Introduction to Artificial Intelligence
Local Search

Janyl Jumadinova
September 19, 2016
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?
### Search Summary

<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
Local Search

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

- **objective function**
- **global maximum**
- **shoulder**
- **local maximum**
- "flat" local maximum
- **current state**
- **state space**
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.
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
Implementing Hill-Climbing: Environment Setup

Let’s start with a model we built a week ago (on September 12)

First we will create a background by re-writing the set-up procedure:

patches-own [elevation]

to setup
  ca
  setup-patches
  setup-turtles
  reset-ticks
end
We need to define setup-patches and setup-turtles procedures:

to setup-patches
    ask patches
        [ set elevation (random 10000) ]
    diffuse elevation 1
    ask patches
        [ set pcolor scale-color green elevation 1000 9000 ]
end
to setup-turtles
    crt 100
    ask turtles [ if (shade-of? green color)
        [ set color red ]
        setxy random-xcor random-ycor ]
end
We need to define and display the highest and lowest points in our terrain:

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

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