display the highest and lowest points in our terrain:

```plaintext
globals [highest ;; the highest patch elevation
           lowest] ;; the lowest patch elevation

- Let’s set up two monitors in the Interface tab with the Toolbar (highest and lowest)
```
Modify the setup-patches procedure:

to setup-patches
  ask patches
    [ set elevation (random 10000) ]
diffuse elevation 1
  ask patches
    [ set pcolor scale-color green elevation 1000 9000 ]
set highest max [elevation] of patches
set lowest min [elevation] of patches
  ask patches [  
    if (elevation > (highest - 100))  
      [set pcolor white]
    if (elevation < (lowest + 100))  
      [set pcolor black] ]
end
Hill-climbing

1. The turtles cannot see ahead farther than just one patch
2. Each turtle can move only one square each turn
3. Turtles are blissfully ignorant of each other

;;; each turtle goes to the highest elev-n in a radius of 1 to move-to-local-max
ask turtles [
  uphill elevation
  if ( [elevation] of patch-ahead 1 > elevation )
  [ fd 1 ]
]
end
Hill-climbing

- Every patch picked a random elevation, and then we diffused these values one time
- This doesn’t provide a continuous spread of elevation across the graphics window
- So, we diffuse more!

```
repeat 5 [ diffuse elevation 1 ]
```
Hill-climbing

Let’s plot the number of turtles who have reached the ’peak-zone’ (within 1% of the highest elevation)

to do-plots
  set-current-plot "Turtles at Peaks"
  plot count turtles with
    [ elevation >= (highest - 100) ]
end

- Create a slider for the number of turtles and replace the hard-coded value with it
Hill-climbing

We may want to stop the model after all the turtles have found their local maxima
- Add turtles-moved? variable to global variables
- At the end of the go procedure, add a test to see if any turtles have moved

to go
    set turtles-moved? false
    move-to-local-max
    do-plots
    if (not turtles-moved?)
        [ stop ]
end

- In move-to-local-max if a turtle moves, set turtles-moved? to True
# Other Search Algorithms

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uninformed Search</td>
<td>Uniform-Cost</td>
</tr>
<tr>
<td>Uninformed Search</td>
<td>Depth-limited</td>
</tr>
<tr>
<td>Uninformed Search</td>
<td>Iterative Deepening DFS</td>
</tr>
<tr>
<td>Uninformed Search</td>
<td>Bidirectional</td>
</tr>
<tr>
<td>Informed Search</td>
<td>Iterative-deepening A*</td>
</tr>
<tr>
<td>Informed Search</td>
<td>Recursive best-first search</td>
</tr>
<tr>
<td>Informed Search</td>
<td>Simplified memory-bounded A*</td>
</tr>
<tr>
<td>Local Search</td>
<td>Simulated annealing</td>
</tr>
<tr>
<td>Local Search</td>
<td>Local beam search</td>
</tr>
<tr>
<td>Local Search</td>
<td>Genetic algorithm</td>
</tr>
</tbody>
</table>