Reminders

- HW2 has been released. It is due on Tuesday. It covers **Uninformed Search** using several classic puzzles for example exercises.
- My office hours are Mondays from 11am-1pm Eastern on Gather Town.
- Plenty of additional office hours. See the course homepage for more details.
- R2D2 supplies are gone for now – you can work in teams
- I encourage you to form study groups.

CIS 421/521:
ARTIFICIAL INTELLIGENCE

Informed Search

Review: Search problem definition

States: a set ***S***

An initial state $\boldsymbol{s_i} \in \boldsymbol{S}$

Actions: a set ***A***

$\forall$***s Actions(s)*** = *the set of actions that can be executed in* ***s****, that are applicable* in ***s***.

Transition Model: $\forall$***s*** $\forall$***a*** $\in$***Actions(s) Result(s, a)*** $\rightarrow$ $\boldsymbol{s_r}$

$\boldsymbol{s_r}$ *is called a successor of* ***s***

$\{\boldsymbol{s_i}\} \cup$ ***Successors****(*$\boldsymbol{s_i}$*)** *= state space*

Path cost (Performance Measure): Must be additive

e.g. sum of distances, number of actions executed, …

c(x,a,y) is the step cost, assumed ≥ 0

(where action ***a*** goes from state ***x*** to state ***y****)*

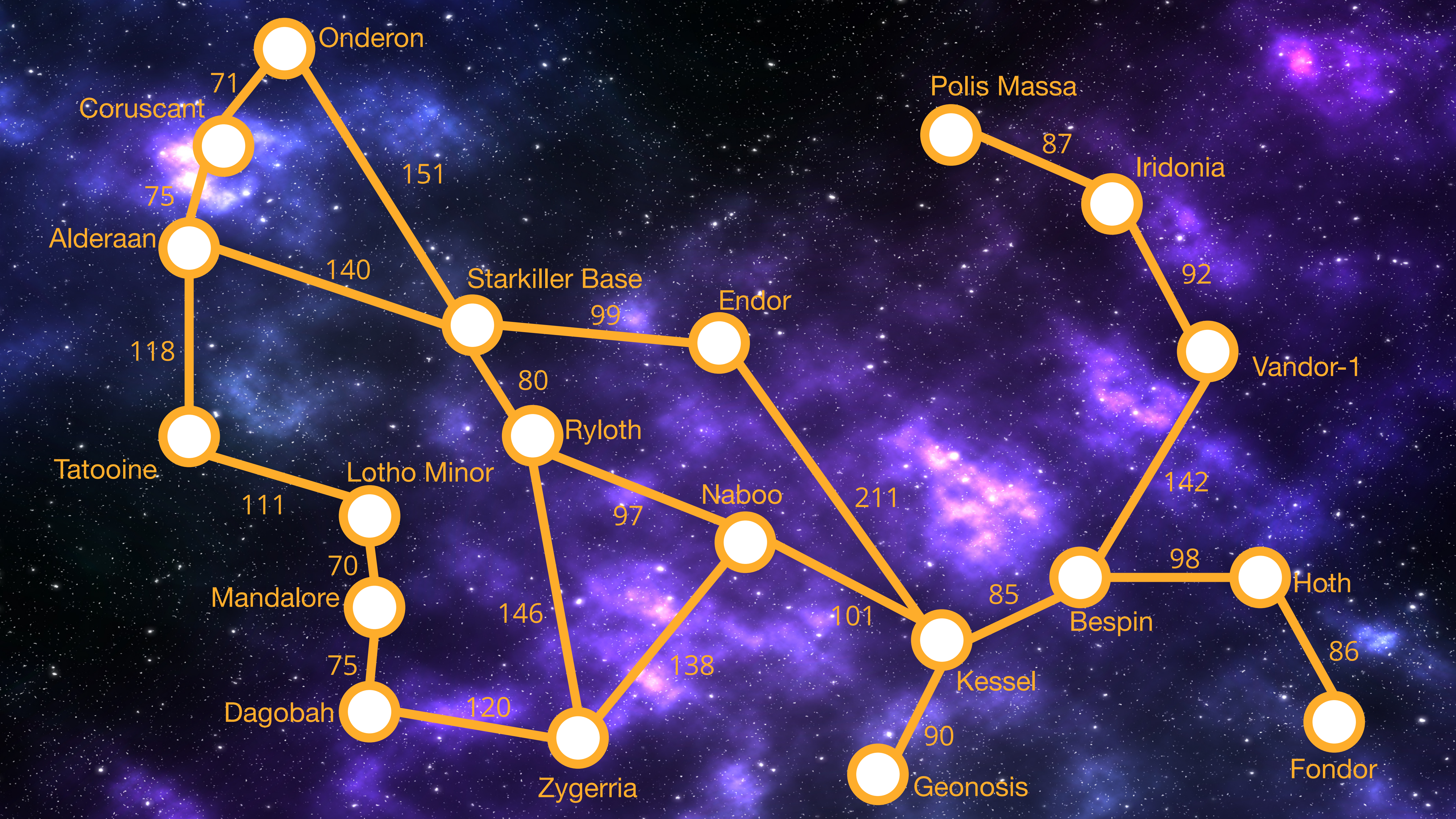
Goal test: ***Goal(s)***

Can be implicit, e.g. ***checkmate(s)***

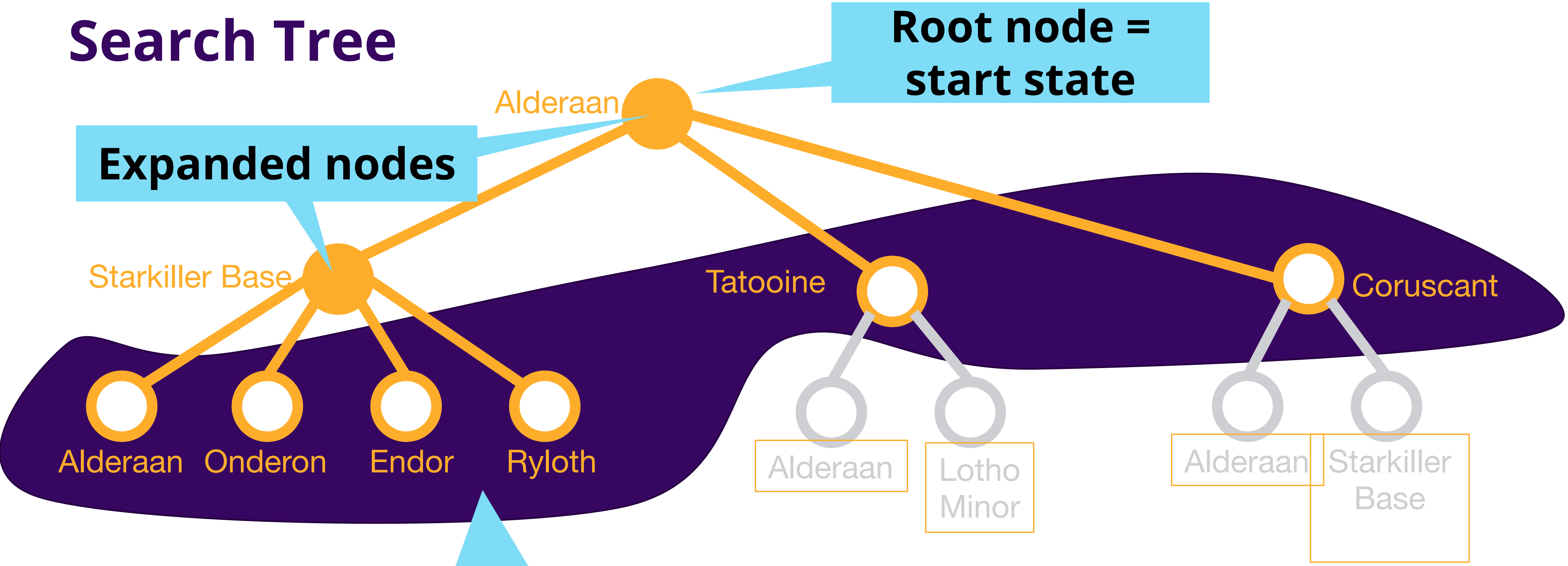
s is a *goal state* if ***Goal(s)*** is *true*

Review: Useful Concepts

- *State space*: the set of all states reachable from the initial state by *any* sequence of actions
  - *When several operators can apply to each state, this gets large very quickly*
  - *Might be a proper subset of the set of configurations*
- *Path*: a sequence of actions leading from one state s_j to another state s_k
- *Frontier:* those states that are available for *expanding* (for applying legal actions to)
- *Solution*: a path from the initial state s_i to a state s_g that satisfies the goal test


Onderon
71
Coruscant
151
75
Alderaan
140
Starkiller Base
99
Endor
118
80
Ryloth
Tatooine
Lotho Minor
111
97
Naboo
211
70
Mandalore
146
101
75
138
Dagobah
120
Zygerria
Polis Massa
87
Iridonia
92
Vandor-1
142
98
Hoth
85
Bespin
Kessel
86
90
Geonosis
Fondor

Search Tree



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

Determines the search process

Review: Search Strategies

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 ∞)

Animation of Graph BFS algorithm set to music 'flight of bumble bee'

https://youtu.be/x-VTfcmrLEQ

Animation of Graph DFS algorithm
Depth First Search of Graph
set to music 'flight of bumble bee'

https://youtu.be/NUgMa5coCoE

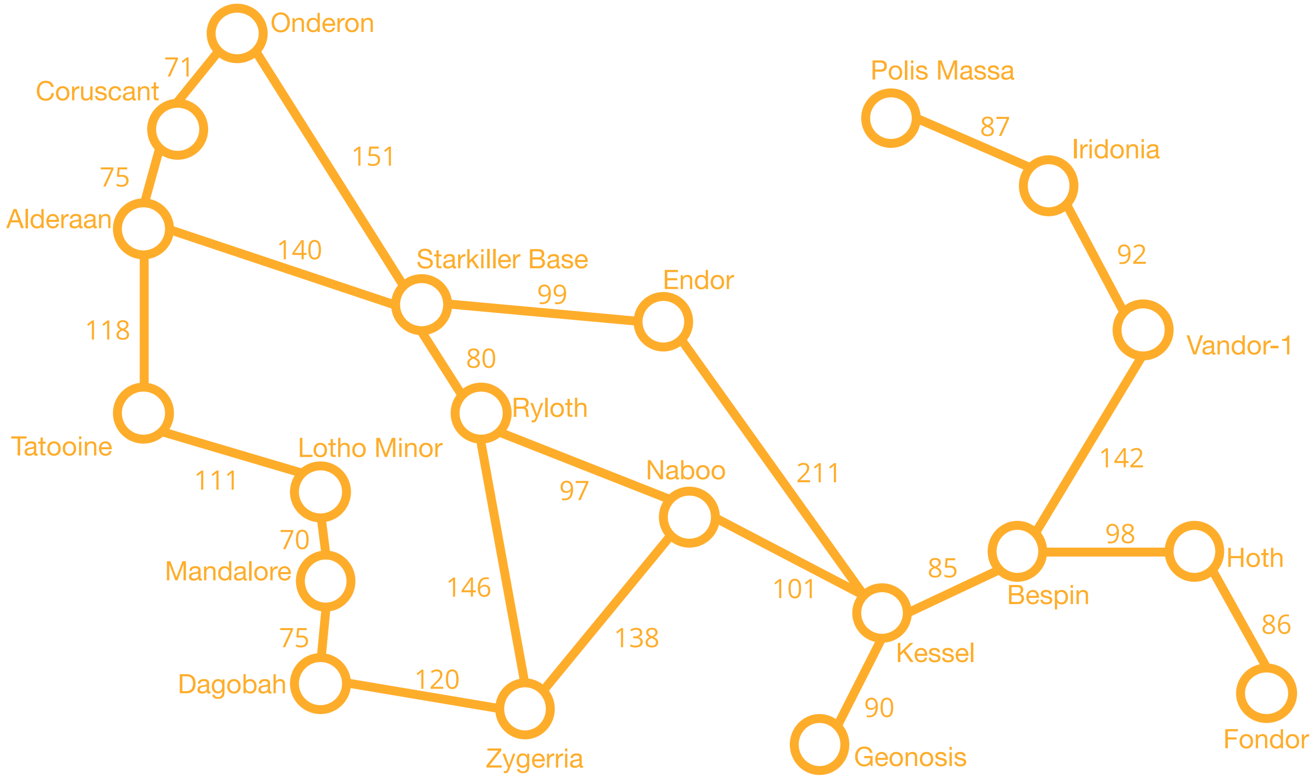
"Uniform Cost" Search

"In computer science, *uniform-cost* search (UCS) is a tree search algorithm used for traversing or searching a *weighted* tree, tree structure, or graph." - Wikipedia

Motivation: Map Navigation Problems

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

This is only true for some problems



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

- More general: Assigning a (potentially) unique cost to each step
  - N_0, N_1, N_2, N_3 = nodes visited on path *p* from N_0 to N_3
  - *C(i,j):* Cost of going from N_i to N_j
  - If N_0 the root of the search tree,

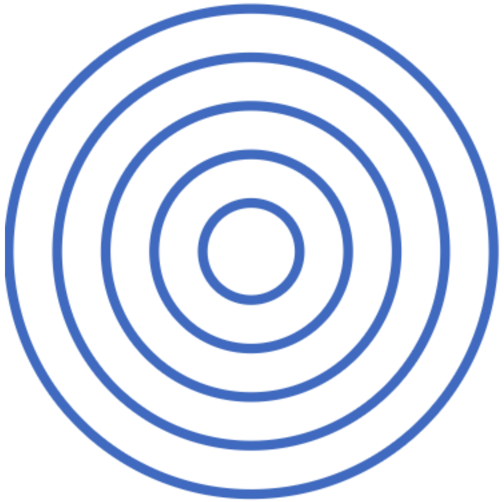
$$g(N3)=C(0,1)+C(1,2)+C(2,3)$$

Uniform-cost search (UCS)

- Extension of BF-search:
  - **Expand node with *lowest path cost***
- Implementation:
  - *frontier* = priority queue ordered by g(n)
- Subtle but significant difference from BFS:
  - Tests if a node is a goal state when it is selected for expansion, not when it is added to the frontier.
  - Updates a node on the frontier if a better path to the same state is found.
  - So always enqueues a node *before checking whether it is a goal.*

WHY???

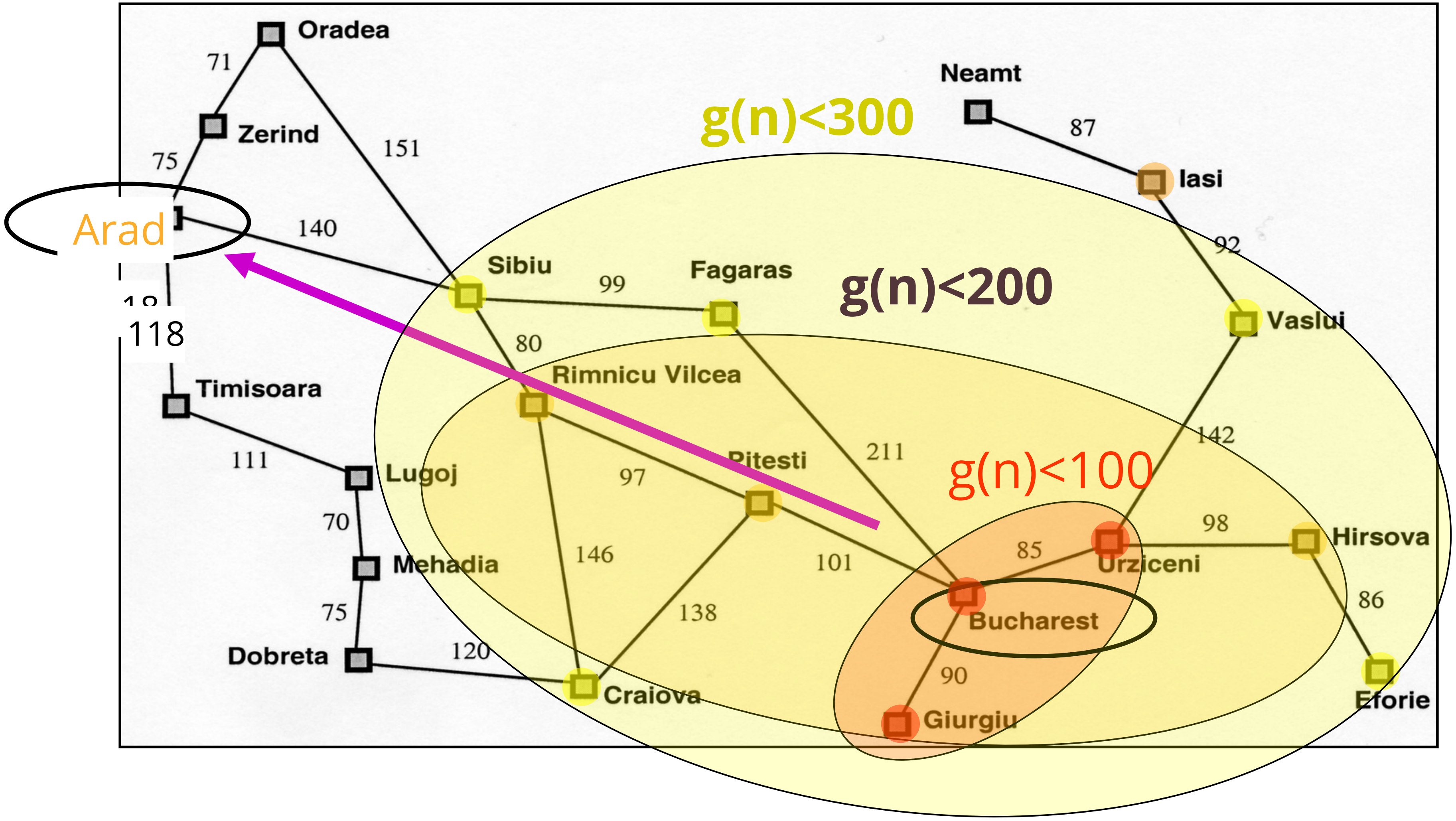
Shape of Search



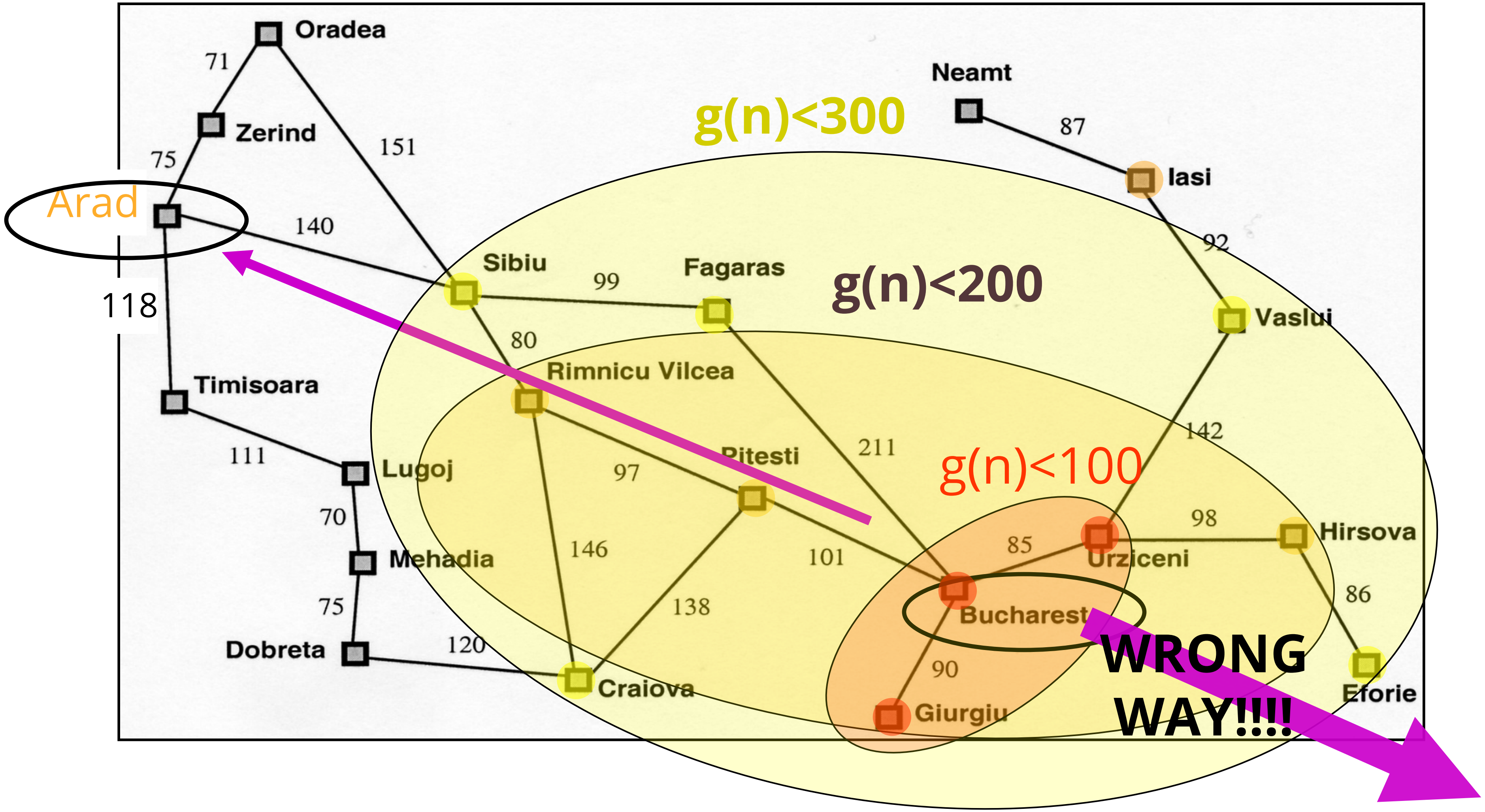
- **Breadth First Search** explores equally in all directions. Its frontier is implemented as a FIFO queue. This results in smooth contours or “plys”.

- **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”.

Is Uniform Cost Search the best we can do? Consider finding a route from Bucharest to Arad..



Is Uniform Cost Search the best we can do? Consider finding a route from Bucharest to Arad..



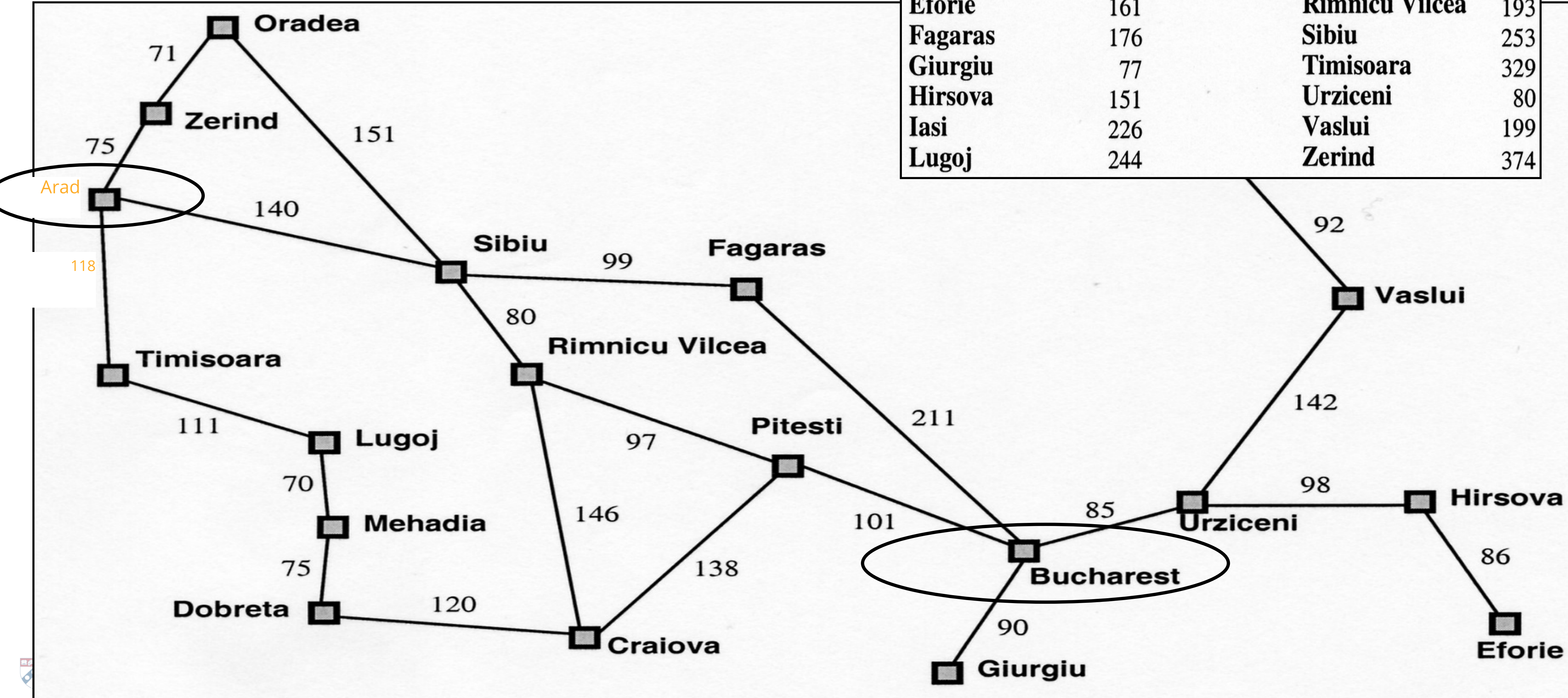
A Better Idea…

- 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***

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



Heuristic (estimate) functions

Heureka! ---Archimedes

[dictionary]*"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood."*

Heuristic knowledge is useful, but not necessarily correct.

Heuristic algorithms use heuristic knowledge to solve a problem.

A *heuristic function h(n)* takes a state *n* and returns an *estimate* of the distance from *n* to the goal.

Greedy Best-First Search

First attempt at integrating heuristic knowledge

Review: Best-first search

Basic idea:

select node for expansion with minimal *evaluation function* ***f(n)***

- where ***f(n)*** is some function that includes ***estimate heuristic h(n)*** of the remaining distance to goal

Implement using priority queue

Exactly UCS with *f(n)* replacing *g(n)*

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":

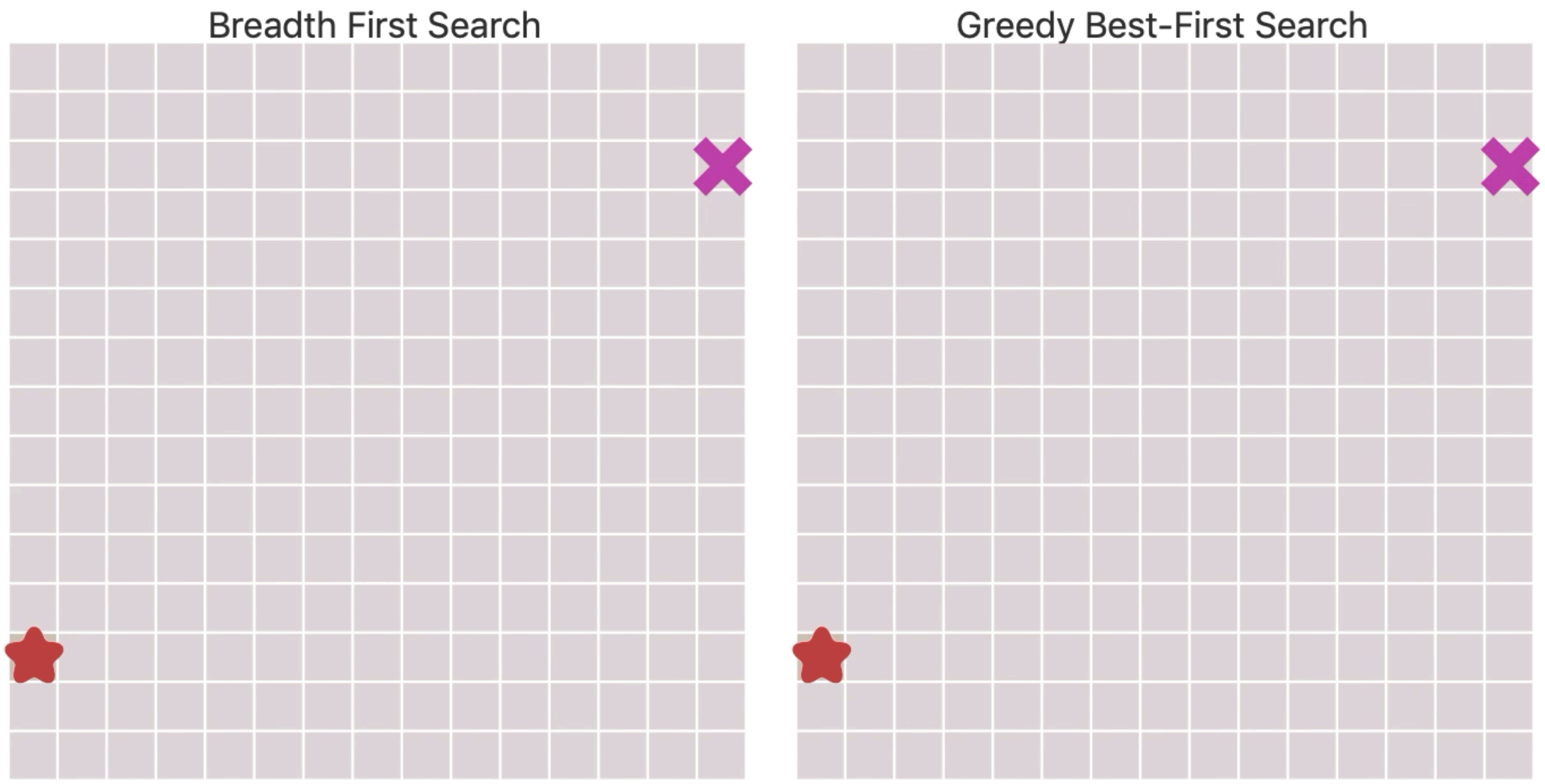
```
def heuristic(a, b):
   # Manhattan distance on a square grid
   return abs(a.x - b.x) + abs(a.y - b.y)
```

Greedy best-first search

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
came_from[start] = None

while not frontier.empty():
    current = frontier.get()

    if current == goal:
        break

    for next in graph.neighbors(current):
        if next not in came_from:
            priority = heuristic(goal, next)
            frontier.put(next, priority)
            came_from[next] = current
```

Code from Amit Patel
of Red Blob Games

BFS v. Greedy Best-First Search

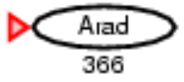


https://www.redblobgames.com/pathfinding/a-star/introduction.html

Greedy best-first search example

Frontier queue:

Arad 366



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

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

Greedy best-first search example

Frontier queue:

Sibiu 253

Timisoara 329

Zerind 374

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

Greedy best-first search example

Frontier queue:

Fagaras 176

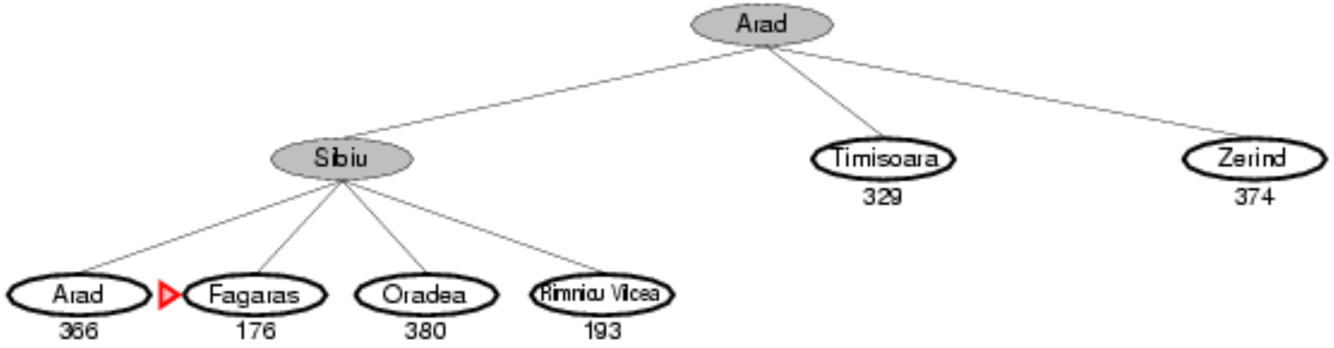
Rimnicu Vilcea 193

Timisoara 329

Arad 366

Zerind 374

Oradea 380



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

Greedy best-first search example

Frontier queue:

Bucharest 0

Rimnicu Vilcea 193

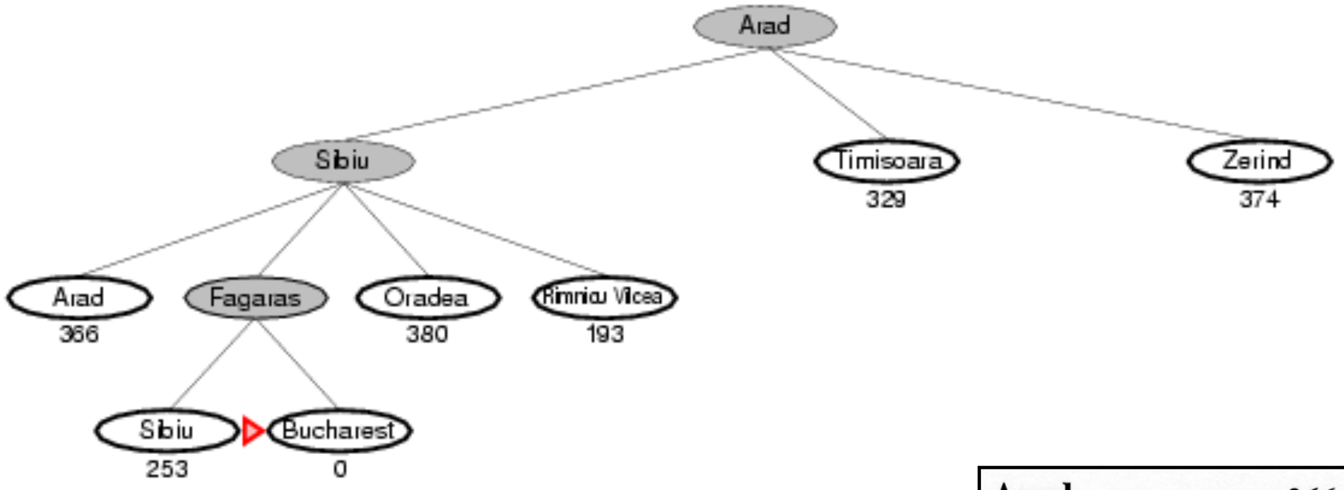
Sibiu 253

Timisoara 329

Arad 366

Zerind 374

Oradea 380



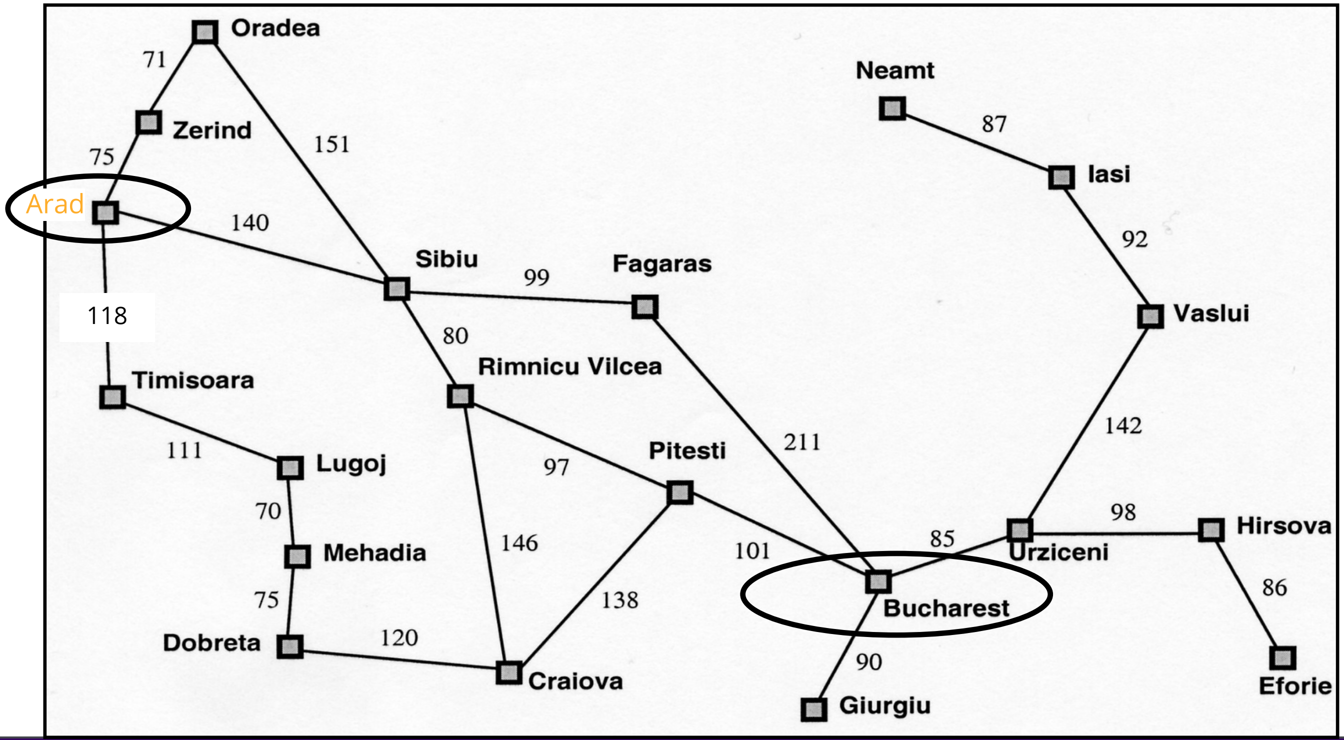
Goal reached !!

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

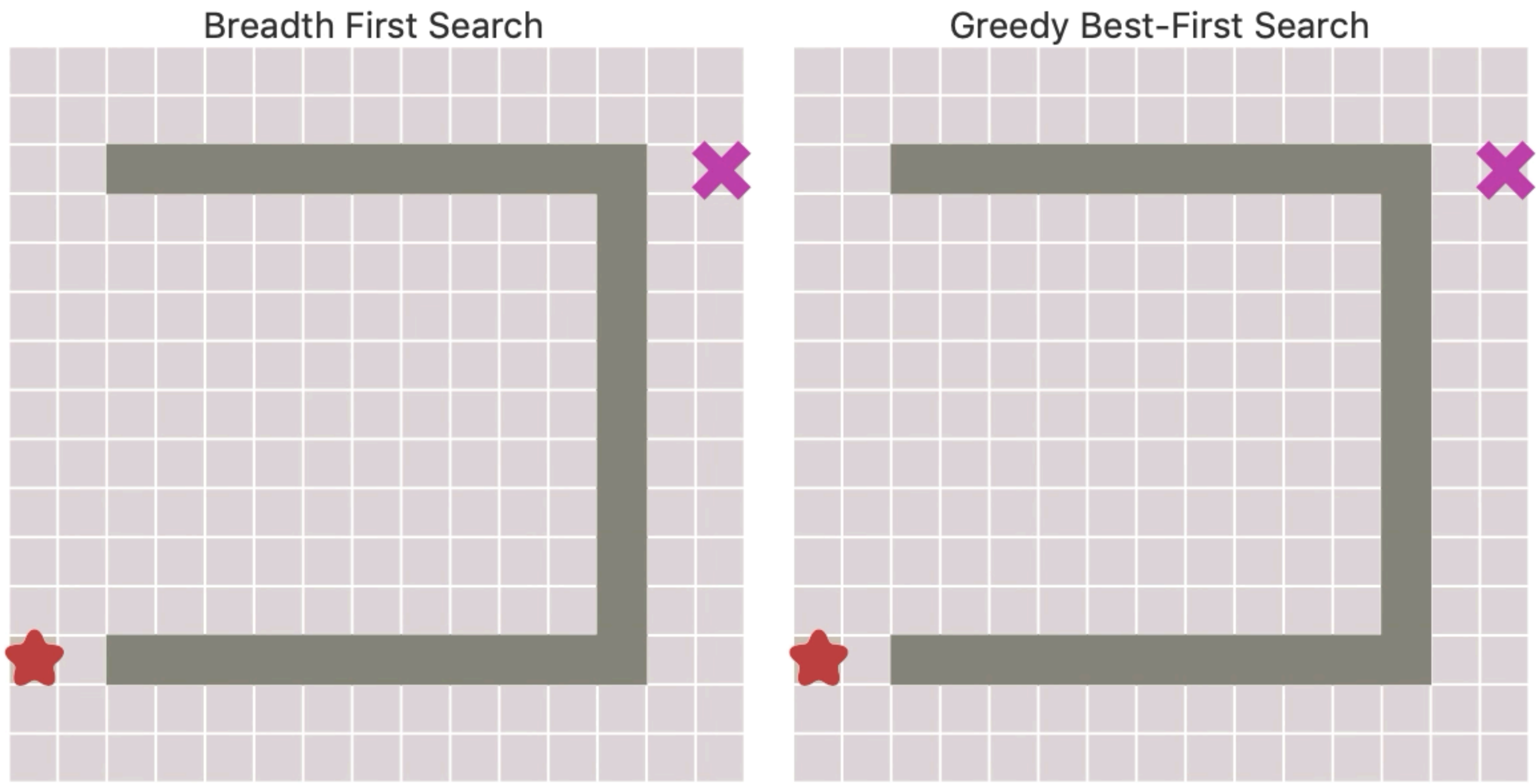
Properties of greedy best-first search

Optimal?

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



BFS v. Greedy Best-First Search

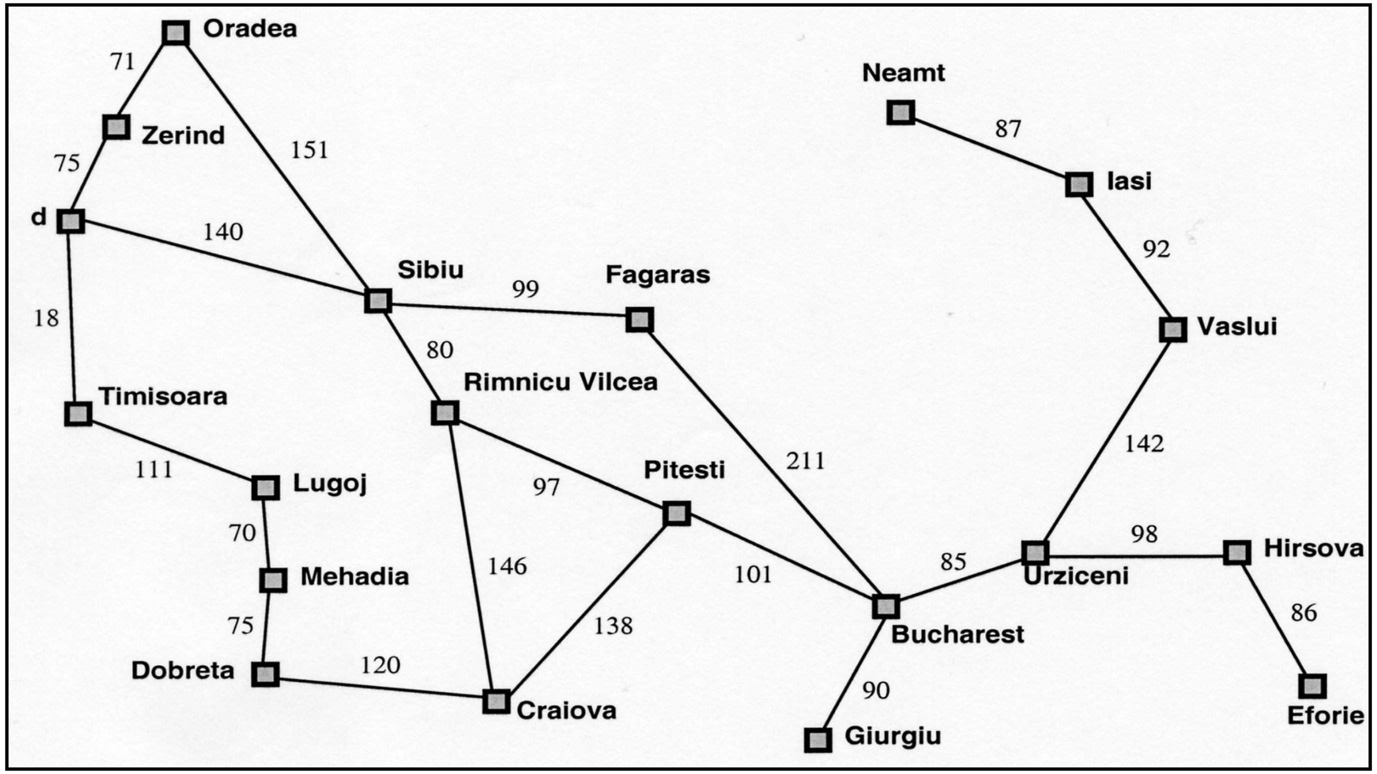


https://www.redblobgames.com/pathfinding/a-star/introduction.html

Properties of greedy best-first search

Complete?

- No – can get stuck in loops,
- e.g., Iasi → Neamt → Iasi → Neamt →…



A* search

A* search

Best-known form of best-first search.

Key Idea: avoid expanding paths that are already expensive, but expand most promising first.

Simple idea: *$f(n)=g(n) + h(n)$*

- ***g(n)*** the actual cost (so far) to *reach* the node
- ***h(n)*** estimated cost to *get from the node to the goal*
- ***f(n)*** estimated *total cost* of path through *n* to goal

Implementation: Frontier queue as priority queue by increasing *f(n)* *(as expected…)*

Key concept: Admissible heuristics

A heuristic *h(n)* is *admissible* if it *never overestimates* the cost to reach the goal;

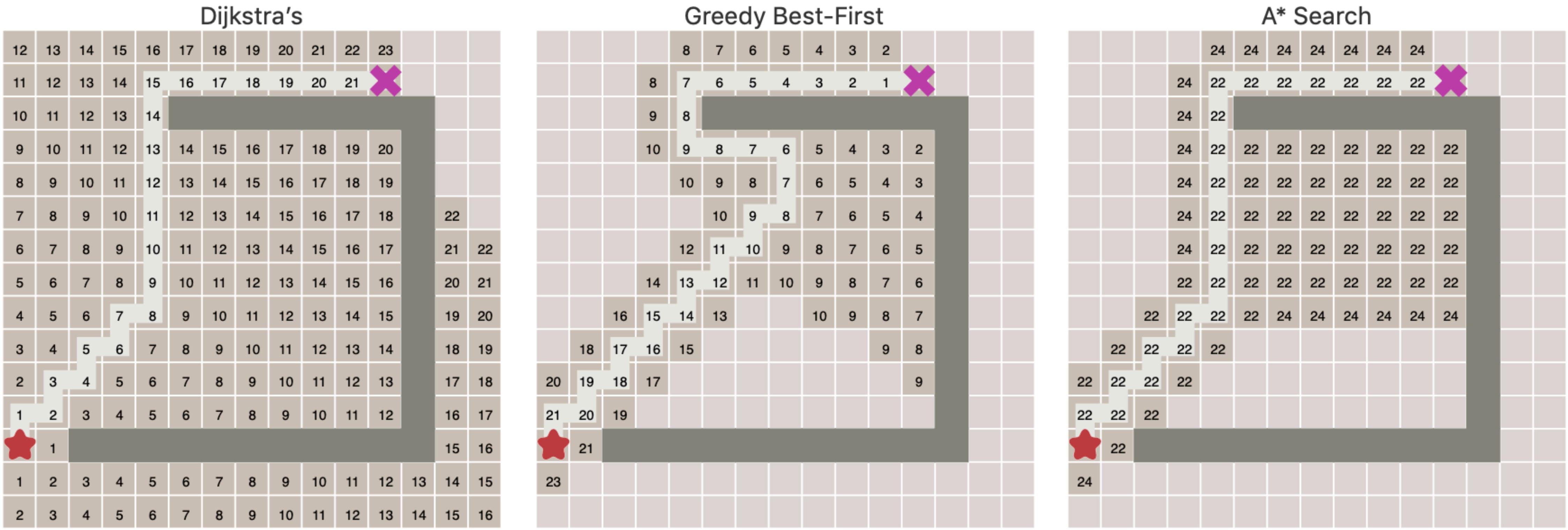
i.e. it is *optimistic*

- Formally: $\forall n$, n a node:
  - $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n
  - $h(n) \geq 0$ so $h(G)=0$ for any goal G.

Example: $h_{SLD}(n)$ never overestimates the actual road distance

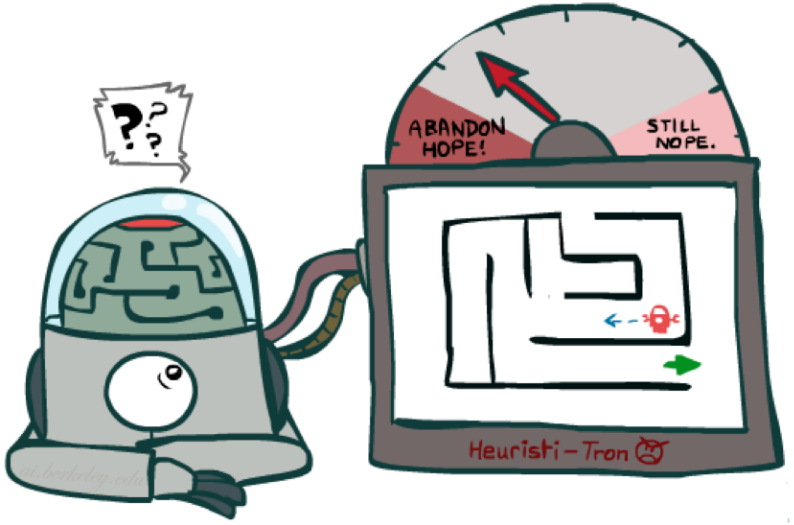
Theorem: If *h(n)* is *admissible*, A^* using Tree Search is *optimal*

A* is optimal with admissible heuristic

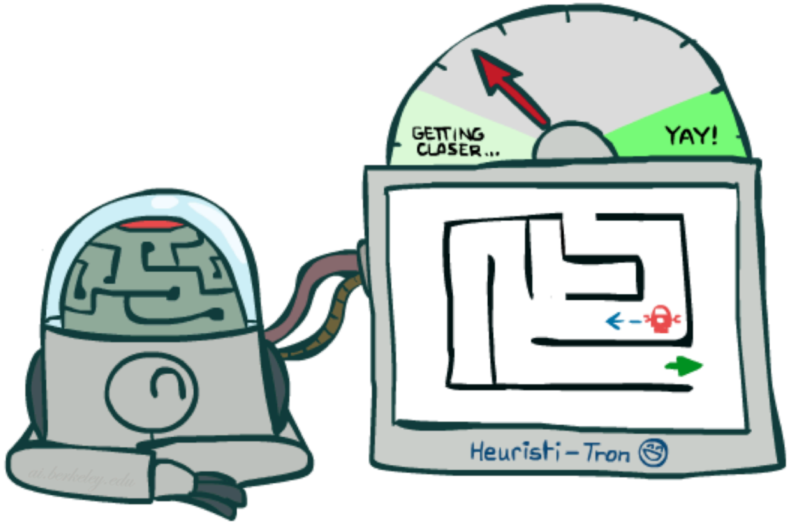


https://www.redblobgames.com/pathfinding/a-star/introduction.html

Idea: Admissibility



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



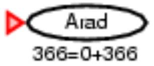
Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

A* search example

Frontier queue:

Arad 366



A* search example

Frontier queue:

Sibiu 393

Timisoara 447

Zerind 449

We add the three nodes we found to the Frontier queue.

We sort them according to the **g()+h()** calculation.

A* search example

Frontier queue:

Rimricu Vicea 413

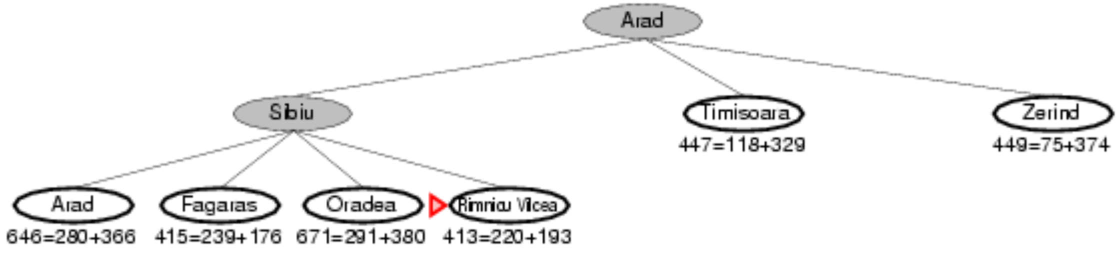
Fagaras 415

Timisoara 447

Zerind 449

Arad 646

Oradea 671



When we expand Sibiu, we run into Arad again. Note that we've already expanded this node once; but we still add it to the Frontier queue again.

A* search example

Frontier queue:

Fagaras 415

Pitesti 417

Timisoara 447

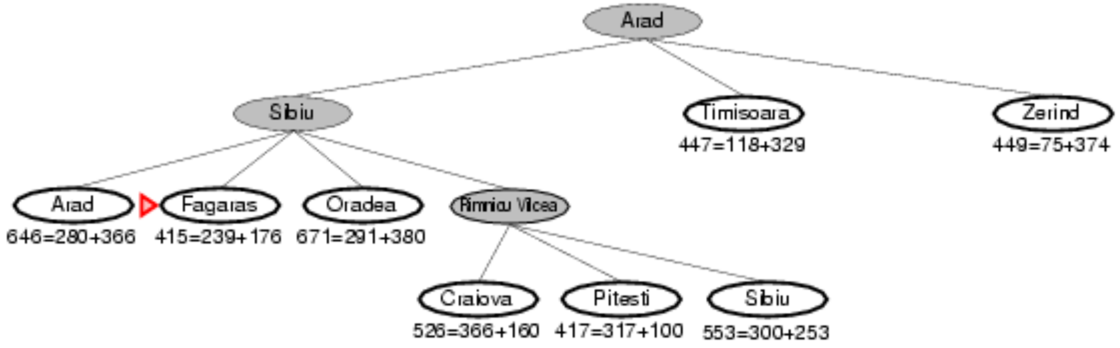
Zerind 449

Craiova 526

Sibiu 553

Arad 646

Oradea 671



We expand Rimricu Vicea.

A* search example

Frontier queue:

Pitesti 417

Timisoara 447

Zerind 449

Bucharest 450

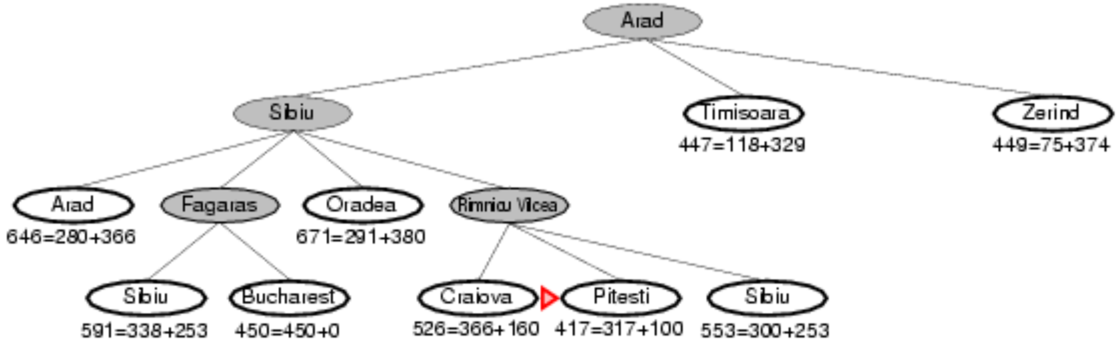
Craiova 526

Sibiu 553

Sibiu 591

Arad 646

Oradea 671



When we expand Fagaras, we find Bucharest, but we're not done. The algorithm doesn't end until we "expand" the goal node – it has to be at the top of the Frontier queue.

A* search example

Frontier queue:

Bucharest 418

Timisoara 447

Zerind 449

Bucharest 450

Craiova 526

Sibiu 553

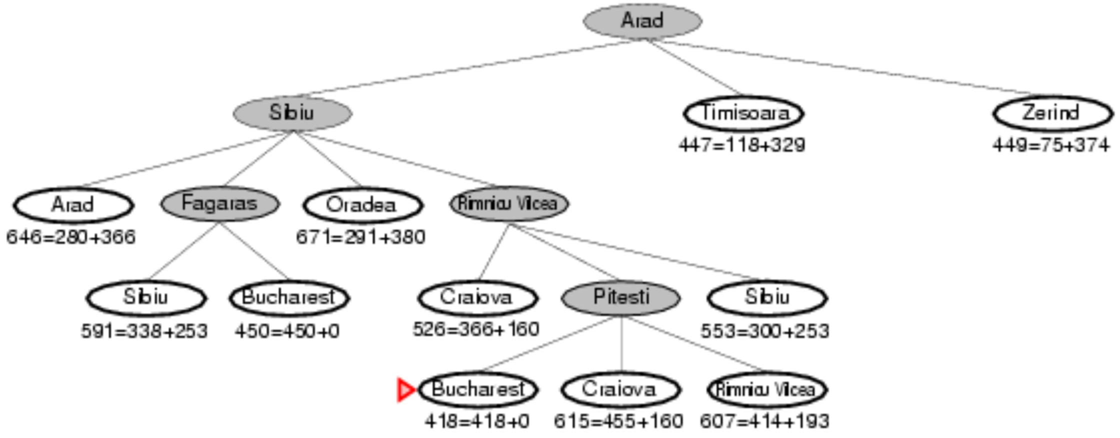
Sibiu 591

Rimricu Vicea 607

Craiova 615

Arad 646

Oradea 671



Note that we just found a better value for Bucharest!

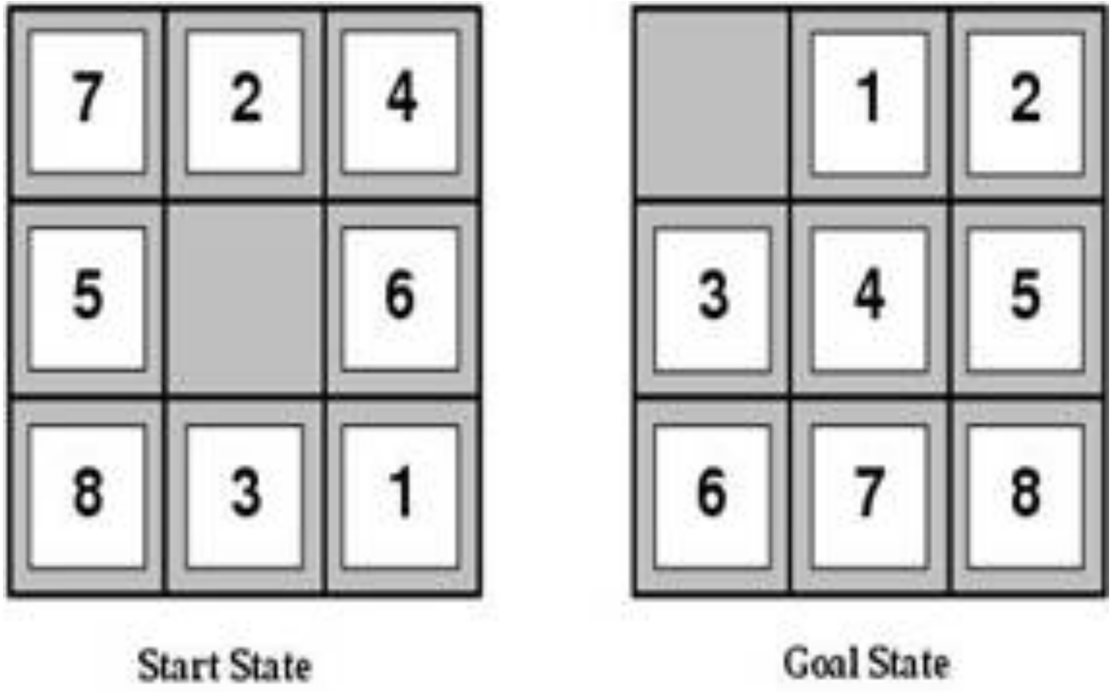
Now we expand this better value for Bucharest since it’s at the top of the queue.

We’re done and we know the value found is optimal!

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**



Example Admissible heuristics

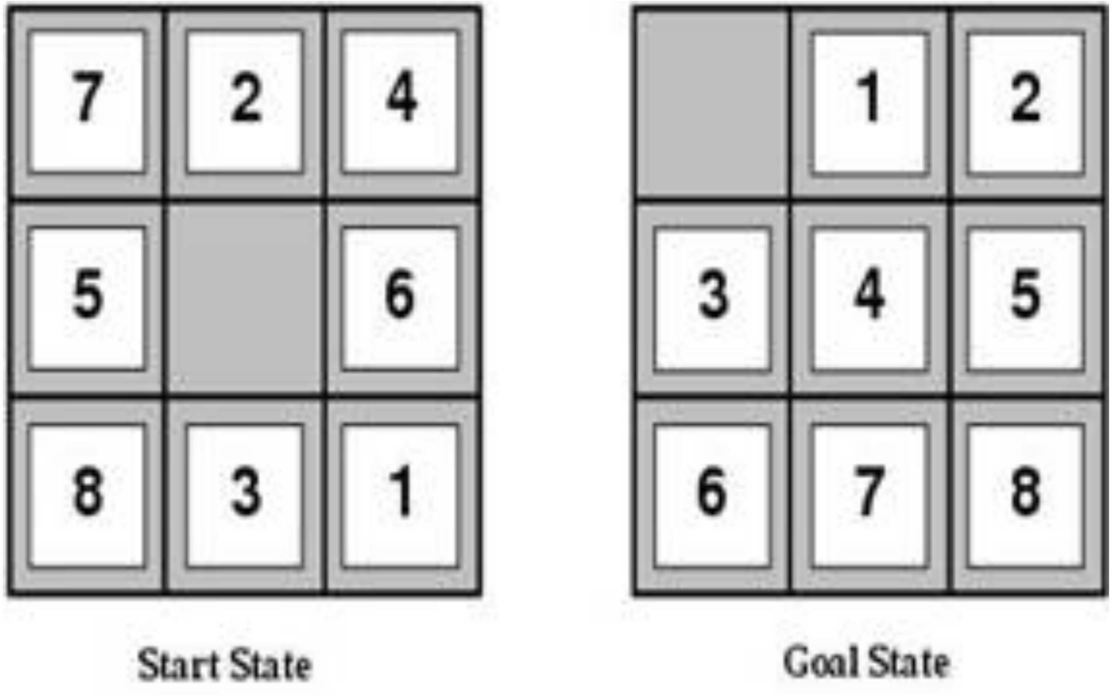
For the 8-puzzle:

$h_{oop}(n)$ = 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



Relaxed problems

A problem with fewer restrictions on the actions than the original is called a *relaxed problem*

The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_{oop}(n)$ gives the shortest solution

If the rules are relaxed so that a tile can move to *any adjacent square,* then $h_{md}(n)$ gives the shortest solution

Defining Heuristics: *h(n)*

Cost of an exact solution to a *relaxed* problem (fewer restrictions on operator)

Constraints on *Full* Problem:

A tile can move from square A to square B *if* A is adjacent to B *and* B is blank.

- Constraints on *relaxed* problems:
  - A tile can move from square A to square B *if* A is adjacent to B. (h_{md})
  - A tile can move from square A to square B *if* B is blank.
  - A tile can move from square A to square B. (h_{oop})

Dominance: A metric on better heuristics

If $h_2(n) \geq h_1(n)$ for all n (both admissible)

- then h_2 *dominates* h_1

So h_2 is optimistic, but more accurate than h_1

- $\boldsymbol{h_2}$ is therefore better for search
- Notice: $\boldsymbol{h_{md}}$ dominates $\boldsymbol{h_{oop}}$

Typical search costs (average number of nodes expanded):

d=12 Iterative Deepening Search = 3,644,035 nodes

A*(h_{oop}) = 227 nodes, A*(h_{md}) = 73 nodes

d=24 IDS = too many nodes

A*(h_{oop}) = 39,135 nodes, A*(h_{md}) = 1,641 nodes

The best and worst admissible heuristics

h*(n) - the (unachievable) Oracle heuristic

- h*(n) = the true distance from the root to n

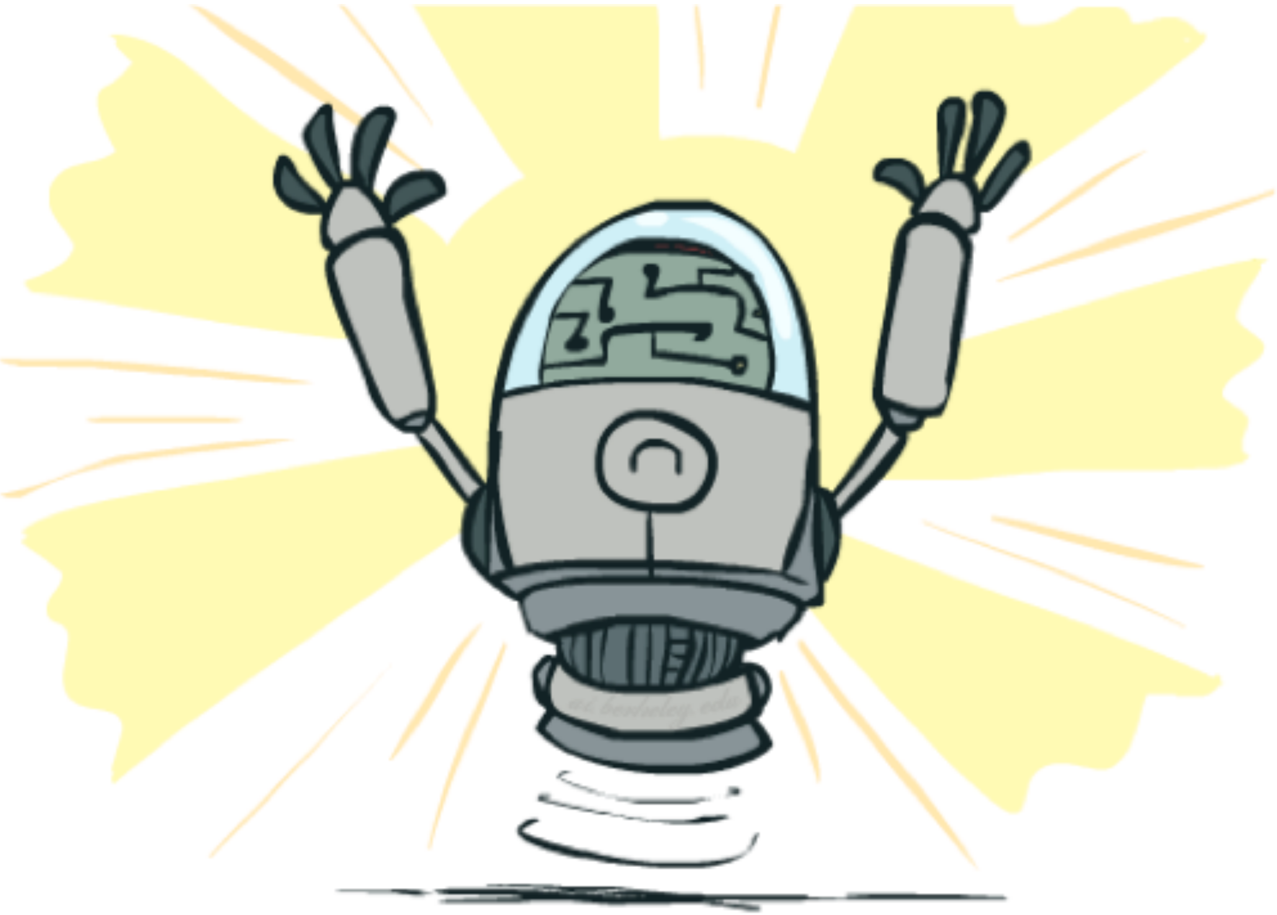
$h_{\text{we're here already}}(n) = h_{\text{teleportation}}(n) = 0$

Admissible: both yes!!!

h*(n) *dominates all other heuristics*

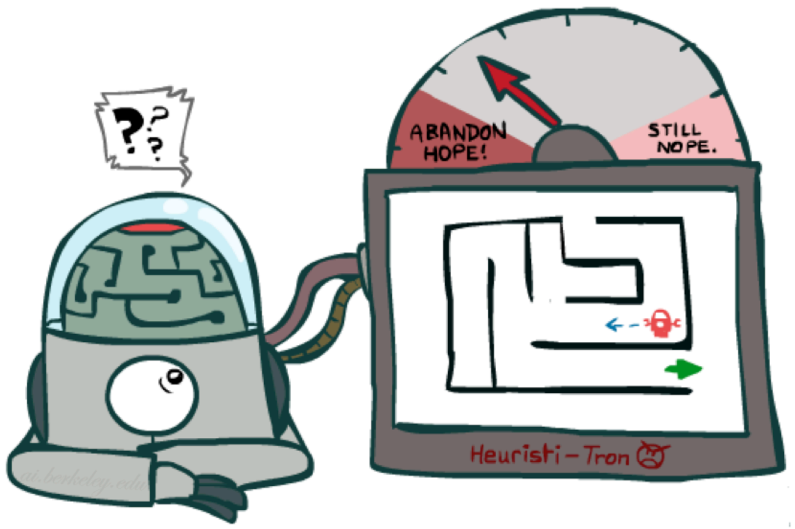
$h_{\text{teleportation}}(n)$ *is dominated by all heuristics*

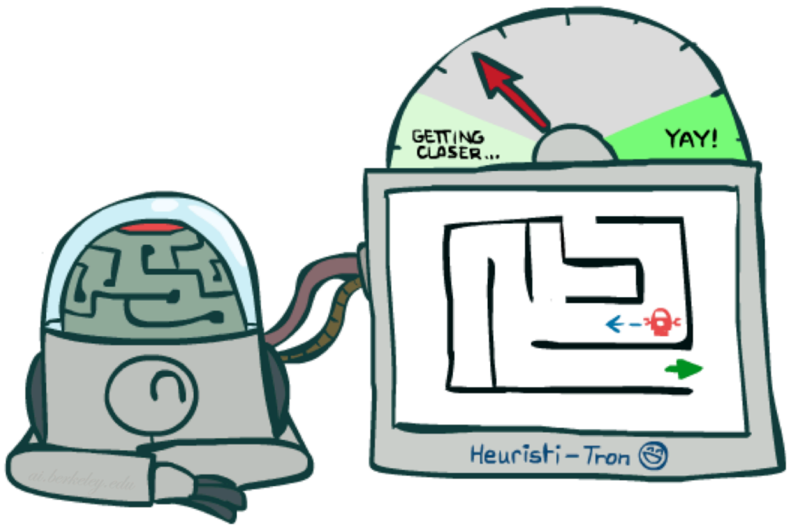
Optimality Search



Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

Idea: Admissibility





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

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

Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

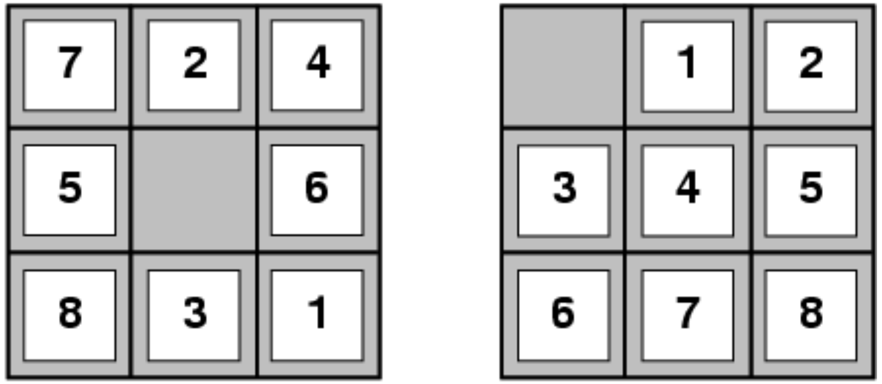
Admissible Heuristics

A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

Is Manhattan Distance admissible?



Start State Goal State

Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search

Assume:

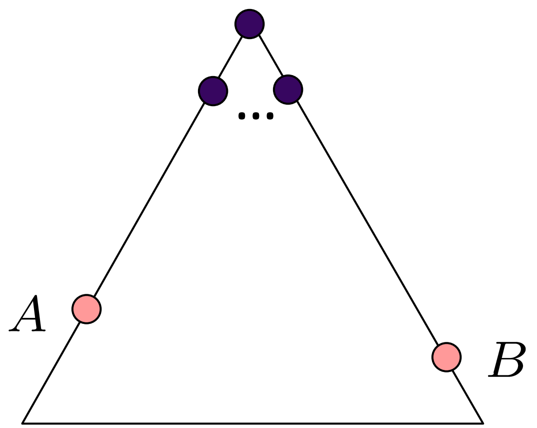
A is an optimal goal node

B is a suboptimal goal node

h is admissible

Claim:

A will exit the fringe before B



Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

Optimality of A* Tree Search

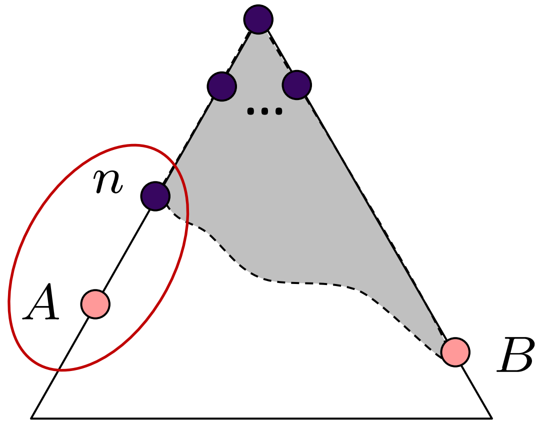
Proof:

Imagine B is on the fringe

Some ancestor *n* of A is on the fringe, too (maybe A!)

Claim: *n* will be expanded before B

- f(n) is less or equal to f(A)



$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

Optimality of A* Tree Search

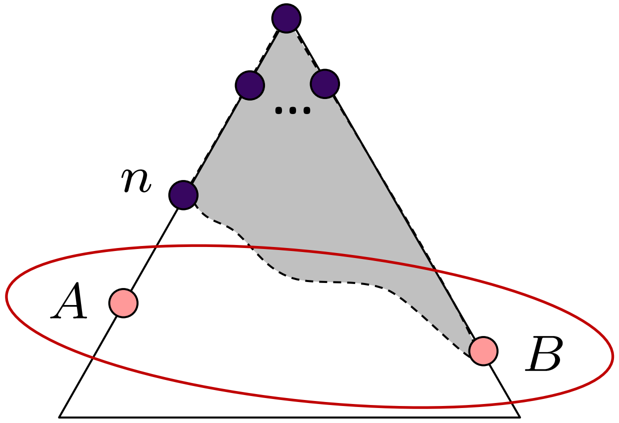
Proof:

Imagine B is on the fringe

Some ancestor *n* of A is on the fringe, too (maybe A!)

Claim: *n* will be expanded before B

- f(n) is less or equal to f(A)
- f(A) is less than f(B)



$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

Optimality of A* Tree Search

Proof:

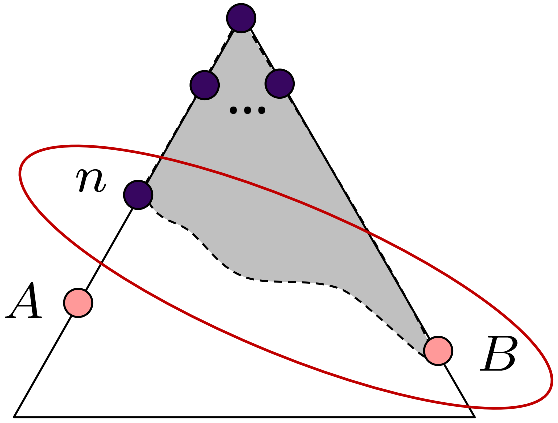
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B

  f(n) is less or equal to f(A)

  f(A) is less than f(B)

  n expands before B

- All ancestors of A expand before B
- A expands before B
- A* search is optimal



$$f(n) \leq f(A) < f(B)$$

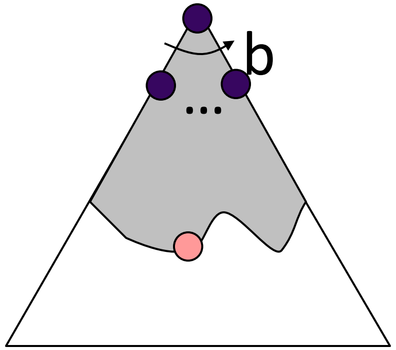
Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

Properties of A*

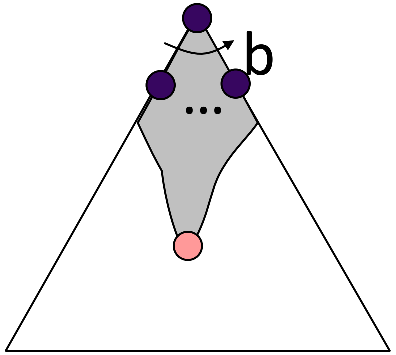
Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

Properties of A*

Uniform-Cost



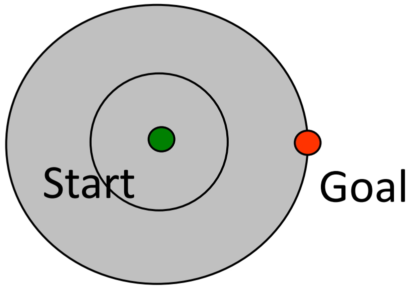
A*



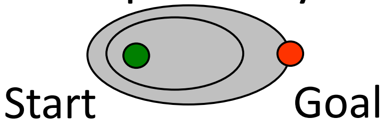
Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

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



Slide credit: Dan Klein and Pieter Abbeel
http://ai.berkeley.edu

A* Applications

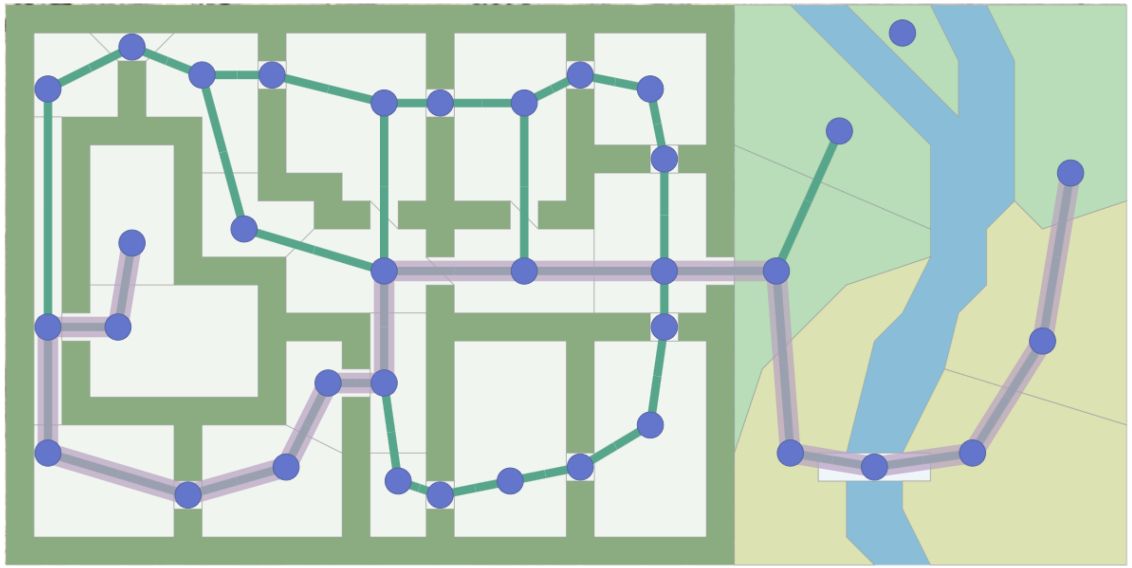
Pathing / routing problems (A* is in your GPS!)

Video games

Robot motion planning

Resource planning problems

…



Supplemental Reading

I recommend this A* tutorial by Amit Patel of Red Blob Games

https://www.redblobgames.com/pathfinding/a-star/introduction.html

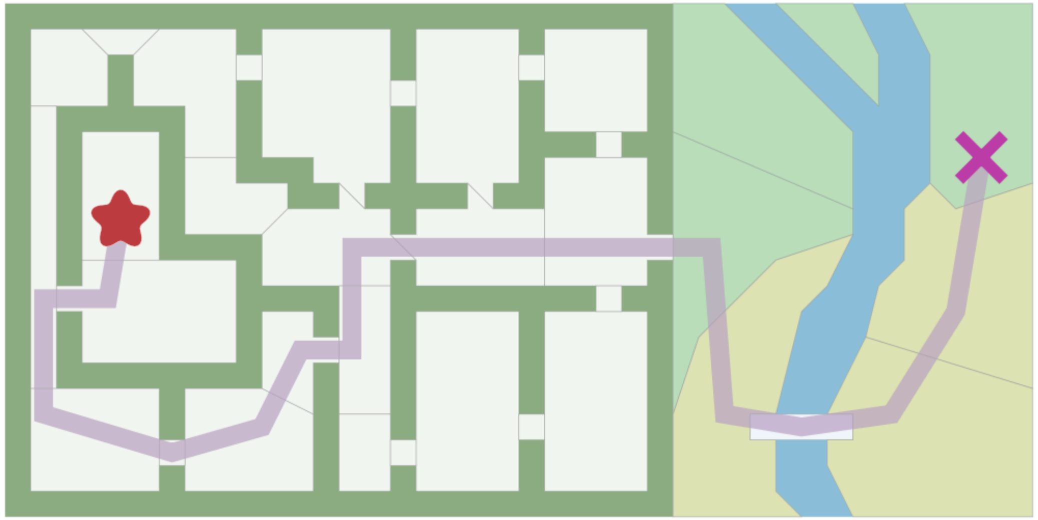
Introduction to the A* Algorithm

from Red Blob Games

Home Blog Links Twitter About Search

Created 26 May 2014, updated Aug 2014, Feb 2016, Jun 2016

In games we often want to find paths from one location to another. We're not only trying to find the shortest distance; we also want to take into account travel time. Move the blob (start point) and cross (end point) to see the shortest path.



To find this path we can use a *graph search* algorithm, which works when the map is represented as a graph. **A*** is a popular choice for graph search. **Breadth First Search** is the simplest of the graph search algorithms, so let's start there, and we'll work our way up to A*.