CS 232: Artificial Intelligence

Fall 2023

Prof. Carolyn Anderson
Wellesley College

Reminders

- Homework 3 will be released today
- Lyra has help hours Wednesday
- My help hours canceled due to conference
- Next class is REMOTE (on Zoom)

Q1: What is 1 thing that interested you in the reading?

Q2: What is 1 thing that you found confusing or concerning?

Q1: What is 1 thing that interested you in the reading?

Image Generation - cool!

Sandwiches - hidden layers

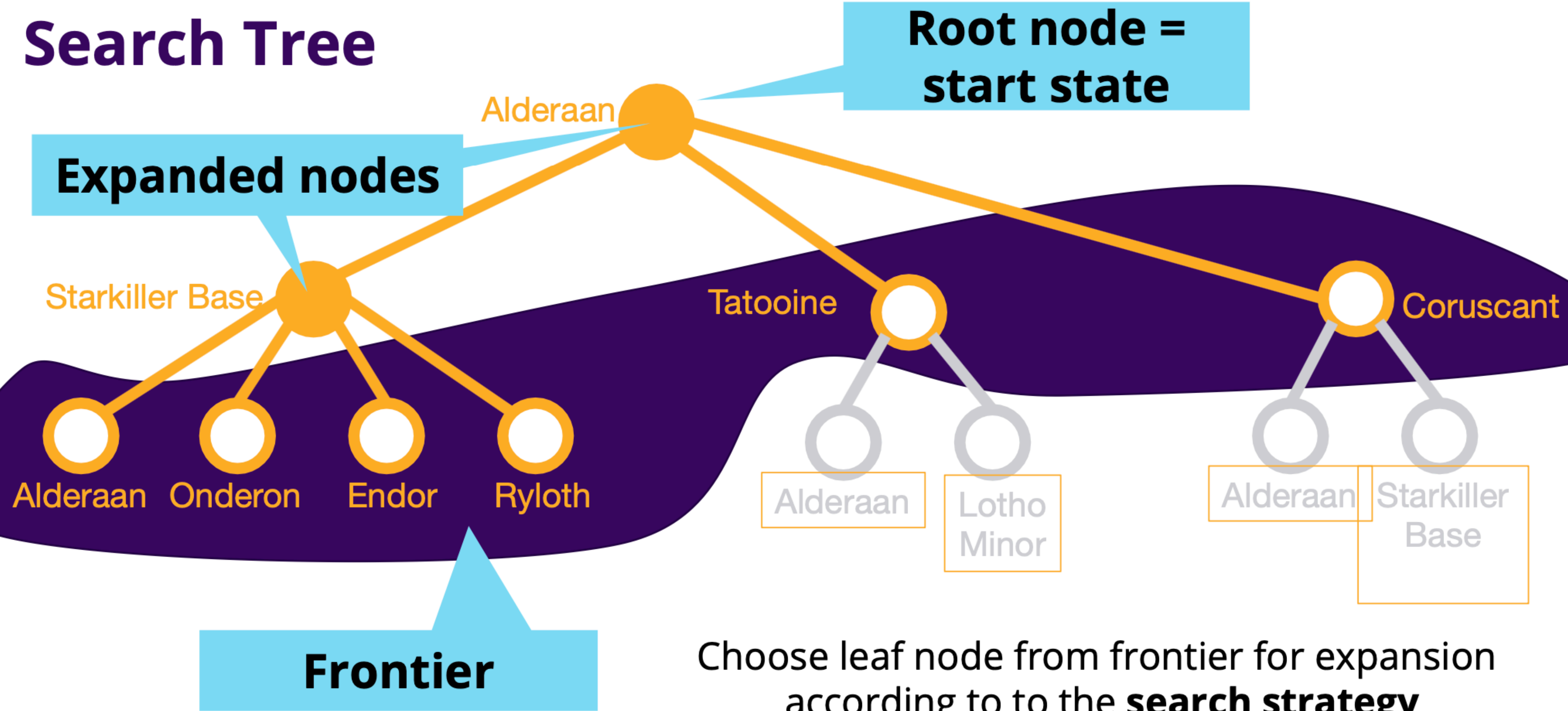
Generative Adversarial Networks (GANs)

Q2: What is 1 thing that you found confusing or concerning?

Data sparsity / class imbalance

Recap

Search Tree



Choose leaf node from frontier for expansion according to to the **search strategy**

Determines the search process

Search Strategies

Review: ***Strategy*** = order of tree expansion

- Implemented by different queue structures (LIFO, FIFO, priority)

Dimensions for evaluation

- ***Completeness*** - always find the solution?
- ***Optimality*** - finds a least cost solution (lowest path cost) first?
- ***Time complexity*** - # of nodes generated *(worst case)*
- ***Space complexity*** **- # of nodes simultaneously in memory** *(worst case)*

Time/space complexity variables

- *b*, *maximum branching factor* of search tree
- *d*, *depth* of the shallowest goal node
- *m*, maximum length of any path in the state space (potentially ∞)

Graph Search vs Tree Search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
    initialize the frontier using the initial state of *problem*
    loop do
        if the frontier is empty **then return** failure
        choose a leaf nose and remove it from the frontier
        if the node contains a goal state **then return** the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) returns a solution, or failure
    initialize the frontier using the initial state of *problem*
    initialize the explored set to be empty
    loop do
        if the frontier is empty **then return** failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state **then return** the corresponding solution
        add node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier* or *explored set

Slides adapted from Chris Callison-Burch

Depth-limited search: A building block

Depth-First search *but with depth limit l.*

- i.e. nodes at depth l *have no successors.*
- No infinite-path problem!

If $l = d$ (by luck!), then optimal

- But:
  - **If $l < d$ then incomplete** ☹
  - **If $l > d$ then not optimal** ☹

Time complexity: $O(b^l)$

Space complexity: $O(bl)$ ☺

Summary of algorithms

YES if m is not infinite

Criterion	Breadth-First	Depth-First	Depth-limited	Iterative deepening
Complete?	YES	NO	NO	YES
Time	b^d	b^m	b^l	b^d
Space	b^d	bm	bl	bd
Optimal?	YES	NO	NO	YES

d = depth of shallowest goal

m = maximum depth (depth of longest path)

Informed Search

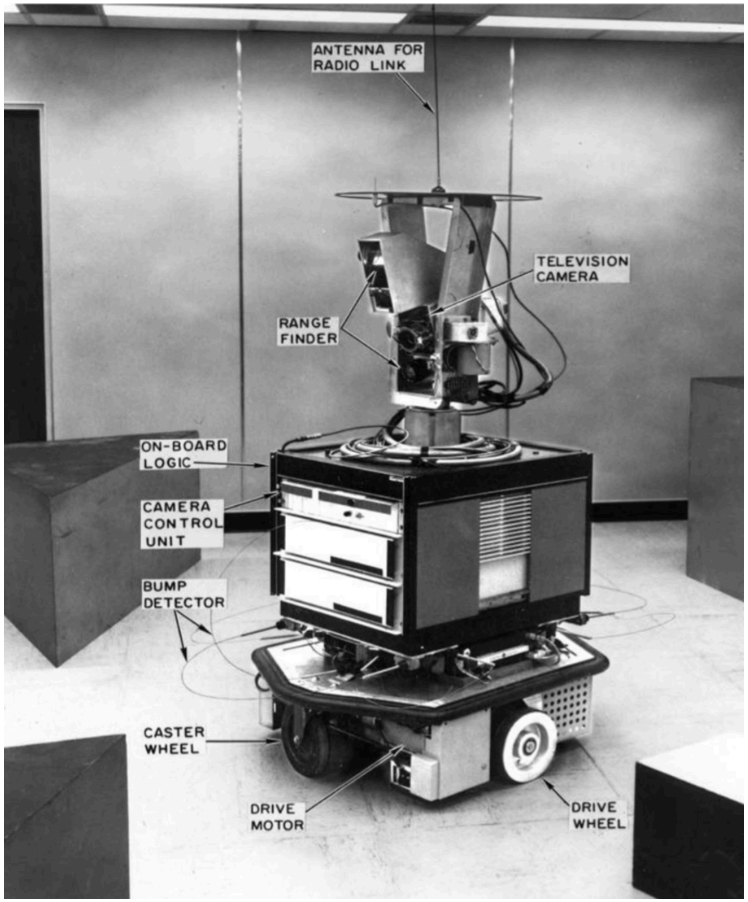
Informed Search

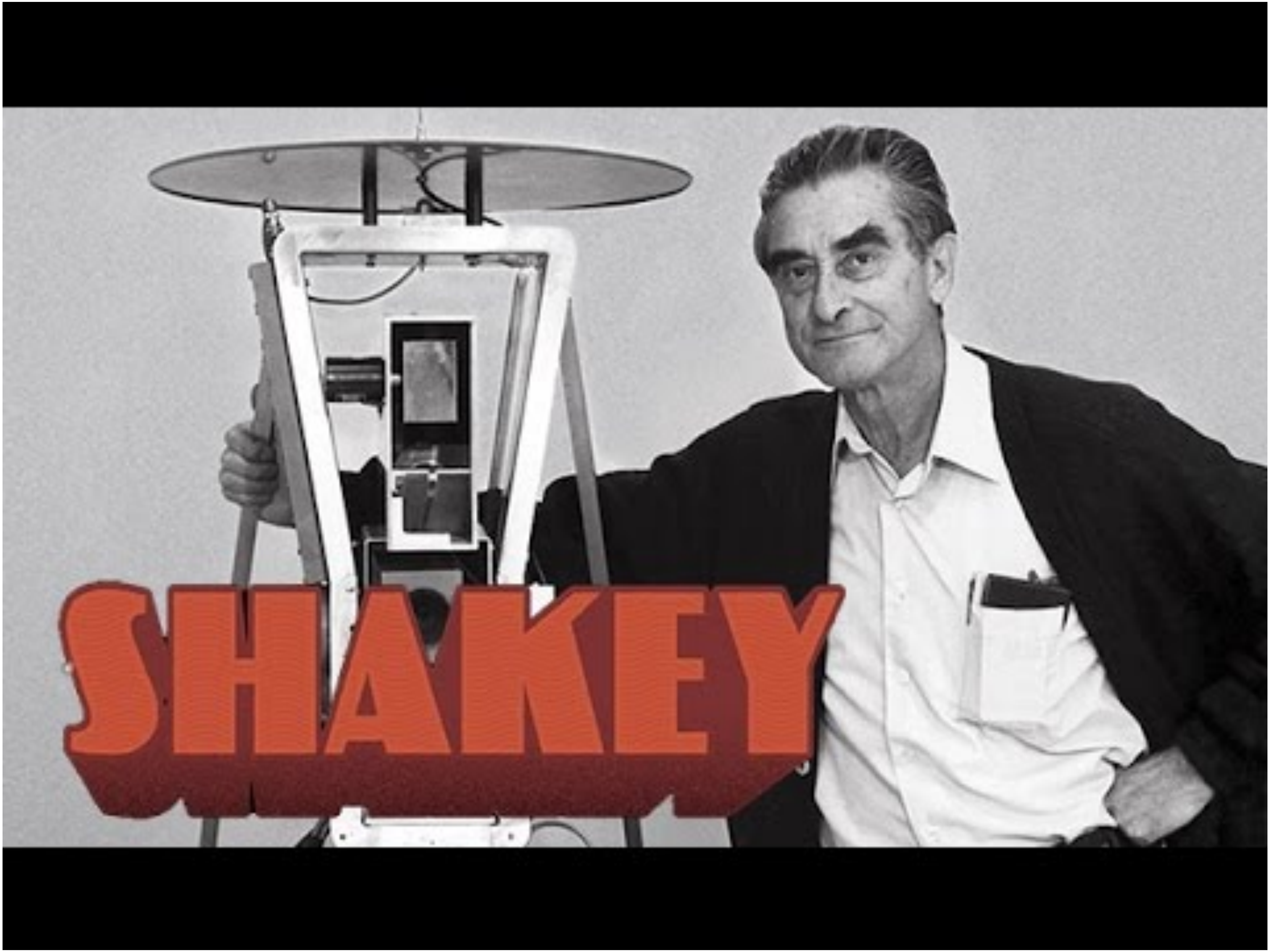
An **informed search** strategy uses **domain-specific information** about the location of the goals in order to find a solution **more efficiently** than uninformed search.

Hints will come as part of a **heuristic function** denoted h(n).

One of the most famous informed search algorithms is **A*** which was developed for **robot navigation**.

Shakey the robot was developed at the Stanford Research Institute from 1966 to 1972.

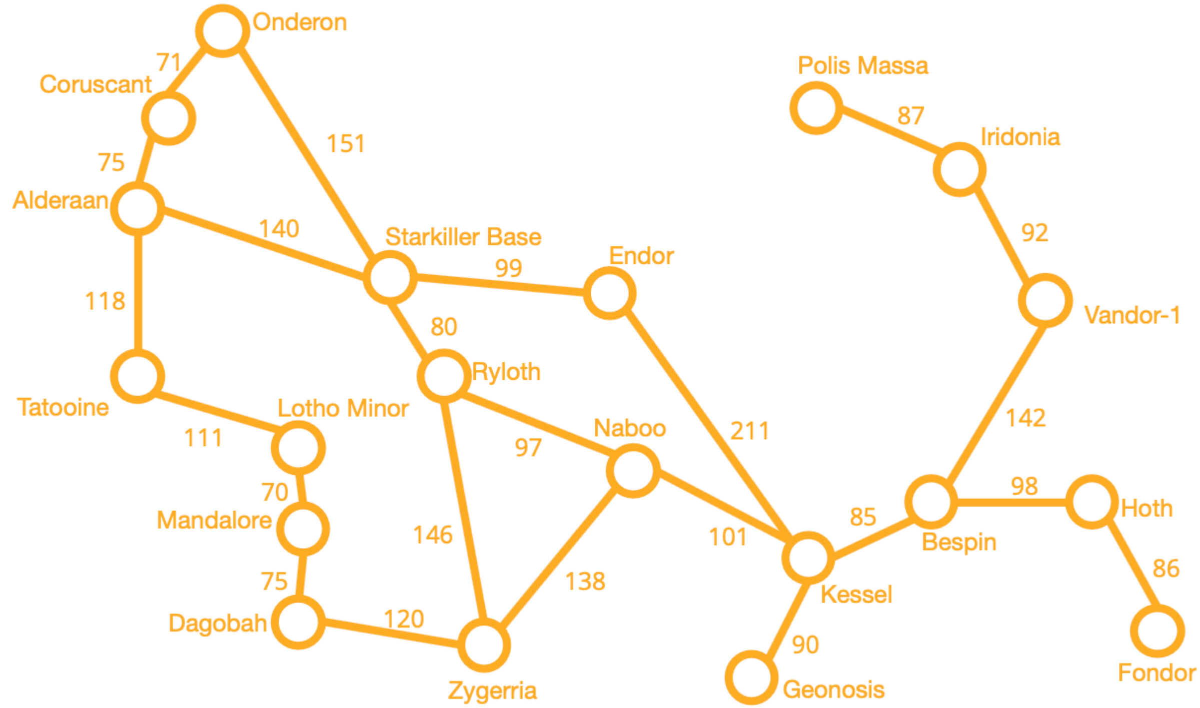



SHAKEY

Motivation: Map Navigation Problems

All our search methods so far assume *step-cost = 1*

This is only true for some problems



Slides adapted from Chris Callison-Burch

g(N): the path cost function

- **Our assumption so far: All moves equal in cost**
  - Cost = # of nodes in path-1
  - *g(N)* = *depth(N)* in the search tree

Assign a unique cost to each step:

$C(i,j)$ cost from N_i to N_j

$g(N_3) = C(0,1) + C(1,2) + C(2,3)$

Uniform-cost search (UCS)

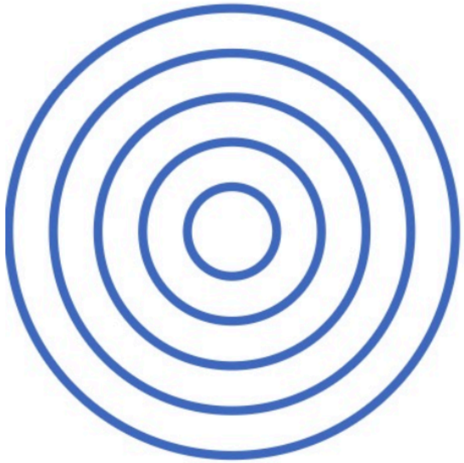
Extend BFS w/ cost

Frontier: priority queue ordered by $g(n)$

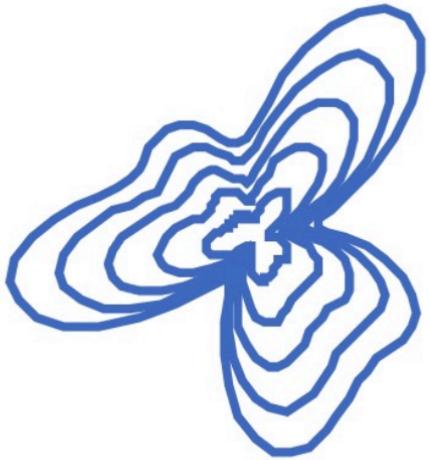
Test if a node is a goal only when we <u>expand</u> it

Update a node on the frontier if we find a cheaper path to the same state

Shape of Search



- **Breadth First Search** explores equally in all directions. Its frontier is implemented as a FIFO queue. This results in smooth contours or "plys".



- **Uniform Cost Search** lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favors lower cost paths. Its frontier is a priority queue. This results in "cost contours".

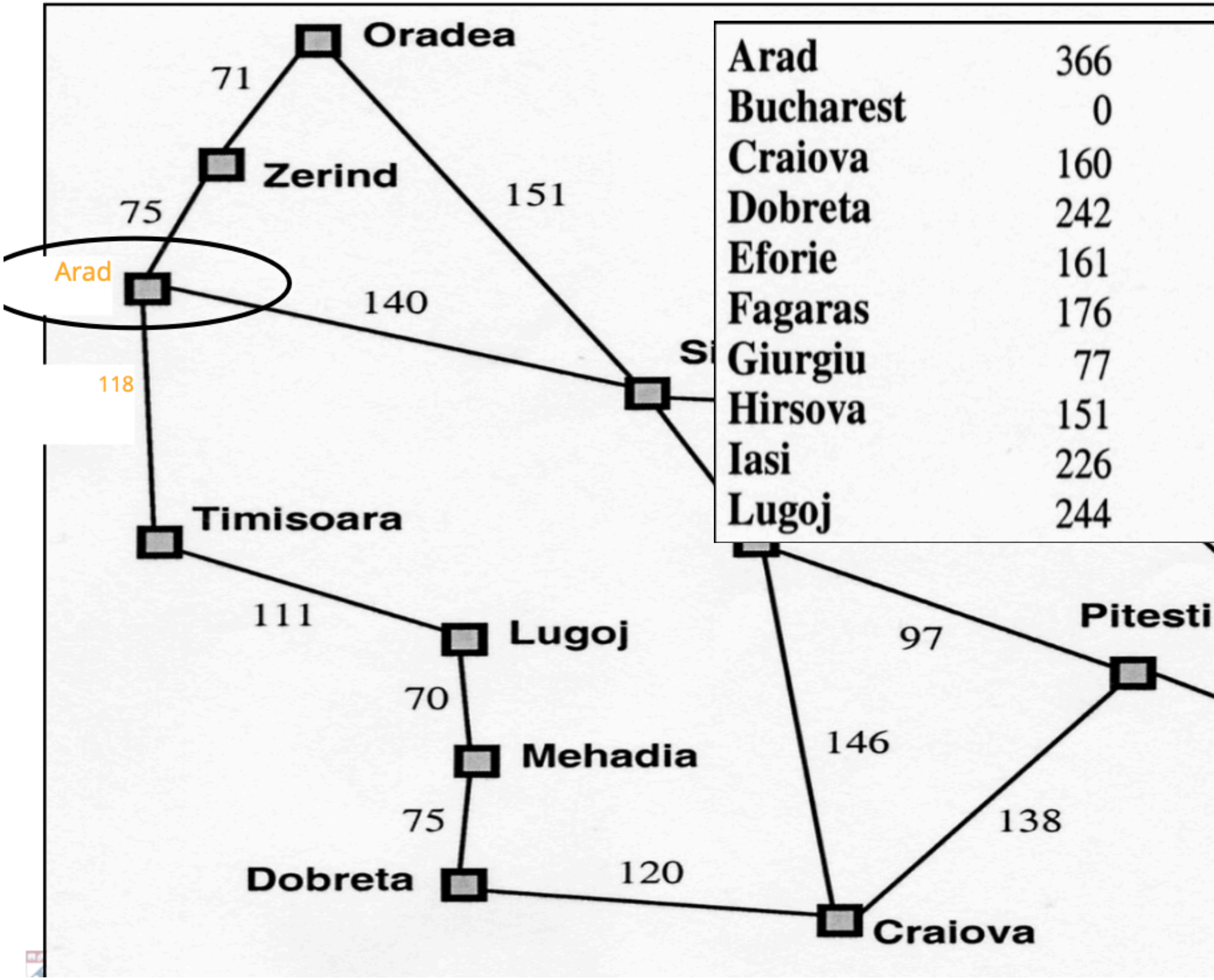
A Better Idea…

- Node expansion based on *an estimate* which *includes distance to the goal*
- General approach of informed search:
  - *Best-first search*: node selected for expansion based on an *evaluation function **f(n)***
    - ✓ ***f(n)*** includes *estimate* of distance to goal ***(new idea!)***
- Implementation: Sort frontier queue by this new ***f(n)***.
  - Special cases: **greedy search**, and ***A* search***

$g(n)$ = cost function tell us cost from start to n

$f(n)$ = heuristic that estimates cost from n to goal

Simple, useful estimate heuristic: straight-line distances



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

Slides adapted from Chris Callison-Burch

Greedy Best-First Search

Greedy best-first search: *f(n) = h(n)*

Expands the node that *is estimated* to be closest to goal

Completely ignores *g(n):* the cost to get to *n*

In our Romanian map, $h(n) = h_{SLD}(n)$ = straight-line distance from *n* to Bucharest

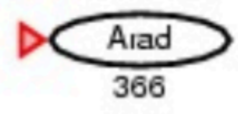
In a grid, the heuristic distance can be calculated using the “Manhattan distance”:

```
def heuristic(a, b):
   # Manhattan distance on a square grid
   return abs(a.x - b.x) + abs(a.y - b.y)
```

Greedy best-first search example

Frontier queue:

Arad 366



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

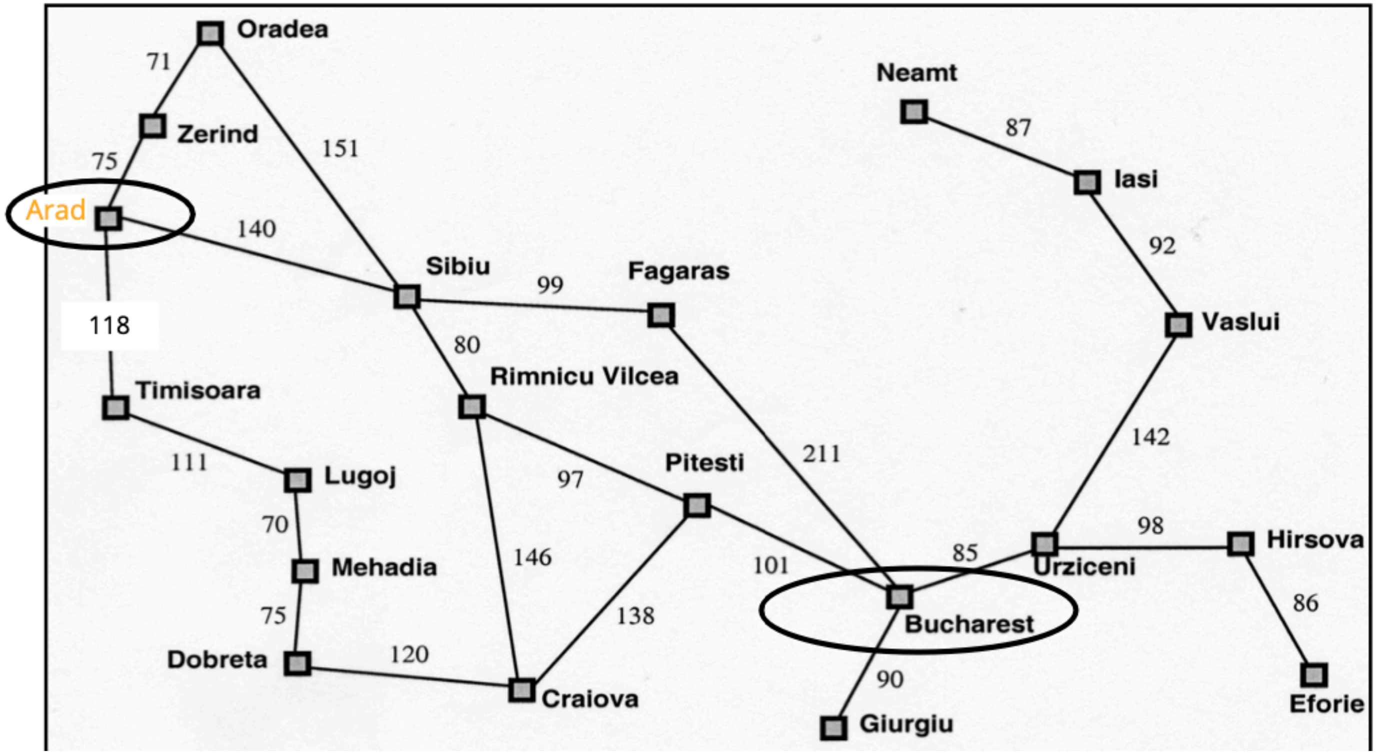
- **Initial State = Arad**
- **Goal State = Bucharest**

Slides adapted from Chris Callison-Burch

Properties of greedy best-first search

Optimal?

- No!
  - Found: *Arad → Sibiu → Fagaras → Bucharest (450km)*
  - Shorter: *Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest (418km)*



A* Search

A* search

Avoid paths that are $\underset{g(n)}{\underline{\text{expensive}}}$ but prioritize $\underset{h(n)}{\underline{\text{promising}}}$ paths.

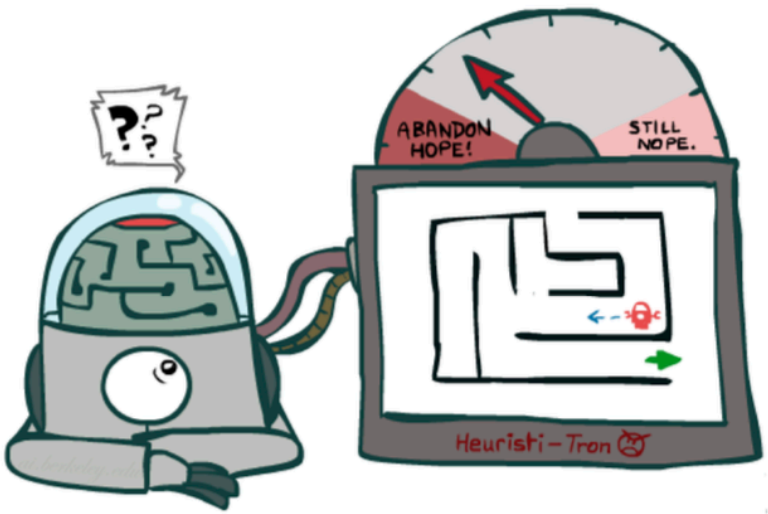
$f(n) = g(n) + h(n)$

$g(n)$ = actual cost from start to node

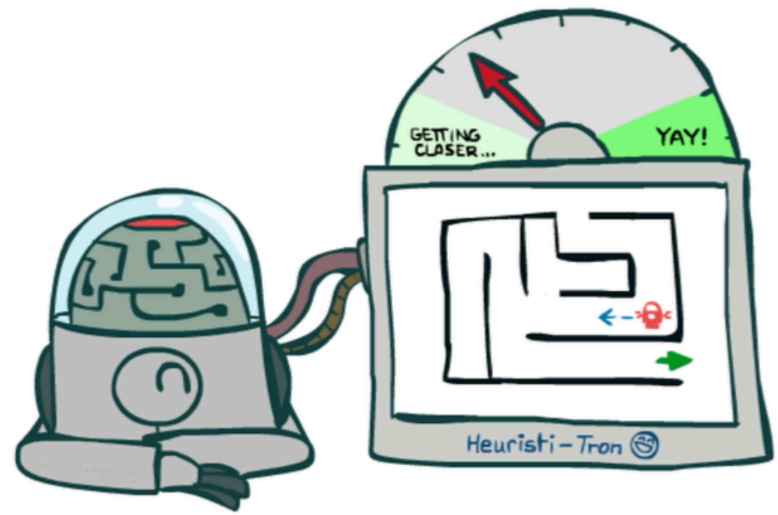
$h(n)$ = estimated cost from node to goal

$f(n)$ = estimated total cost of path from start to goal

Idea: Admissibility



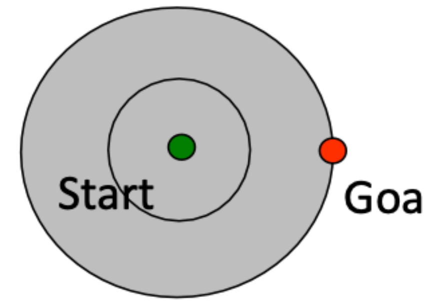
Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the frontier



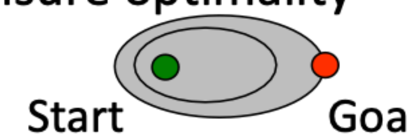
Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

UCS vs A* Contours

Uniform-cost expands equally in all "directions"



A* expands mainly toward the goal, but does hedge its bets to ensure optimality



A* Applications

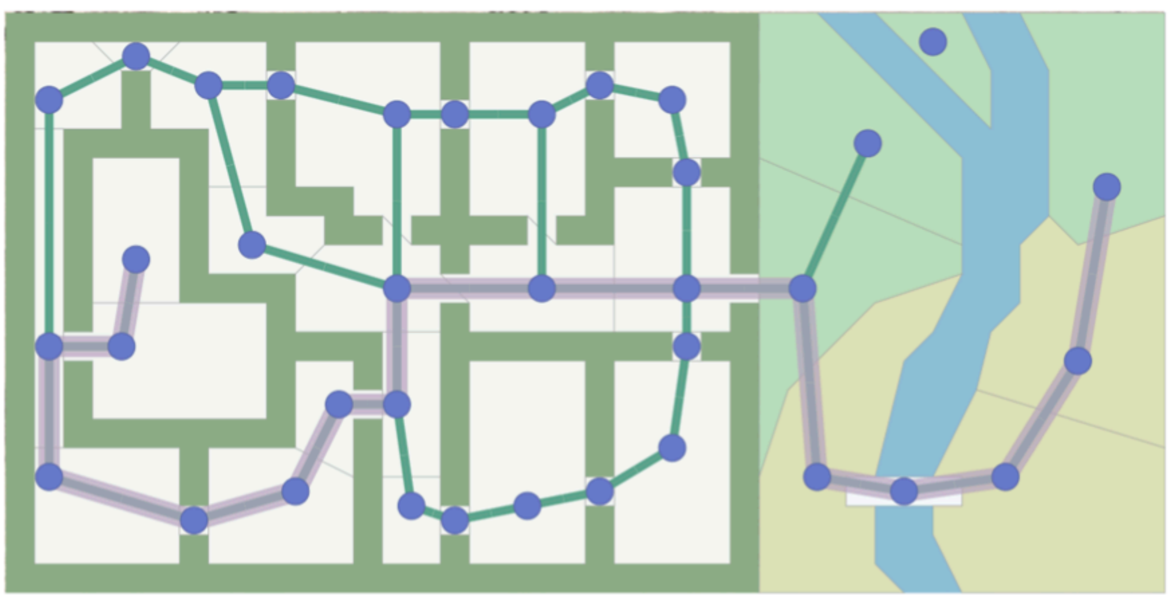
Pathing / routing problems (A* is in your GPS!)

Video games

Robot motion planning

Resource planning problems

…



Slides adapted from Chris Callison-Burch

Heuristics

Heuristic Functions

For the 8-puzzle

- **Avg. solution cost is about 22 steps**
    - **(branching factor ≤ 3)**
- **(branching factor ≤ 3)**
- **A good heuristic function can reduce the search process**

Admissible Heuristics

For the 8-puzzle:

$h_{oop}(n)$ = number of out of place tiles

$h_{md}(n)$ = total Manhattan distance (i.e., # of moves from desired location of each tile)

$h_{oop}(S) = 8$

$h_{md}(S) = 3+1+2+2+2+3+3+2 = 18$

Key: Admissibility

Inadmissible (pessimistic) heuristics break optimality by pushing good plans too far back on the frontier, which means they may never get expanded.

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs. That means that the true best plan will always be expanded.