Recitations and office hours

- **Office hours TAs**: Tuesday 4:30-5:30 in Annenberg 213
- **Office hours Instructor**: Wednesday 4-5 in Annenberg 300
- **Recitations**: Thursday 4:30-5:30 in Annenberg 107
  - Optional but encouraged
  - Will review material from class
  - Discuss projects in more detail
Challenge projects

- Two projects
  - “Crowdsourcing science”
  - “AI in games”
- Details given in recitation on Thursday, and posted on the website
- Implementation in Python
- Milestone after ~4 weeks
  - Simpler version of the challenge
  - Feedback on implementation
- Final challenge
  - Compare algorithms in a competition
“Crowdsourcing science”

- Recruit population to help map out rare bird species
- Existing field guides cumbersome to use
- Create “intelligent field guide” that lets people identify bird species by “asking the right questions”

Welinder et al ‘10
Intelligent field guide

- **Project goal**
  - Use AI techniques to adaptively ask questions to identify bird species

- **Data / Input**
  - Data set containing answers to questions about 200 birds

- **Performance measure: accuracy; # questions used**
  - Milestone: Only correct answers
  - Final challenge: “Noisy” answers; hold out test set; allowed to use images..
PAC-MAN vs. PAC-MAN

- Game description
  - “Symmetric” version of PAC-MAN
  - Who eats most “pac dots” wins
  - Eating a “power pellet” turns PAC-MAN into a ghost

- Project goal
  - Develop AI to win the eating competition

- Data / Input
  - Description of maze (graph)
  - Location of “pac dots” and “power pellets”

- Performance measure: # pac dots eaten
  - Milestone: No power pellets; known maze
  - Final challenge: power pellets; motion noise; new maze
Independent projects

- For students who do AI-related research
- Need to come up with your own project
- Must be “something new” for this class
- Groups of 2-3 students
- Will be advised by TAs and instructor
- Need to submit
  - Project proposal
  - Milestone report
  - Final report
- Most students expected to choose one of the two challenge projects
Project timeline

- By next Monday (Oct 11)
  - Form teams of 3 students
  - Indicate preference for projects
- Milestone implementation due: November 3
- Final implementation due: December 1
“Get from state A to B as quickly as possible”

A fundamental problem in many AI problems
  - Navigation, VLSI layout, resource allocation, planning, ...

For now assume
  - Fully observable environment and deterministic actions
Search with goal based agents

- Agent has
  - model of environment (map, puzzle rules, mechanics,...)
    How will the environment change if I do X?
  - Goal checker:
    Declares some environment states as goals

- Performance measure:
  - Sum of action costs
    If all actions cost the same this is called unit cost model

- Agent function:
  - Find cheapest sequence of actions to get to a goal state
State spaces

- Vacuum robot: 8 states
- Rubik’s cube: $10^{19}$ states...
- Climbing stairs: $\infty$ states!

Cannot represent search graph explicitly!

Implicit representation:
- Successor functions maps states to set of (action,state) pairs
- Specifies which states can be reached immediately from any given state
function Tree-Search(problem, fringe) returns a solution, or failure

fringe ← Insert(Make-Node(Initial-State[problem]), fringe)

loop do

if fringe is empty then return failure

node ← Remove-Front(fringe)

if Goal-Test(problem, State(node)) then return node

fringe ← InsertAll(Expand(node, problem), fringe)
# Comparison of algorithms

<table>
<thead>
<tr>
<th>Strategy</th>
<th>BFS</th>
<th>Uniform cost</th>
<th>DFS</th>
<th>Iterative deepening</th>
<th>Bidirect.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>Yes*</td>
<td>Yes*</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$b^{d+1}$</td>
<td>$b^{C/\epsilon}$</td>
<td>$b^{m}$</td>
<td>$b^{d}$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^{d+1}$</td>
<td>$b^{C/\epsilon}$</td>
<td>$b^{m}$</td>
<td>$b^{d}$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Optimal</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>Yes*</td>
<td>Yes*</td>
</tr>
</tbody>
</table>
Informed vs. uninformed search

- Uninformed search
  - Can only distinguish goals from non-goal states
- Informed search
  - Have information about progress towards the optimal solution
  - This lecture!
Informed search

- **Key idea:**
  Use information about how undesirable each node is
  Expand nodes in order of “undesirability”

- Implemented by using a priority queue for *fringe*

- Generalizes uninformed search
  - BFS: Undesirability = depth of node
  - DFS: Undesirability = - depth of node

- If “undesirability” chosen carefully, can get drastically improved performance
Example: Romania

Undesirability?

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</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>Dobroșen</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Făgăraș</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hîrsova</td>
<td>151</td>
</tr>
<tr>
<td>Iași</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vâlcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</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

- Use estimate $h(n)$ of distance to closest goal (heuristic)

- Greedy search sets undesirability of node as $h(n)$
Greedy search

(a) The initial state

Arad
366
(b) After expanding Arad
Greedy search

(c) After expanding Sibiu

Arad 366
Fagaras 176
Oradea 380
Rimnicu Vilcea 193

Sibiu

Arad
Timisoara 329
Zerind 374
Greedy search

(d) After expanding Fagaras

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

Distances:
- Arad to Sibiu: 366
- Arad to Fagaras: 253
Properties of Greedy search

- Complete?  No, can have loops (but can fix by remembering) if finitely many states
- Time complexity?  $O(6^m)$
- Space complexity?  $O(6^m)$ need to remember
- Optimal?  No
Example: Greedy is not complete
Example: Greedy search is not optimal

Greedy solution: Sibiu → Fagaras → Bucharest
Optimal solution: Sibiu → Rimnicu Vilcea → Pitesti → Bucharest

Distance Matrix:
- 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
A*-search

- One of the most useful search algorithms!

- **Key idea**: Prune away expensive paths!

Desirability of node $n$: $f(n) = h(n) + g(n)$
- $g(n) = \text{cost of node } n$
- $h(n) = \text{estimated cost to goal}$
- $f(n) = \text{estimated total cost of cheapest path through } n \text{ to goal}$
A*-search

(a) The initial state

Arad
366=0+366
A*-search

(b) After expanding Arad

\[
\begin{align*}
\text{Sibiu} & : 393 = 140 + 253 \\
\text{Timisoara} & : 447 = 118 + 329 \\
\text{Zerind} & : 449 = 75 + 374
\end{align*}
\]
(c) After expanding Sibiu

A*-search

Arad
Sibiu
Fagaras
Oradea
Rimnicu Vilea
Arad
447=118+329
Zerind
646=280+366 415=239+176 671=291+380 413=220+193
449=75+374
(d) After expanding Rimnicu Vilcea
A*-search

(e) After expanding Fagaras

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

Distances:
- Arad to Sibiu: 646=280+366
- Fagaras to Sibiu: 591=338+253
- Oradea to Sibiu: 671=291+380
- Rimnicu Vilcea to Craiova: 526=366+160
- Pitesti to Sibiu: 553=300+253
- Timisoara to Sibiu: 447=118+329
- Zerind to Sibiu: 449=75+374
A*-search

(f) After expanding Pitesti
Bad choice of heuristic can break A*

- Can “block” (discourage expansion of) nodes that lead to the optimal solution using large $h(n)$

A heuristic function $h(n)$ is called **admissible** if it never overestimates the true cost:

$$h(n) \leq h^*(n) \text{ for all } n$$

\[\text{cost of opt. path from } n \text{ to goal}\]

**Theorem:** For admissible heuristics, A* is optimal
Proof: Optimality of A* 

\[ g(G_i) < g(G_2) \]

\[ wtp: \ f(n) < f(G_2) \]

\[ f(G_2) = g(G_2) + h(G_2) \]

\[ = 0 \]

\[ > g(G_i) \]

\[ \geq f(n) \]

\[ g(m) + h^*(m) = g(G_1) \]

\[ \geq g(m) + h(m) = f(m) \]

So \( A^* \) has to expand all nodes \( n \) on shortest path to \( G_i \) before expanding \( G_2 \) \( \square \)
Monotonic (consistent) heuristics

- A heuristic \( h(n) \) is called **monotonic (consistent)**, if for all nodes \( n, n' \) and actions \( a \) it holds

\[
h(n) \leq c(n, a, n') + h(n')
\]

- Monotonicity implies admissibility

- Example: Straight line distance is monotonic
A* for monotonic heuristics

- A* expands nodes along monotonically increasing f-values
- With monotonicity, even A*-graph search is optimal!
Note on completeness of A*

- Technically, completeness of A* requires lower bound on action cost
- Otherwise, there could be $\infty$-many nodes with

$$f(n) \leq f(G)$$
Complexity of A*

- Generally $O(b^d)$
  - Heuristic $h(n) = 0$ is admissible..

- Can be subexponential if heuristic is accurate:
  \[ |h(n) - h^*(n)| \leq O(\log h^*(n)) \]

- Unfortunately, in practice this can be rarely guaranteed..

- But A* often still works extremely well! 😊
Properties of A* search

- Completeness: Yes * (if lower bound on action cost)
- Time complexity: \( O(b^{c/6}) \) (often much better)
- Space complexity: \( O(b^{c/6}) \) (need to remember)
- Optimality: Yes if admissible h
Reducing memory usage

- For monotonic heuristics, can use variant of IDS: IDA*
- Iteratively increase maximum f value
- Use “f-value bounded DFS”

Trades polynomial increase in running time with up to exponential reduction in space

However, in practice not as useful as IDS for good heuristics
Example: admissible heuristics

$h_1(n) = \# \text{ of misplaced tiles}$

$h_2(n) = \text{ sum of Manhattan distances to Goal}$

Start State

Goal State

7 2 4
5 6
8 3 1

3 4 5
6 7 8
Dominance

- Suppose $h_1$ and $h_2$ are admissible and
  $$h_2(n) \geq h_1(n)$$

- Then $h_2$ dominates $h_1$ and is better for search (expands fewer nodes)

- Given any two admissible heuristics $h_a$ and $h_b$, $h(n) = \max(h_a(n), h_b(n))$

  is admissible and dominates $h_a$ and $h_b$
Example: Benefit of A* search

- $h_1$: number of misplaced tiles
- $h_2$: total Manhattan distance

8-puzzle:
- IDS: 6,384 nodes
- A*(h1): 39 nodes
- A*(h2): 25 nodes

24-puzzle:
- IDS: ~54,000,000,000 nodes
- A*(h1): 39,135 nodes
- A*(h2): 1,641 nodes
Developing admissible heuristics

- Admissibility requires that \( h(n) \leq h^*(n) \)

- Ideally: Want to use \( h(n) = h^*(n) \)
- But computing \( h^*(n) \) is as hard as the search problem

- **Key idea**: Get lower bound by relaxing some constraints

- E.g., in 8 puzzle: Relax constraint that tiles can’t be on top of each other
Example: TSP

- Traveling salesman problem: Find shortest tour through graph visiting all nodes
  - NP complete

Relax: Allow arbitrary connected subgraph
MST can be calc. in $O(n^2)$
What you need to know

- Informed search uses heuristics to choose nodes for expansion
- Greedy search is suboptimal and incomplete
- A* search is optimal for admissible heuristics
- IDA* trades time for space complexity
- Can obtain admissible heuristics by relaxing the search problem