Introduction to Artificial Intelligence

IFT3335 Lecture 4: Informed Search

Bang Liu, Jian-Yun Nie

2 Certain Slides Adapted From or Referred To…

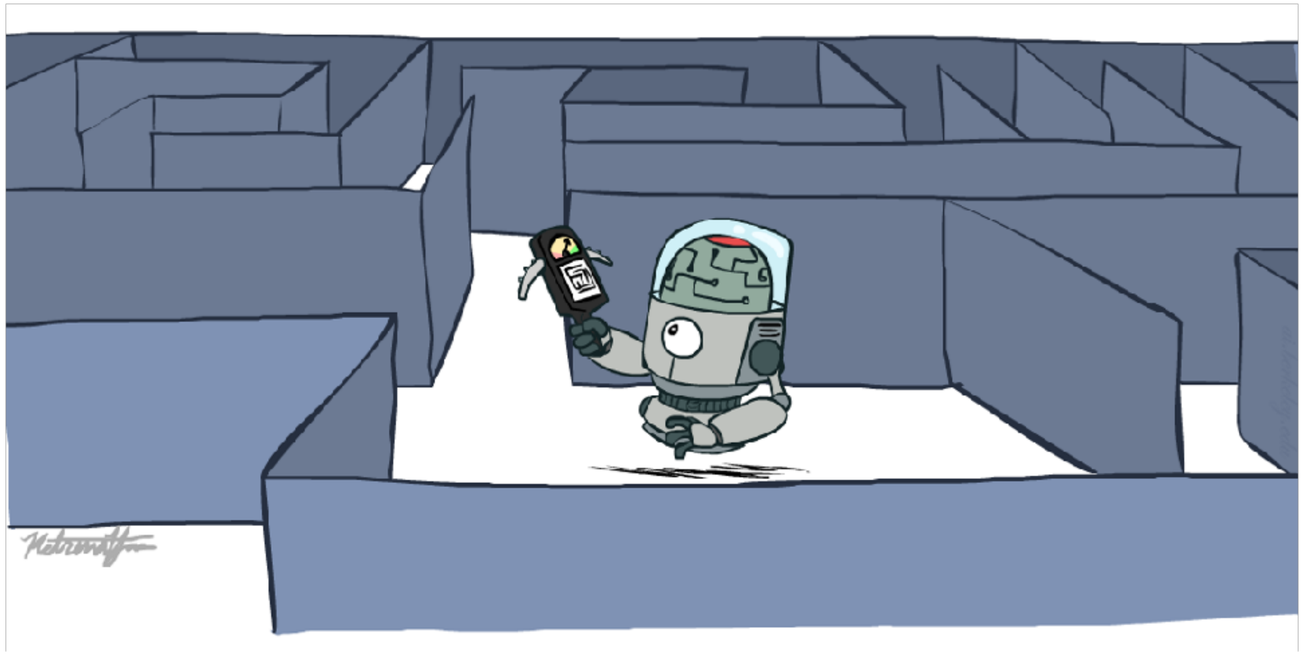
- Slides from **UC Berkeley CS188, Dan Klein and Pieter Abbeel**
  - A* Search and Heuristics: https://inst.eecs.berkeley.edu/~cs188/su21/

- Slides from **UPenn CIS391, Mitch Marcus**
  - Informed Search: https://www.seas.upenn.edu/~cis391/#LECTURES

3 Plan

- Informed search: use problem-specific knowledge
- A* search: optimal search using knowledge
- Heuristics

Informed Search

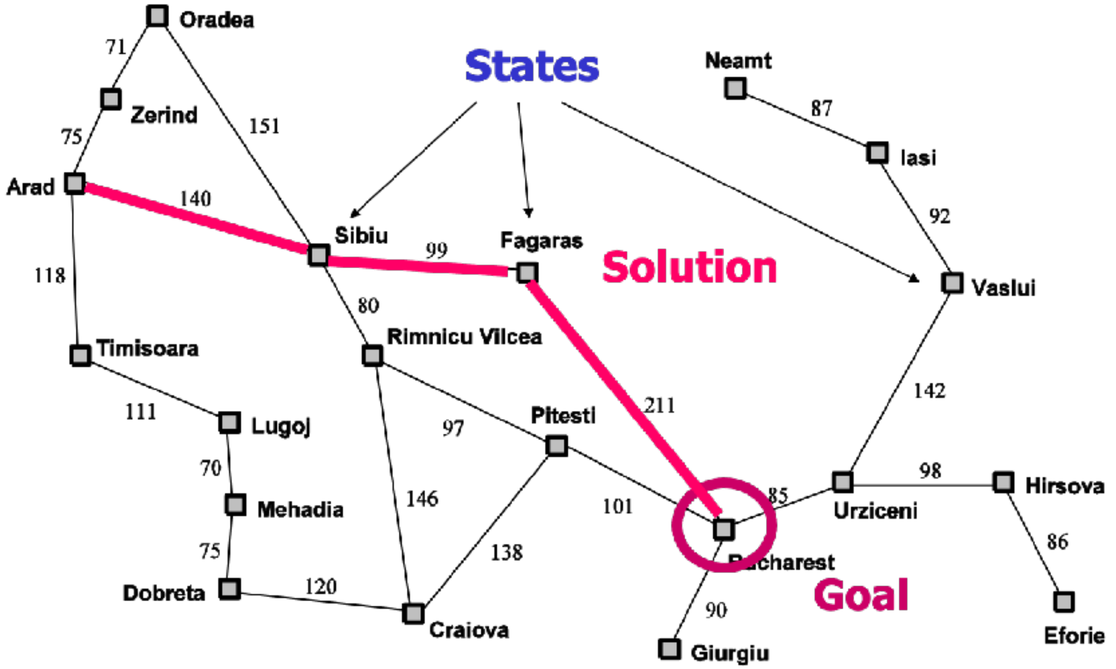


Recap: Search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test

- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)

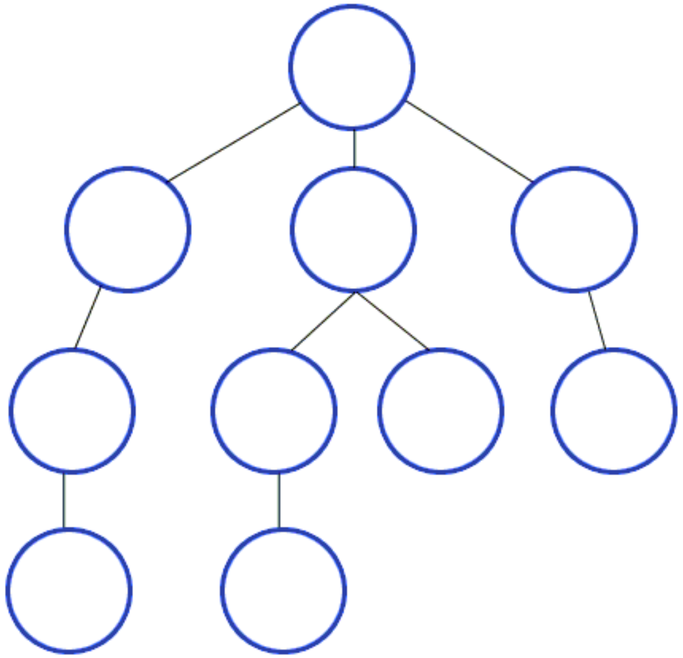
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plan



Example: route finding

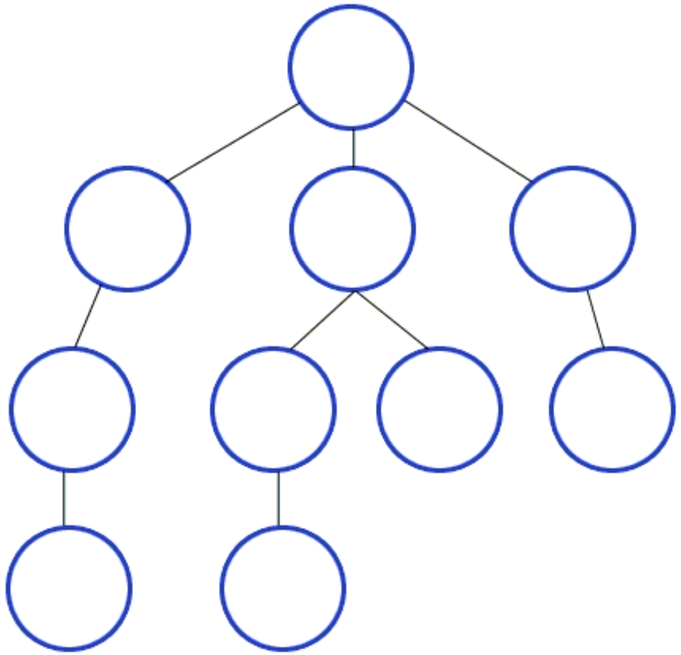
6 Recap: Breadth-First Search (BFS)

- BFS first visit all the nodes at the same depth first and then proceed visiting nodes at a deeper depth (strategy: expand a shallowest node first)



Recap: Depth-First Search (DFS)

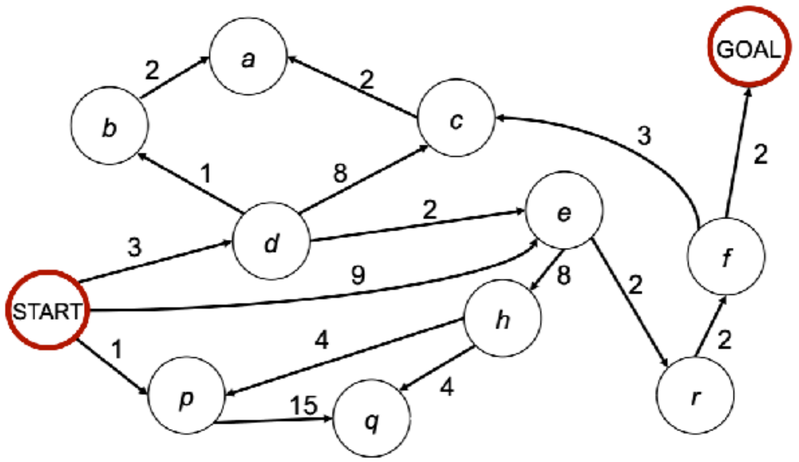
- DFS first explores the depth of the graph before the breadth i.e., it traverses along the increasing depth and upon reaching the end, it backtracks to the node from which it was started and then do the same with the sibling node.

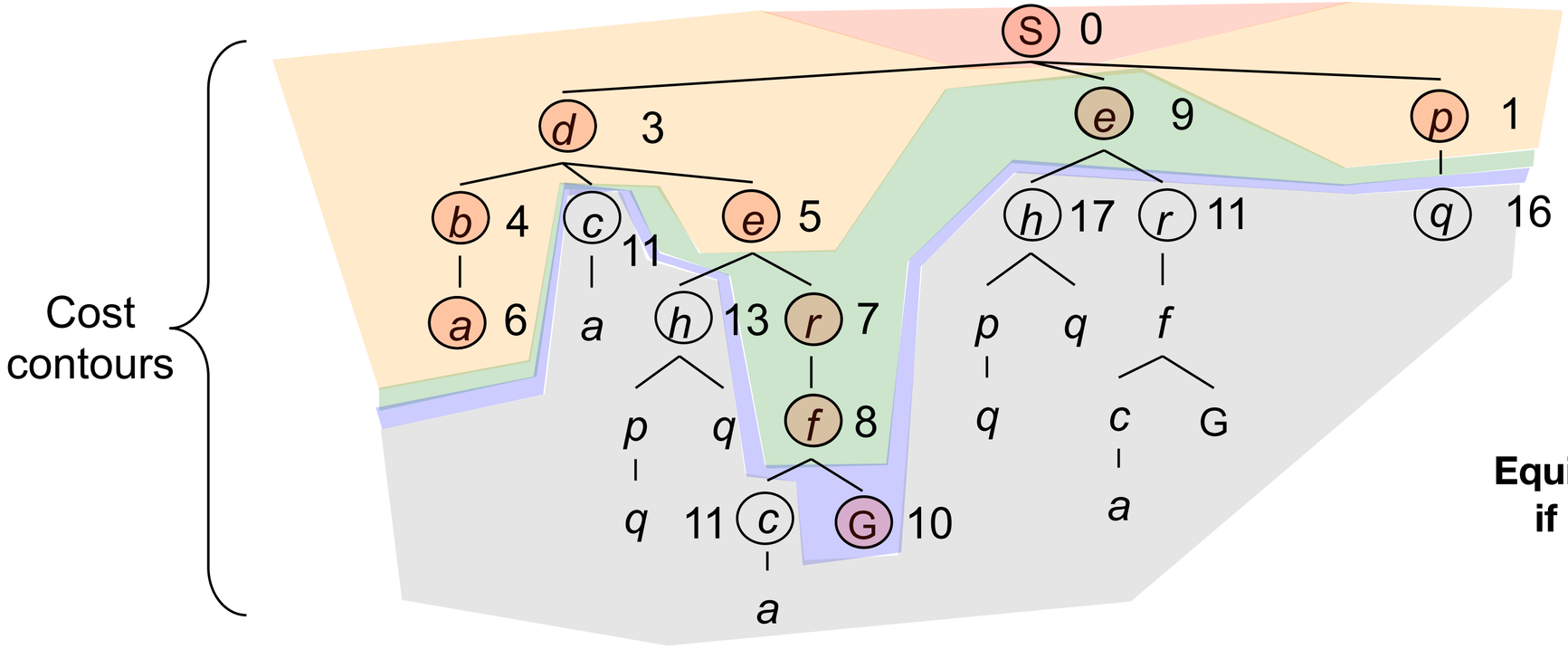


8 Recap: Uniform-Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

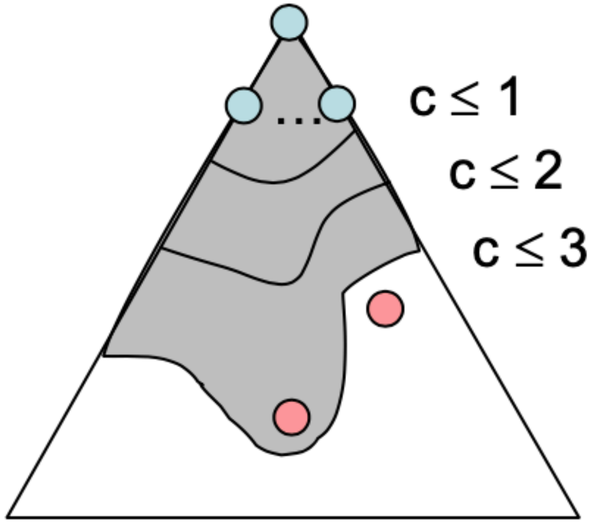


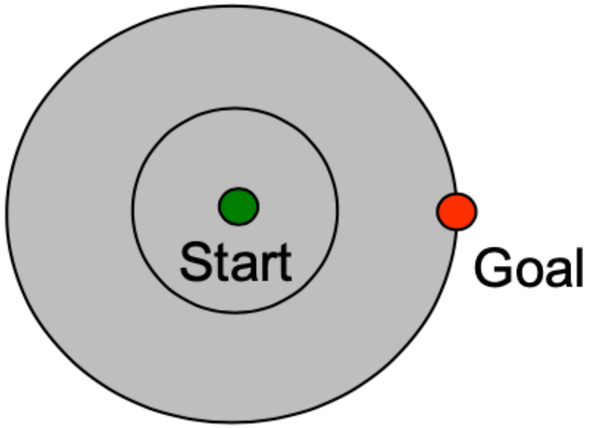


Equivalent to breadth-first if step costs all equal.

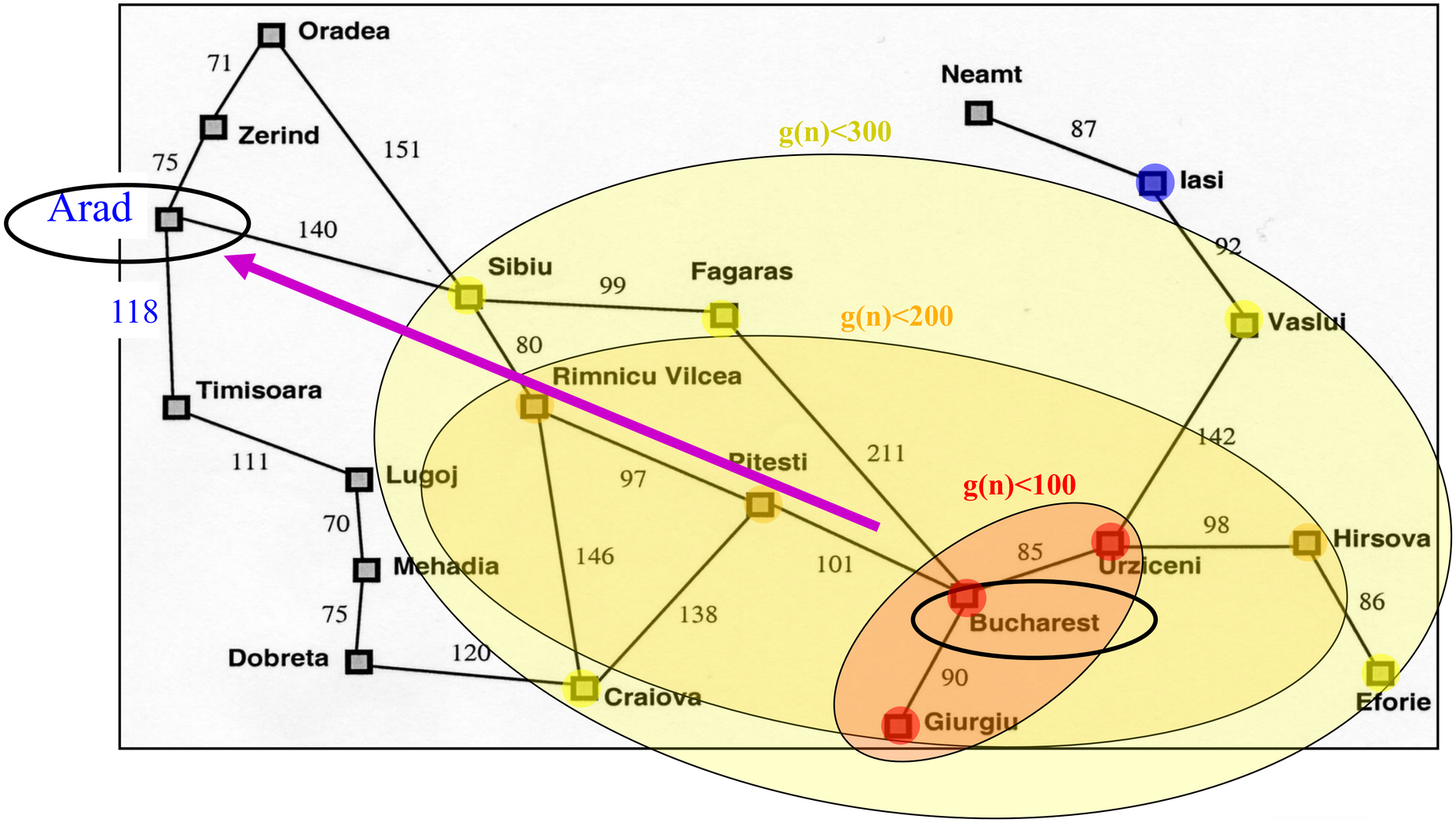
9 Is Uniform-Cost Search the best we can do?

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location

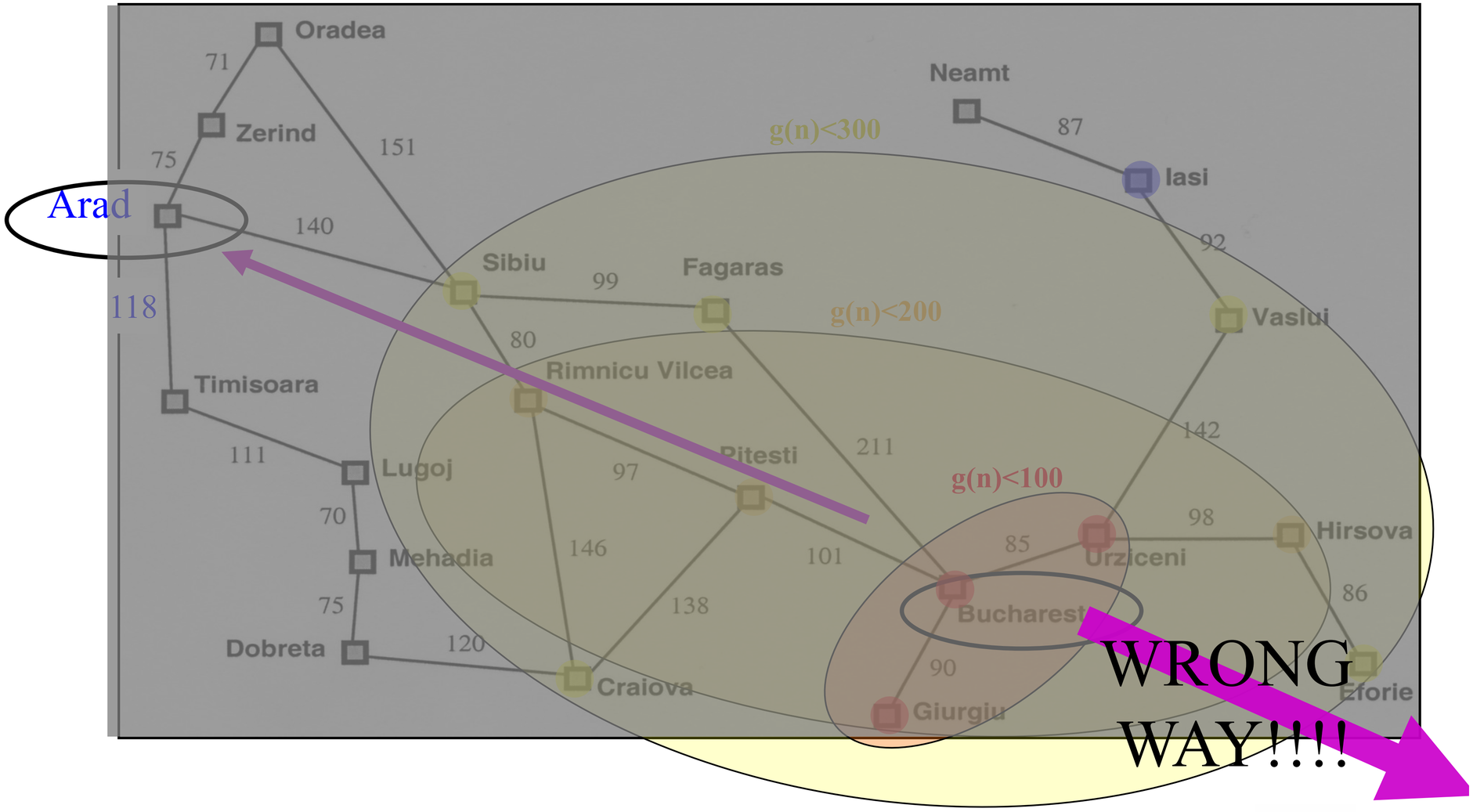




10 Consider finding a route from Bucharest to Arad..



11 Consider finding a route from Bucharest to Arad..



A Better Idea: Use Information about Goal Location

- **Idea:**
  - Node expansion based on an **estimate** which includes **distance to the goal**

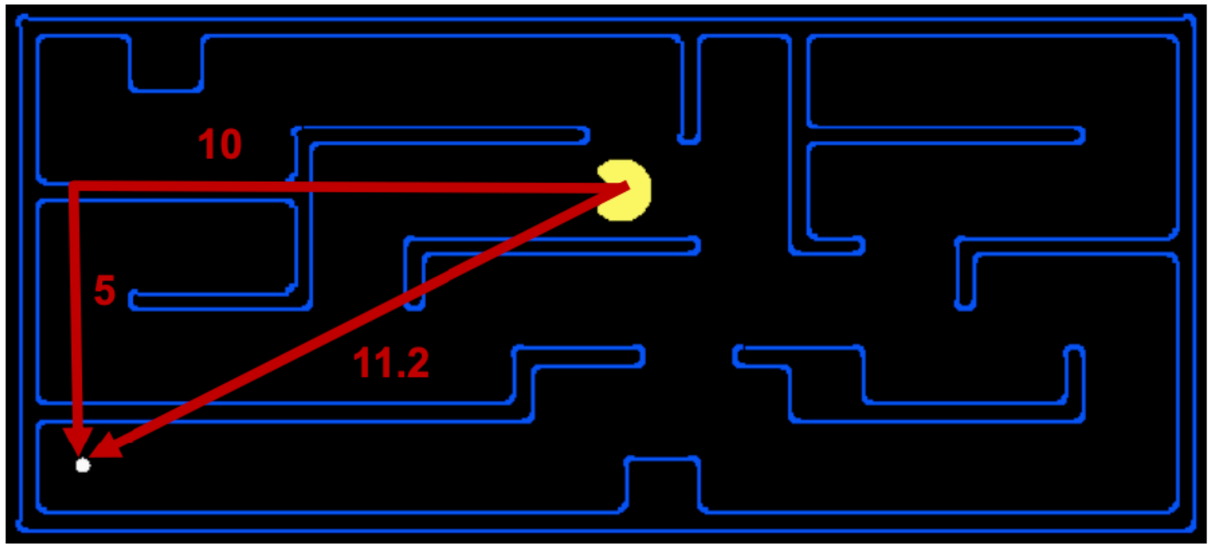
- **General approach of informed search:**
  - **Best-first search**: node selected for expansion base on an **evaluation function** ***f(n)***
  - ***f(n)*** includes estimate of distance to goal (new idea!)

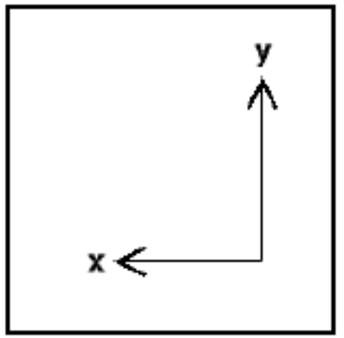
- **Implementation:**
  - Sort frontier queue by this new ***f(n)***.

- **Special cases:**
  - Greedy best-first search
  - **A* search**

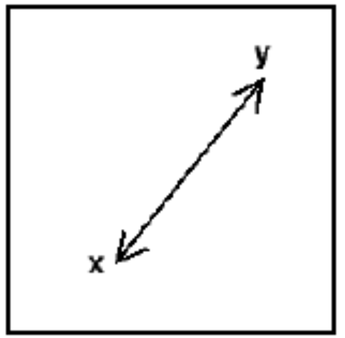
13 Search Heuristics

- A heuristic is:
  - A function *h(n)* that estimates how close a state *n* is to a goal.
    Heuristic knowledge is useful, but not necessarily correct.
  - Heuristic algorithms use heuristic knowledge to solve a problem.
  - Examples: Manhattan distance, Euclidean distance for pathing



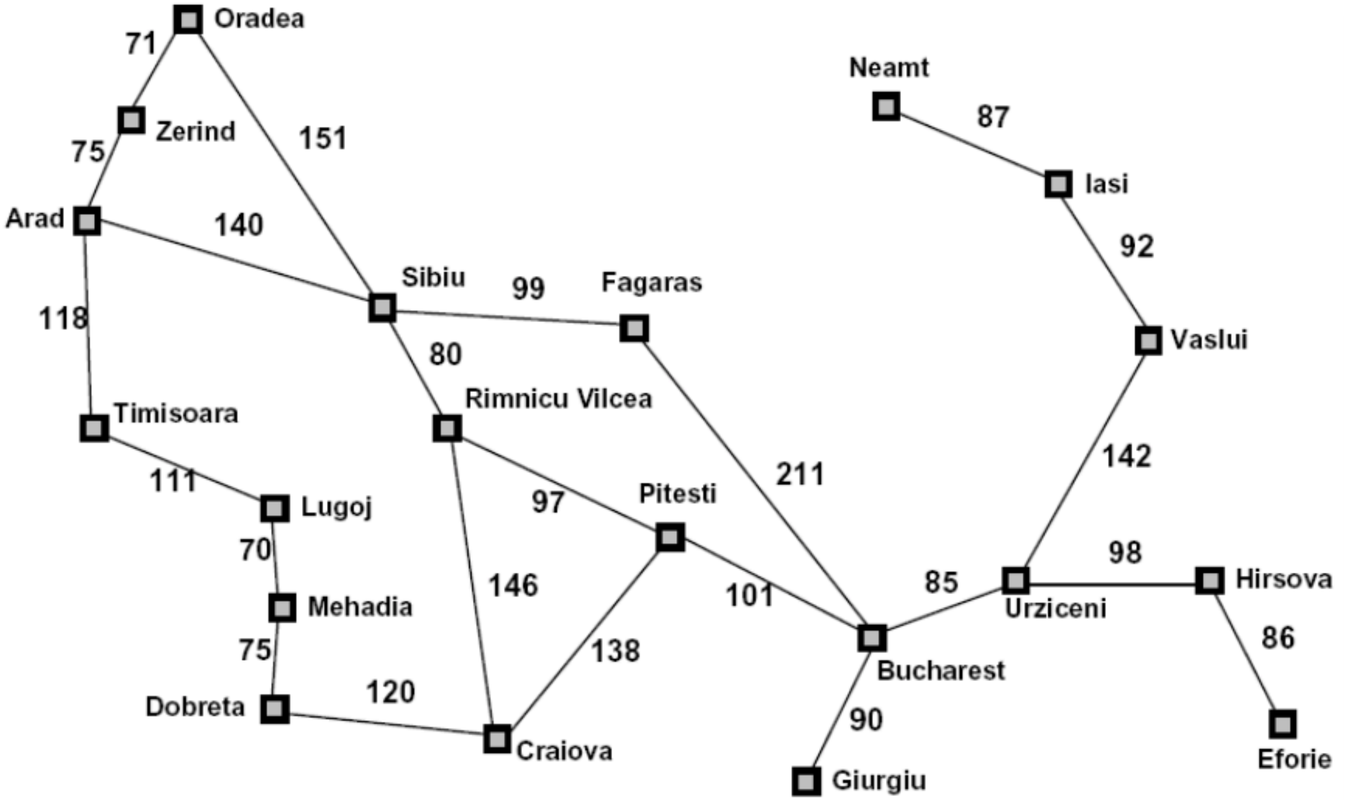


Manhattan



Euclidean

Example: Heuristic Function



Straight−line distance to Bucharest	
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

h(x)

Greedy Search

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

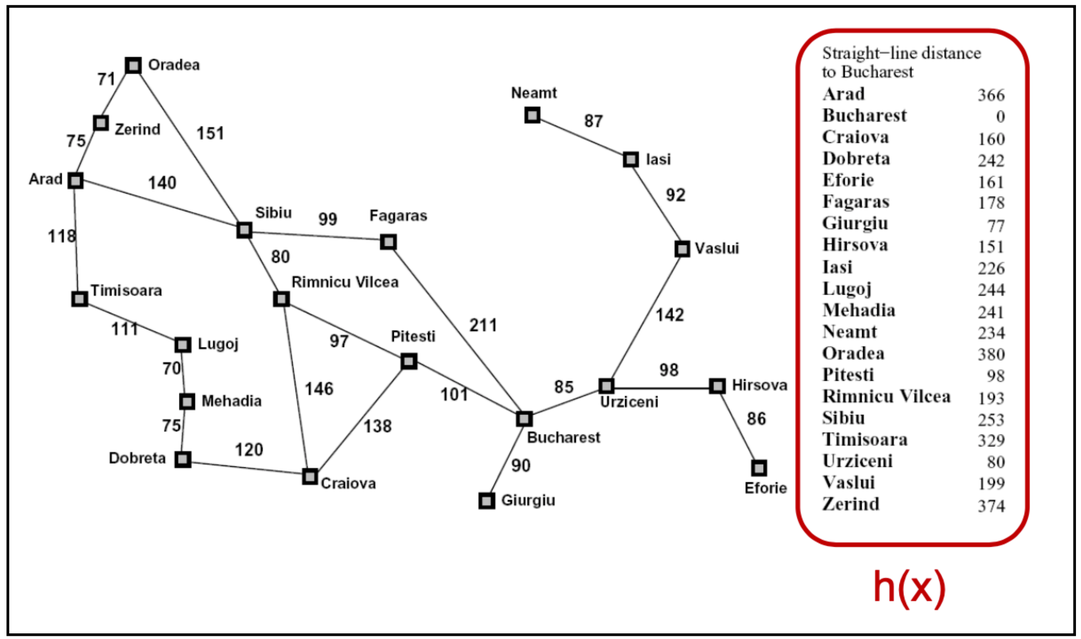
- Exactly UCS with ***f(n)*** replacing ***g(n)***

Greedy Best-First Search: $f(n) = h(n)$

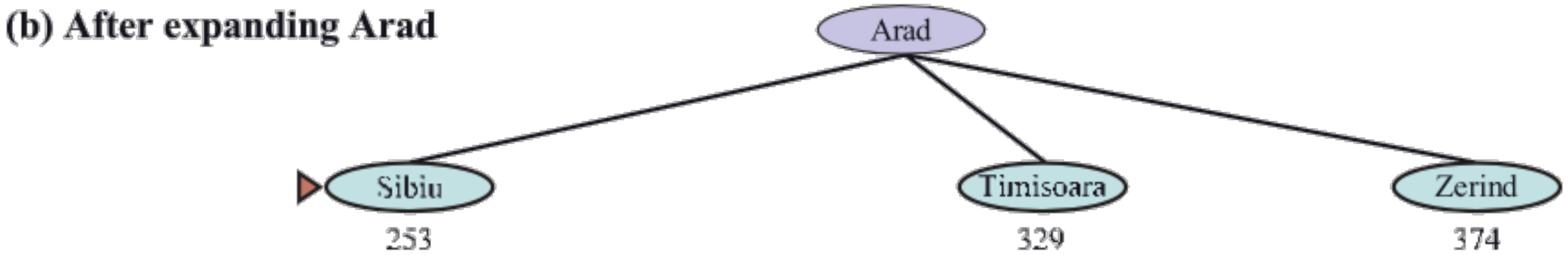
- Expands the node that **is estimated** to be closest to goal
- Completely ignores $g(n)$: the cost to get to n
- Here, $h(n) = h_{SLD}(n) =$ *straight-line distance* from n to Bucharest

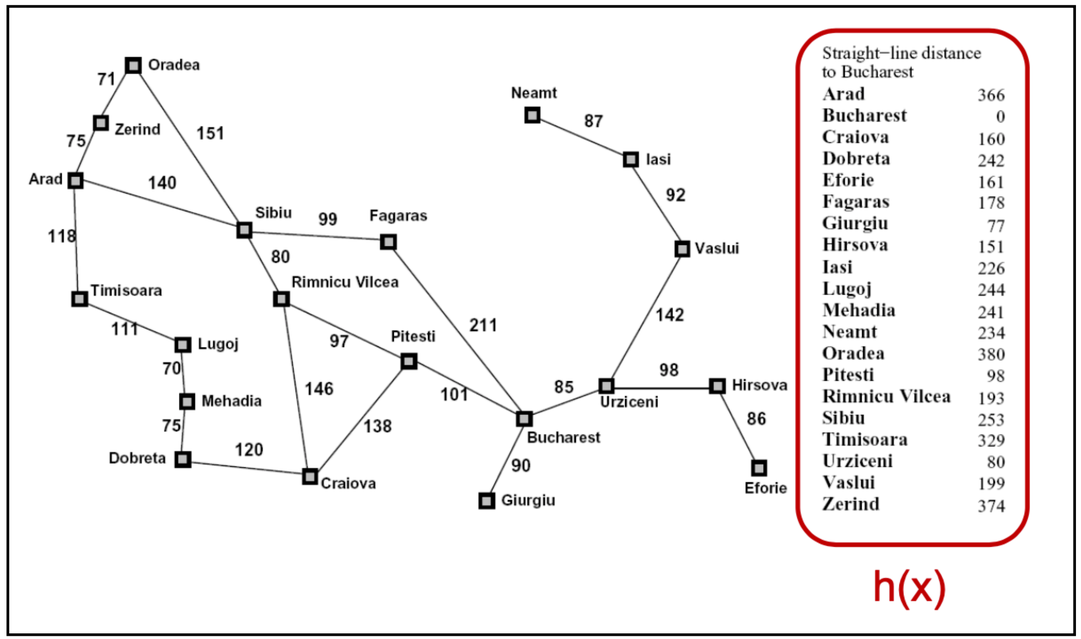
Greedy best-first search example

(a) The initial state

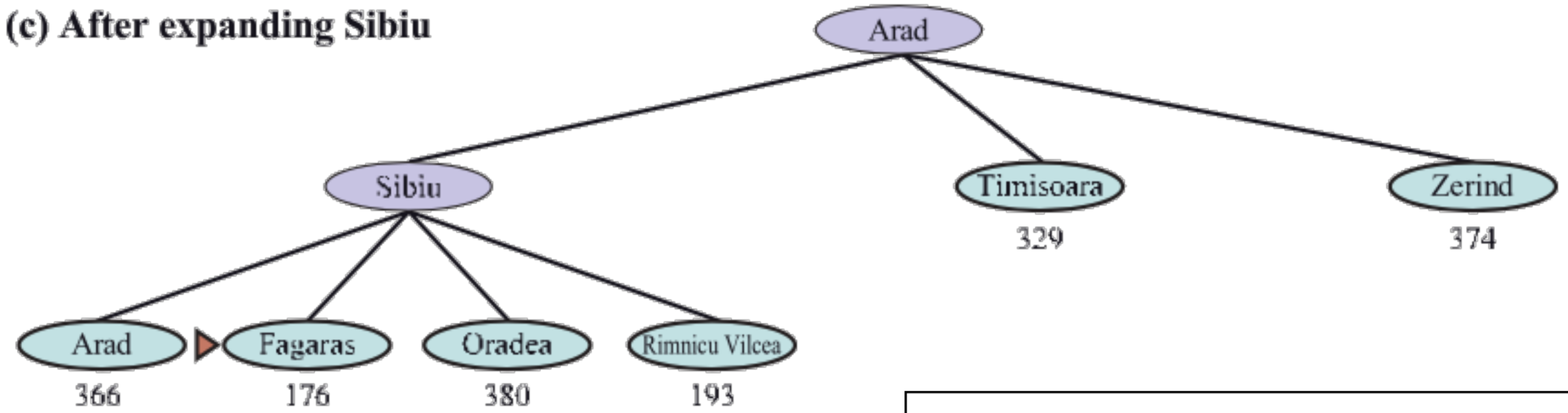


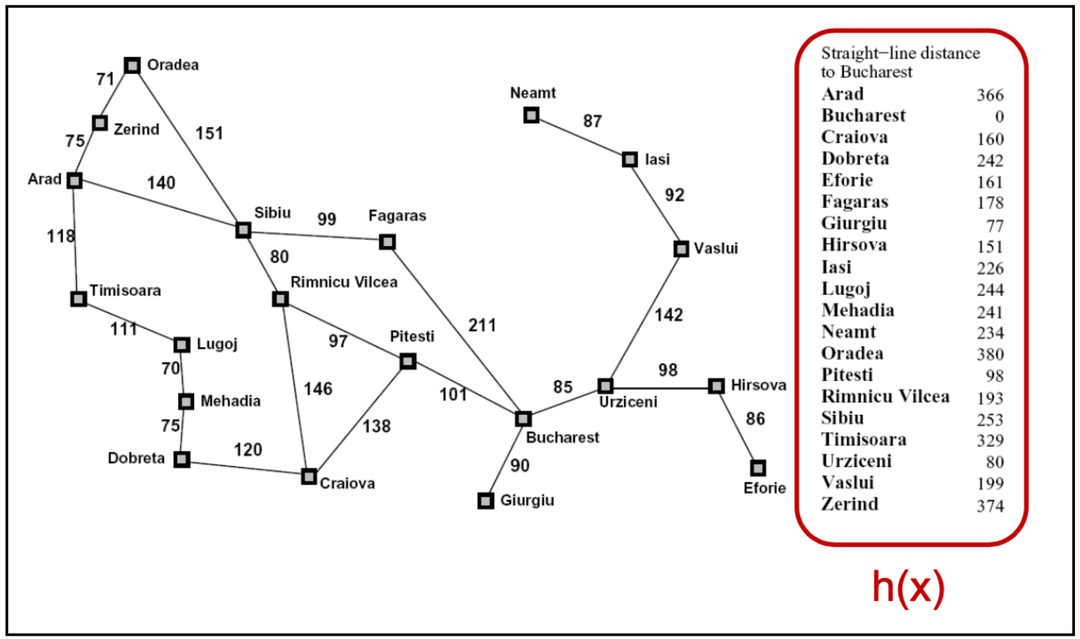
19 Greedy best-first search example



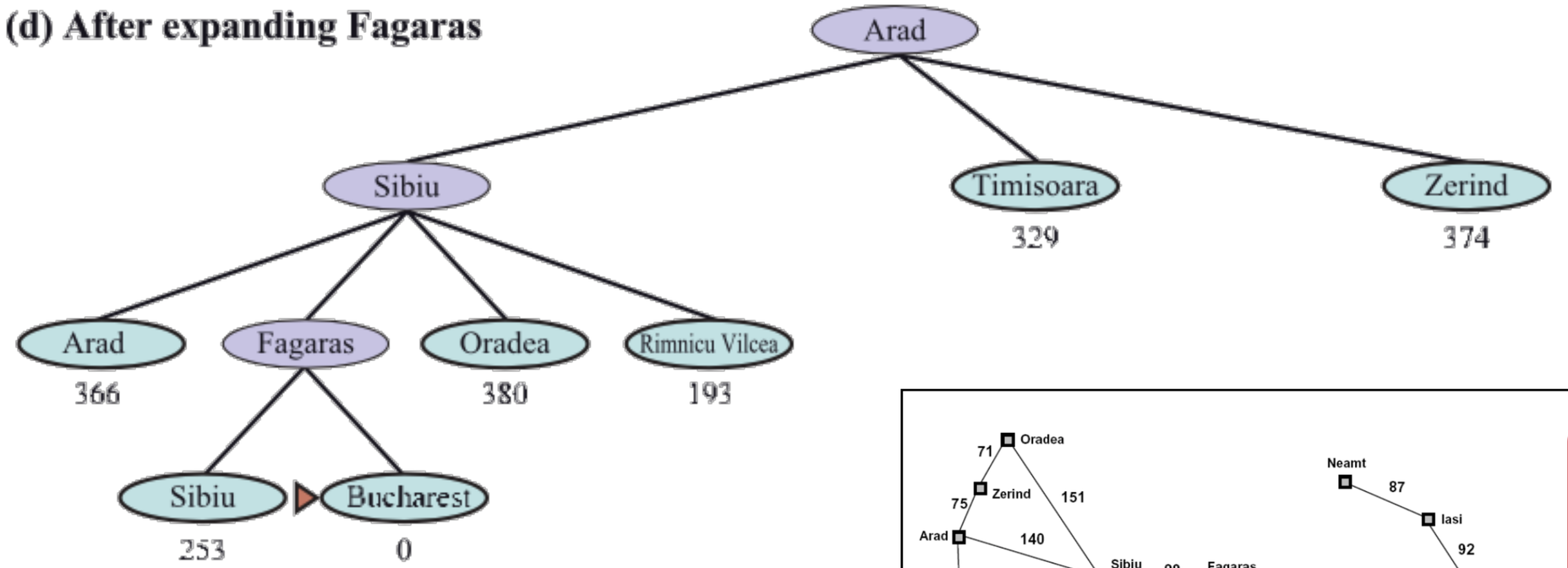


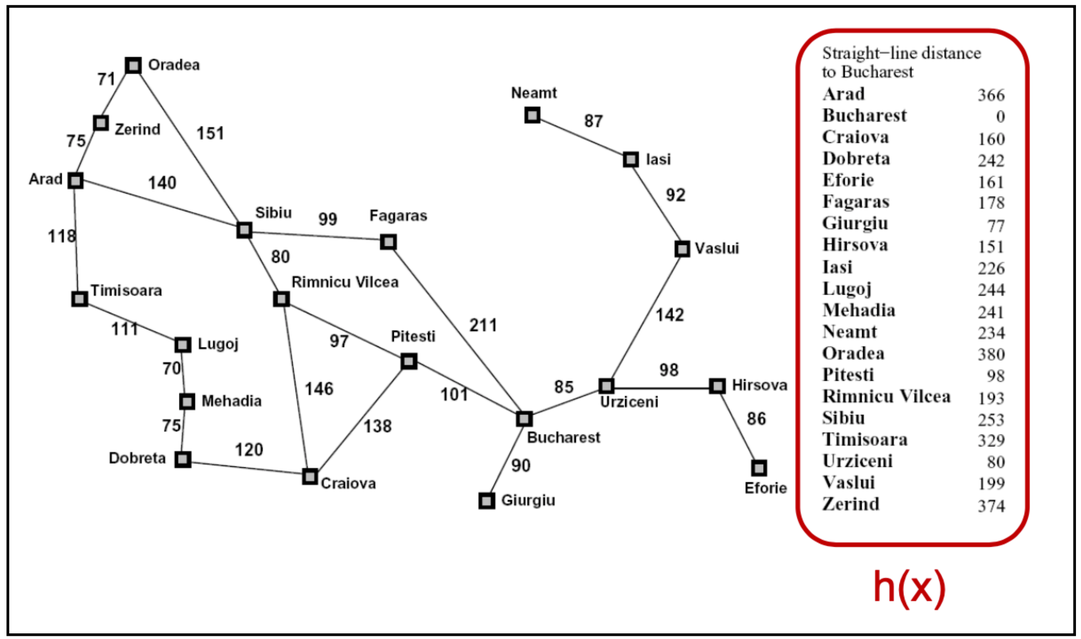
20 Greedy best-first search example





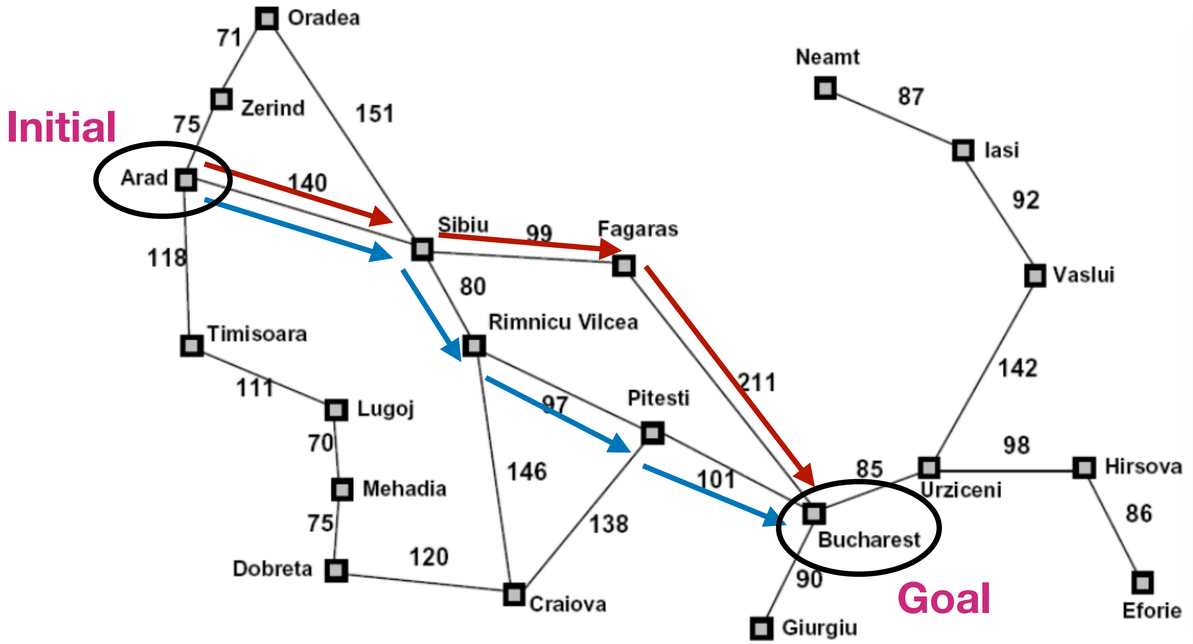
21 Greedy best-first search example





Properties of greedy best-first search

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

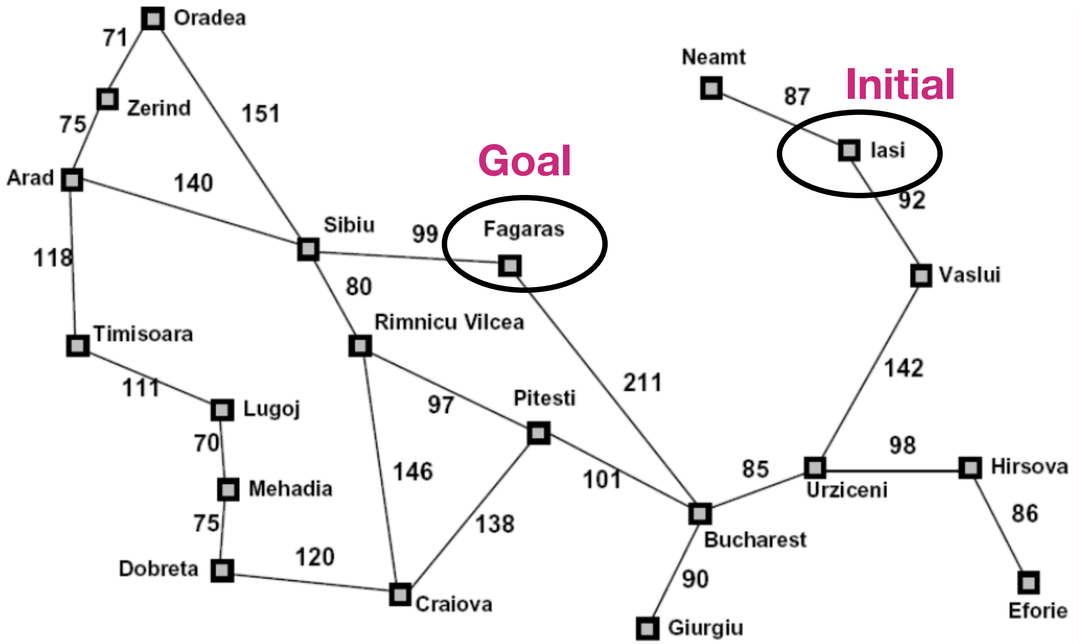


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

23 Properties of greedy best-first search

- **Complete? No!**
  - Can get stuck in loops, e.g., *Iasi*→*Neamt*→*Iasi*→*Neamt*→...



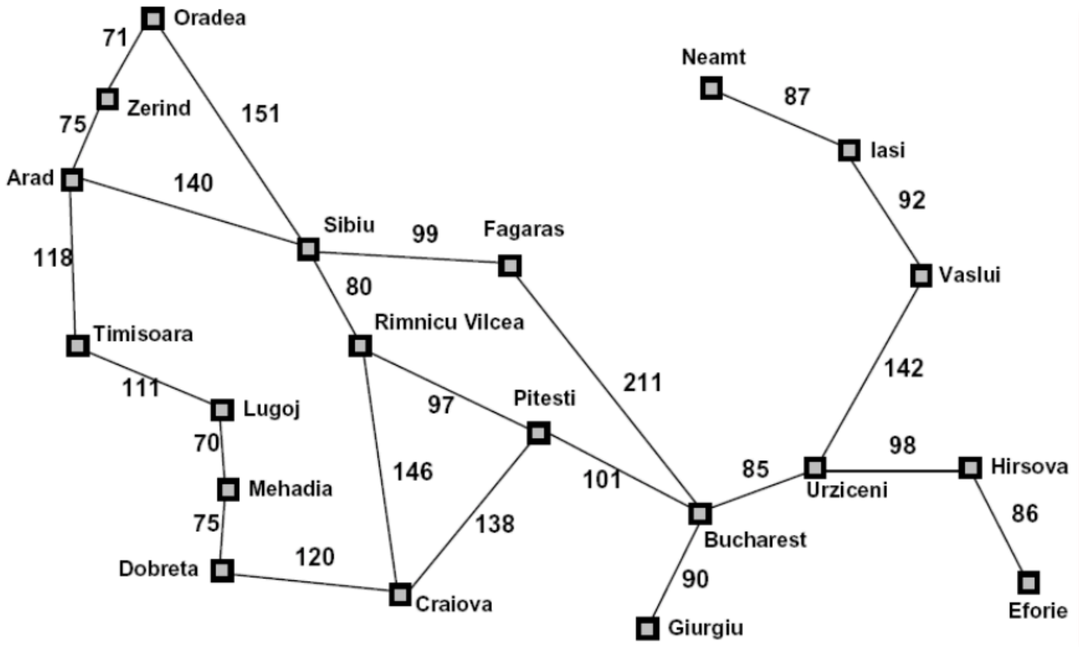
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

24 Properties of greedy best-first search

- **Time?** $O(b^m)$ – worst case (like Depth First Search)
  - But a good heuristic can give dramatic improvement of average cost

- **Space?** $O(b^m)$ – priority queue, so worst case: keeps all (unexpanded) nodes in memory



Straight−line distance to Bucharest	
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

h(x)

A* Search

A* Search: Combining UCS and Greedy

- **Best-known form of best-first search.**

- **Key Idea:** avoid expanding paths that are already expensive, but expand most promising first.
  - **Uniform-cost** orders by path cost, or backward cost ***g(n)***, the cost so far to reach the node ***n***
  - **Greedy** orders by goal proximity, or forward cost ***h(n)***, the estimated cost to get from the node ***n*** to the goal
  - **A* Search** orders by the sum: ***f(n) = g(n) + h(n)***, the estimate total cost of path through ***n*** to goal

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

A* Search Example

Frontier queue:

Arad 366

(a) The initial state

366=0+366

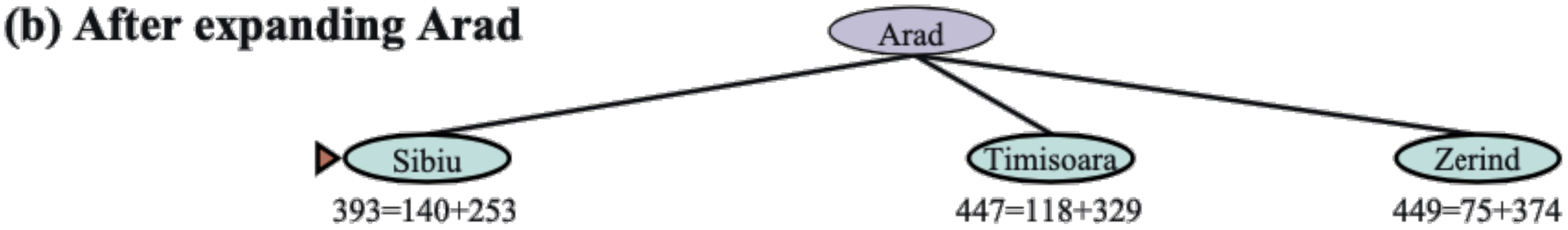
A* Search Example

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

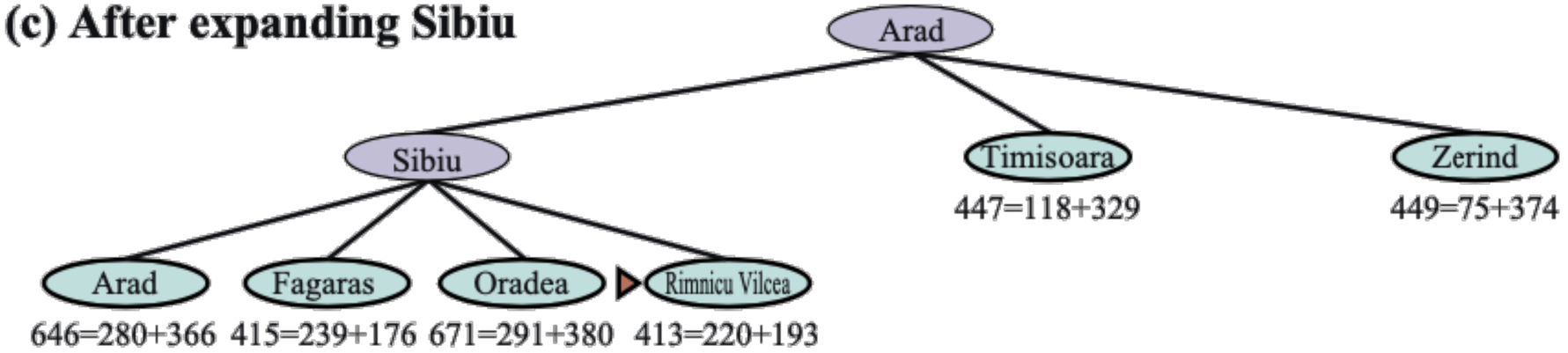
Fagaras 415

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

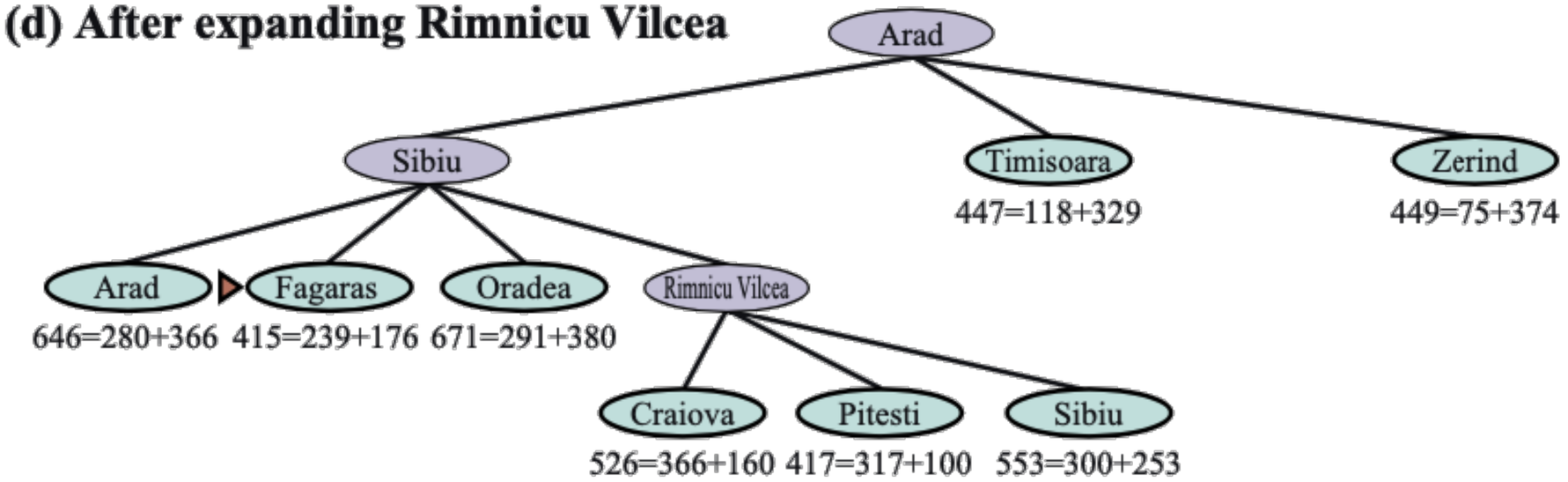
Zerind 449

Craiova 526

Sibiu 553

Arad 646

Oradea 671



We expand Rimnicu Vilcea.

A* Search Example

Frontier queue:

Pitesti 417

Timisoara 447

Zerind 449

Bucharest 450

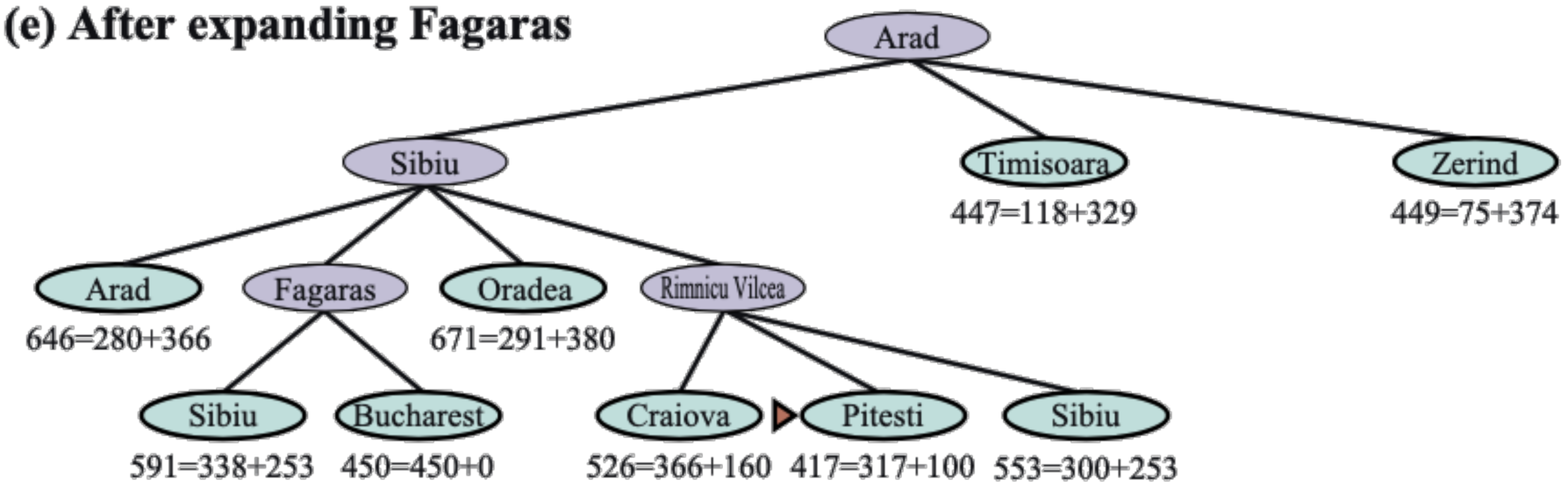
Craiova 526

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. In another words:

A* doesn't stop when enqueue a goal. It stops when dequeue a goal.

A* Search Example

Frontier queue:

Bucharest 418

Timisoara 447

Zerind 449

Bucharest 450

Craiova 526

Sibiu 553

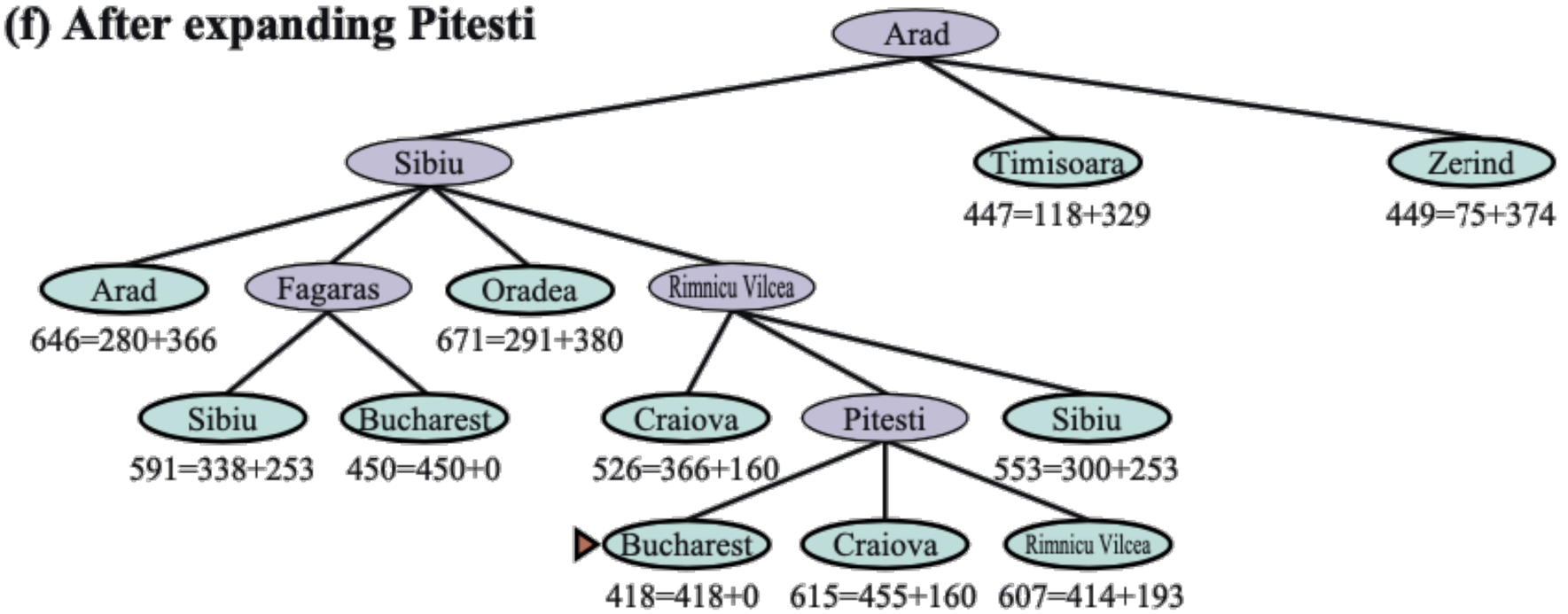
Sibiu 591

Rimnicu Vilcea 607

Craiova 615

Arad 646

Oradea 671

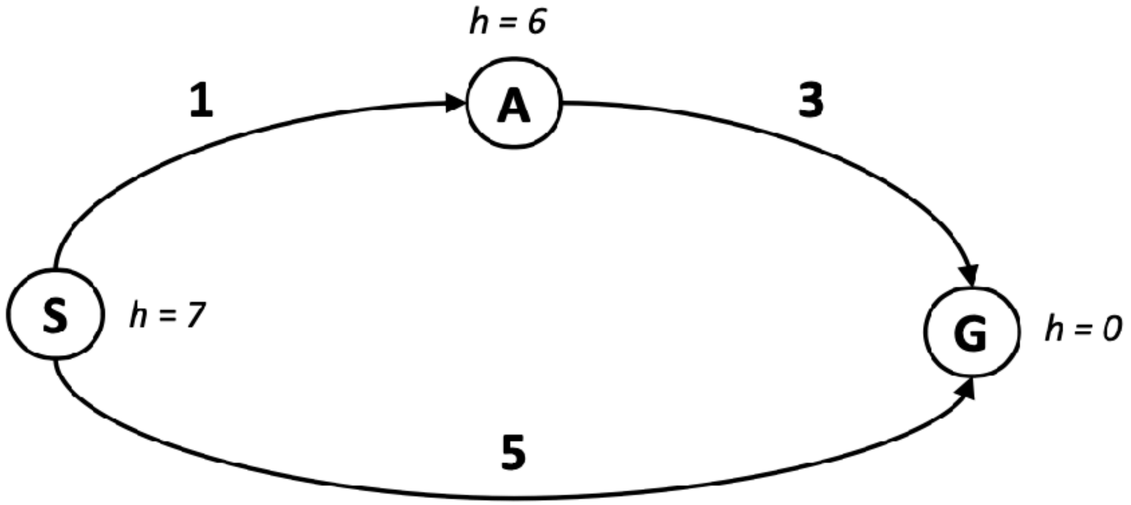


Note that we just found a better value for Bucharest!

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!

33 Is A* Optimal?



- What went wrong?
- Actual bad goal cost < estimated good goal cost
- **We need estimates to be less than actual costs!**

Admissible Heuristics

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

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

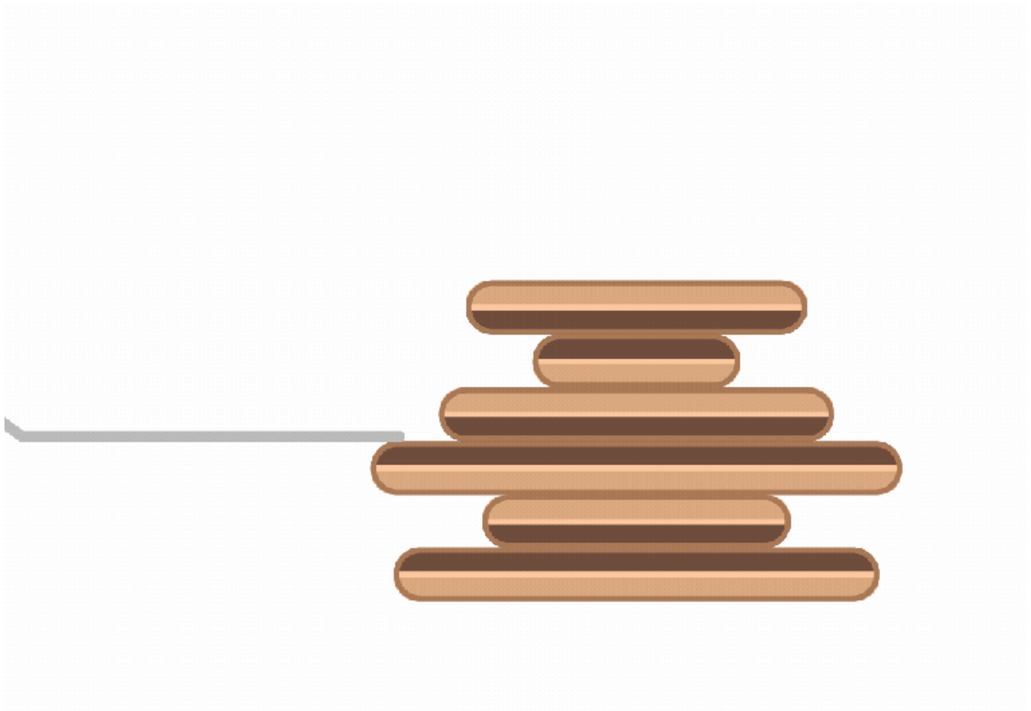
  where $h^*(n)$ is the true cost to a nearest goal

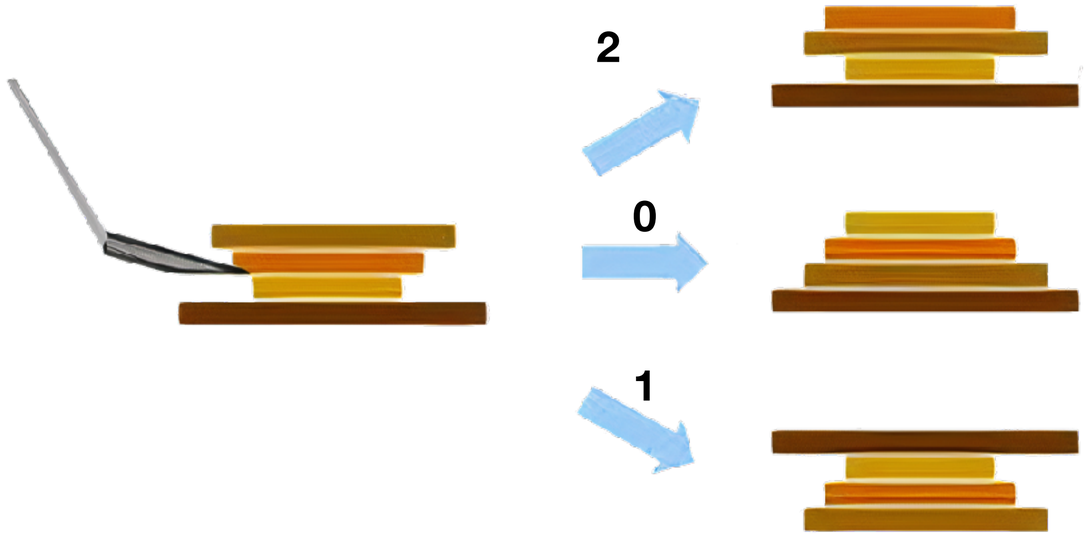
- Inadmissible (or pessimistic because they overestimate the cost) heuristics break optimality by trapping good plans on the fringe.

- Admissible (or optimistic because they can only underestimate the cost) heuristics can only slow down search by assigning a lower cost to a bad plan so that it is explored first but sooner or later A* will find the optimal solution.

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

Example of Admissible Heuristics



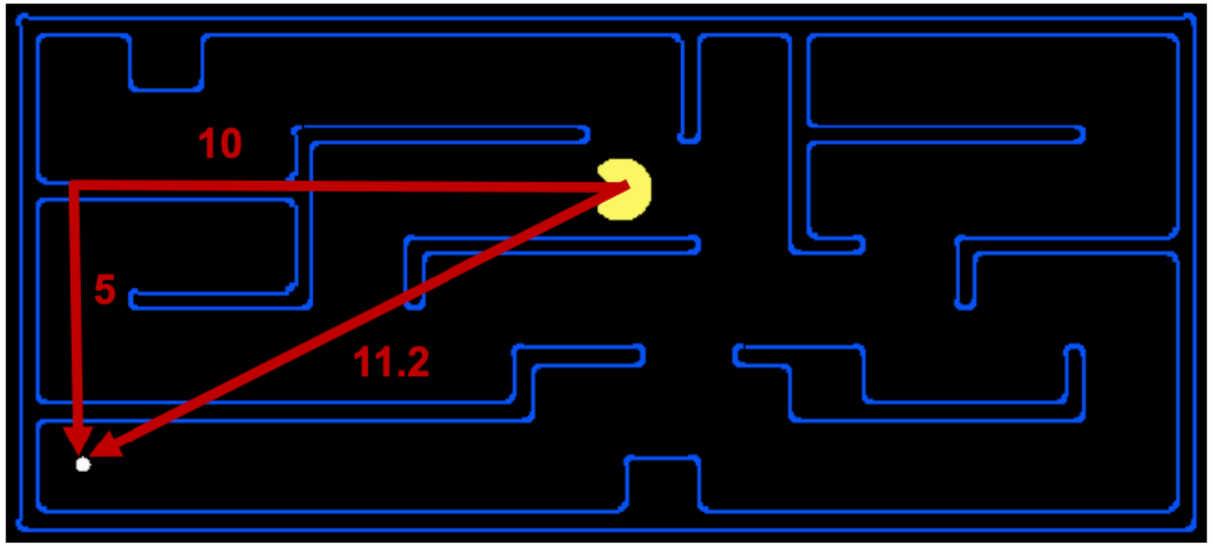


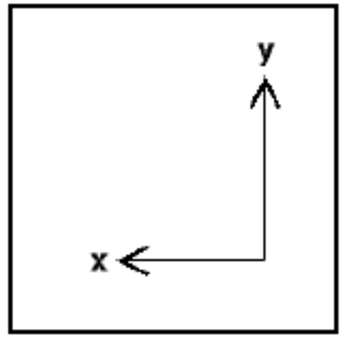
Pancake sorting problem: sorting a disordered stack of pancakes in order of size when a spatula can be inserted at any point in the stack and used to flip all pancakes above it

Cost: Number of pancakes flipped

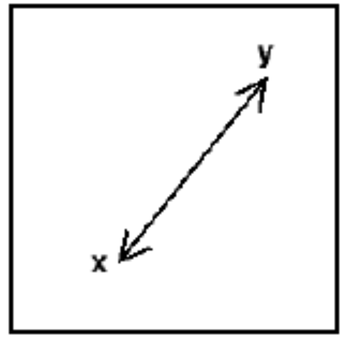
The gap heuristic function: a gap occurs whenever two adjacent pancakes in a stack are not adjacent in the sorted stack; this computation includes the plate as a pancake. The gap heuristic function then counts the number of gaps in the stack.

Example of Admissible Heuristics





Manhattan

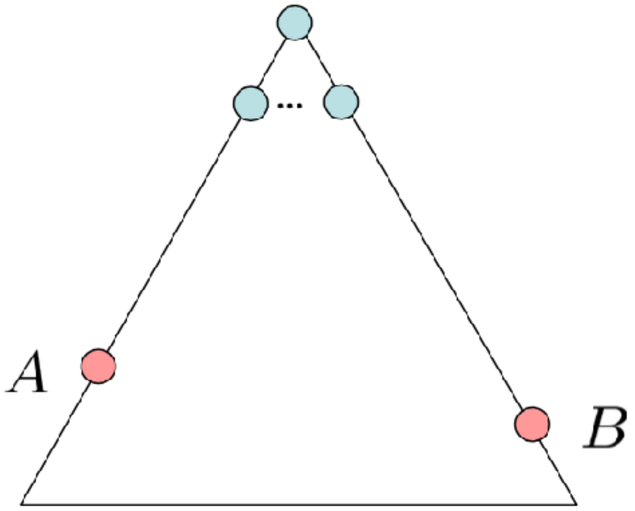


Euclidean

Optimality of A* Tree Search

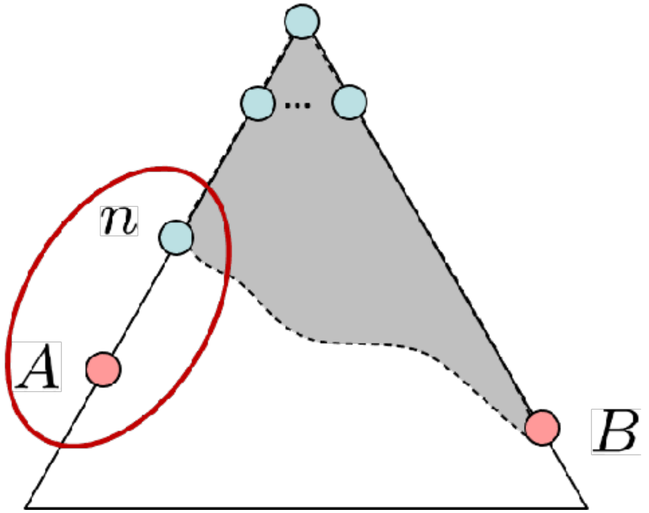
- Assume:
  - A is an optimal goal node
  - B is a suboptimal goal node
  - *h* is admissible

- Claim:
  - A will exit the fringe before B



Optimality of A* Tree Search: Blocking

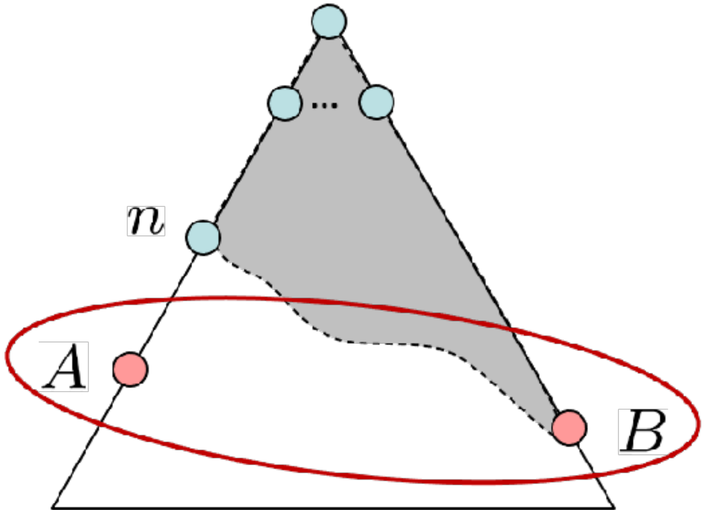
- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim: n will be expanded before B
    1. f(n) is less or equal to f(A)



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

Optimality of A* Tree Search: Blocking

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

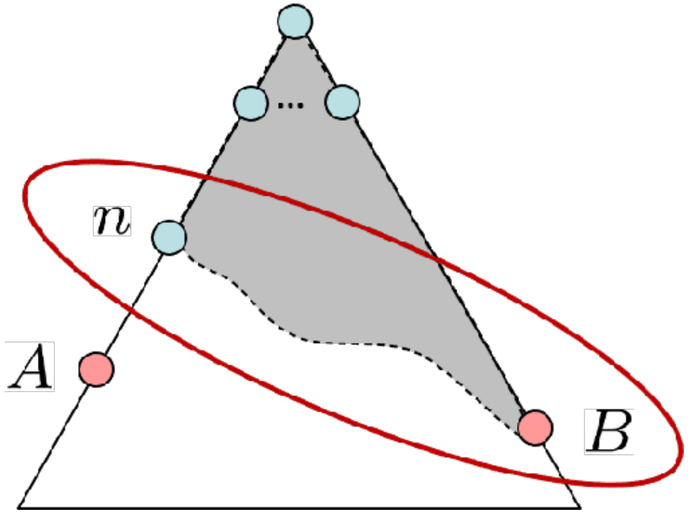


$g(A) < g(B)$ B is suboptimal

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

Optimality of A* Tree Search: Blocking

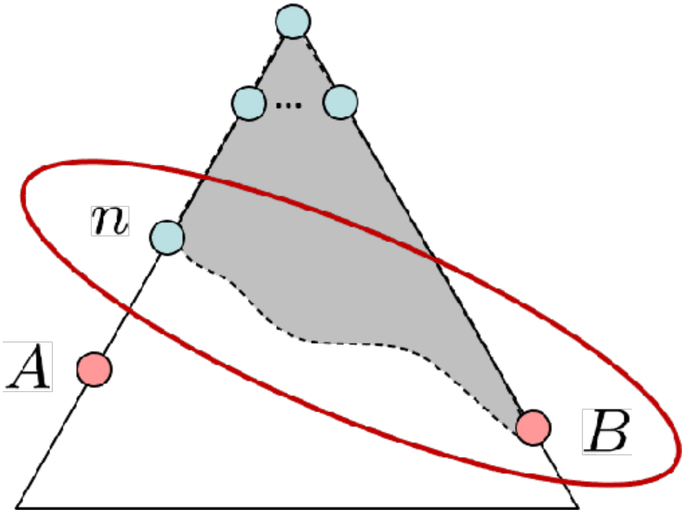
- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim: n will be expanded before B
    1. f(n) is less or equal to f(A)
    2. f(A) is less than f(B)
    3. n expands before B



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

Optimality of A* Tree Search: Blocking

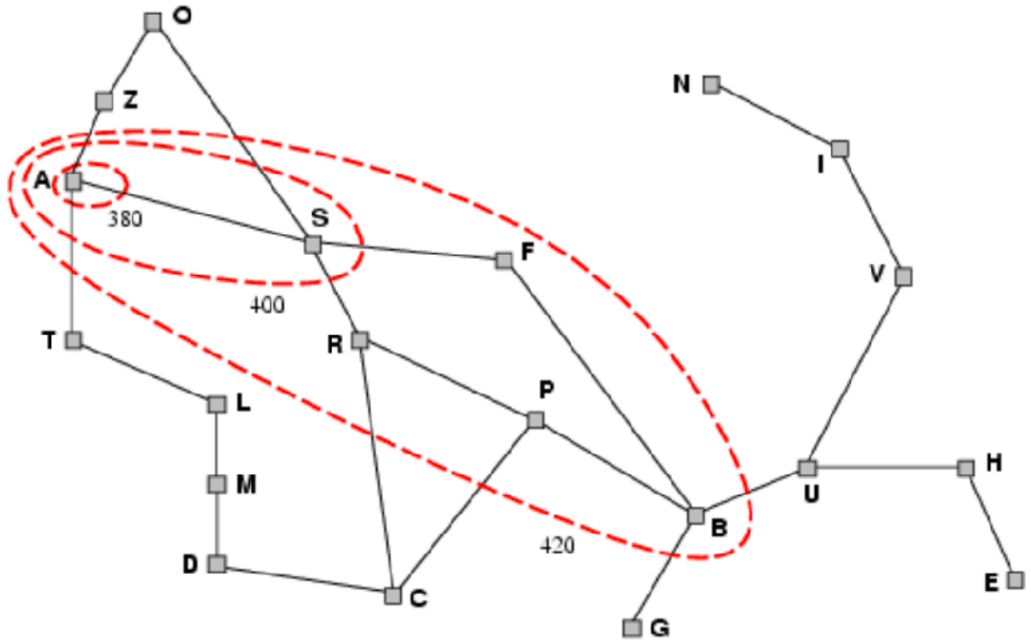
- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim: n will be expanded before B
    1. f(n) is less or equal to f(A)
    2. f(A) is less than f(B)
    3. n expands before B
  - All ancestors of A expand before B
  - A expands before B
  - **A* search is optimal**



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

Optimality of A* (intuitive)

- **Lemma:** A* expands nodes on frontier in order of increasing f value
  - Gradually adds "f-contours" of nodes
  - Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$
  - (After all, A* is just a variant of uniform-cost search....)



Property of A*

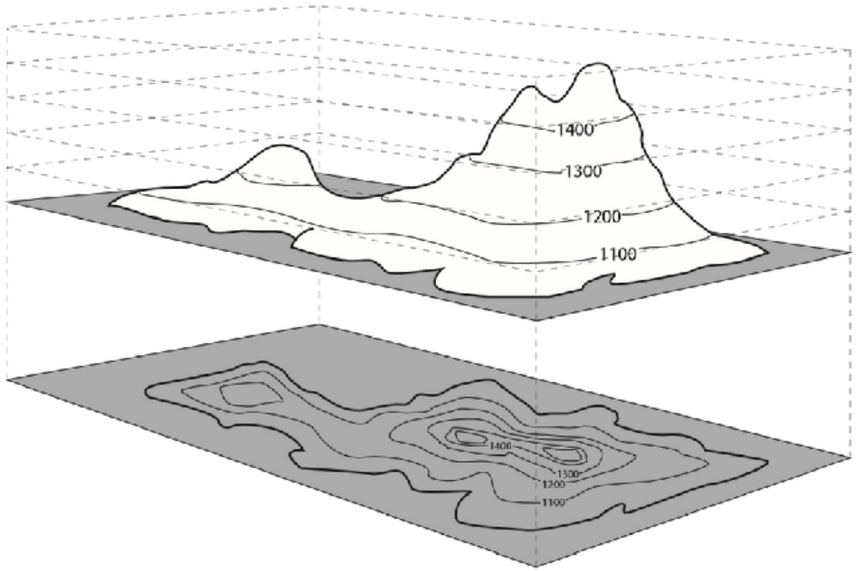
- **Optimality: Yes**

- **Completeness: Yes**
  - Since bands of increasing f are added
  - As long as b is finite (guaranteeing that there aren't infinitely many nodes n with $f(n) < f(G)$)

- **Time complexity:**
  - Number of nodes expanded is still exponential in the length of the solution.

- **Space complexity:**
  - It keeps all generated nodes in memory
  - Hence space is the major problem not time

44 BFS vs UCS vs A* Contours



Contours

Breadth First Search explores equally in all directions. This is an incredibly useful algorithm, not only for regular path finding, but also for procedural map generation, flow field pathfinding, distance maps, and other types of map analysis.

Dijkstra's Algorithm (also called Uniform Cost Search) lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favors lower cost paths. We can assign lower costs to encourage moving on roads, higher costs to avoid forests, higher costs to discourage going near enemies, and more. When movement costs vary, we use this instead of Breadth First Search.

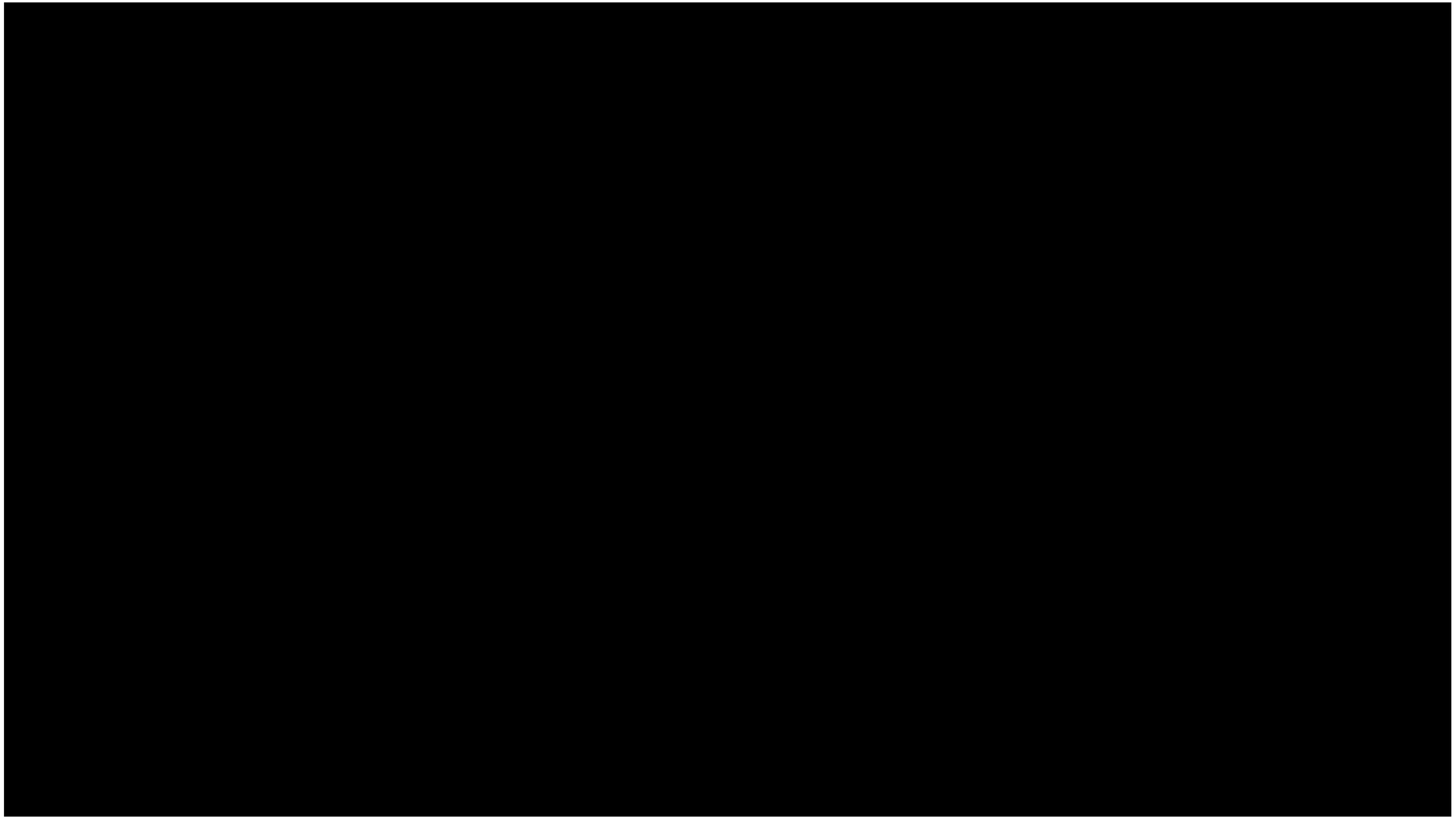
A* is a modification of Dijkstra's Algorithm that is optimized for a single destination. Dijkstra's Algorithm can find paths to all locations; A* finds paths to one location, or the closest of several locations. It prioritizes paths that seem to be leading closer to a goal.

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

In the Maze:
Breadth First Search Algorithm: Time - 2.68s, path length: 275
Best First Search Algorithm: Time - 2.46s, path length: 299
A* Search Algorithm: Time - 3.01s, path length: 275

In the Roughly Open Area:
Breadth First Search Algorithm: Time - 2.58s, path length: 93
Best First Search Algorithm: Time - 0.22s, path length: 101
A* Search Algorithm: Time - 0.76s, path length: 93

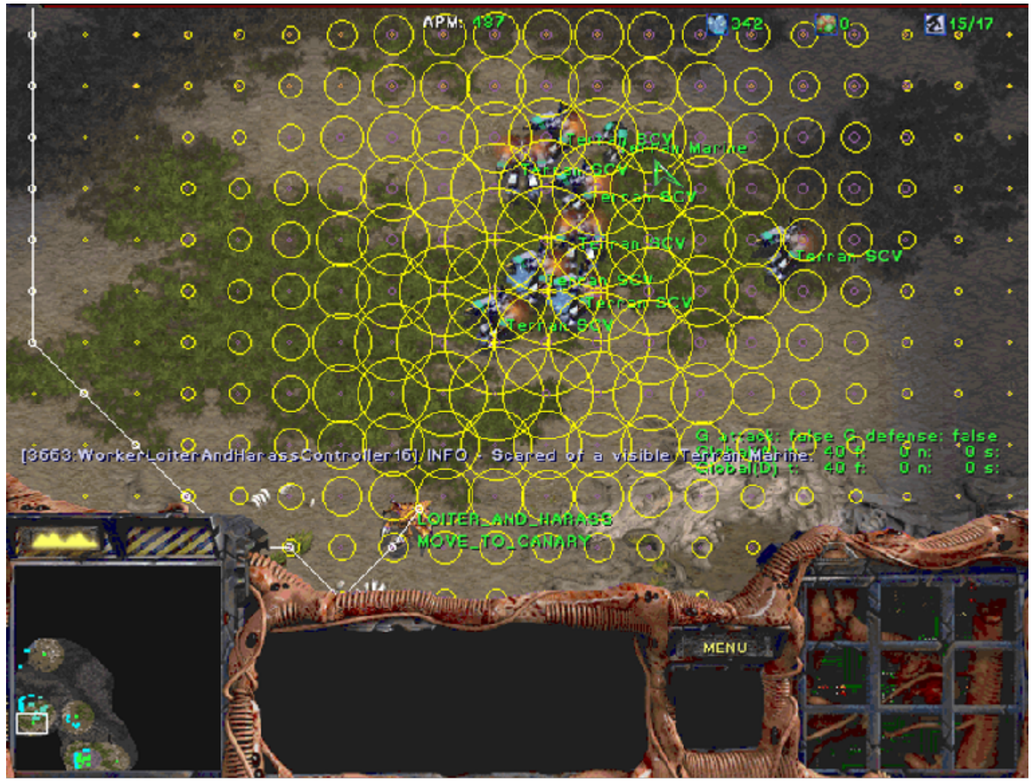
In the Curvy Path Area:
Breadth First Search Algorithm: Time - 2.84s, path length: 123
Best First Search Algorithm: Time - 0.60s, path length: 175
A* Search Algorithm: Time - 2.38, path length: 123



https://www.youtube.com/watch?v=X3x7BILgS-4&ab_channel=TimoBingmann

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- ...



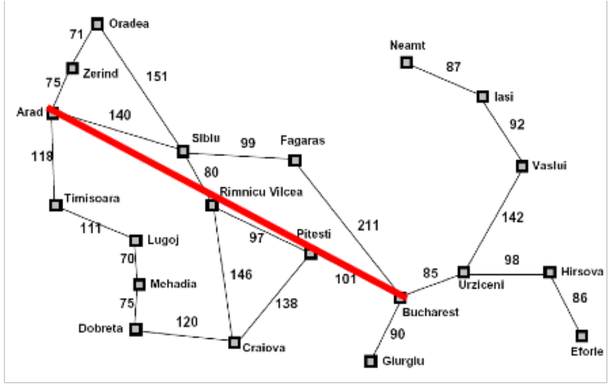
Heuristics

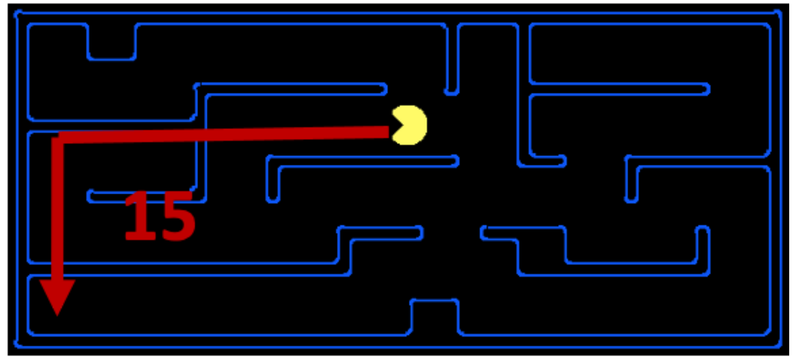
Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

- Often, admissible heuristics are solutions to relaxed problems, where new actions are available

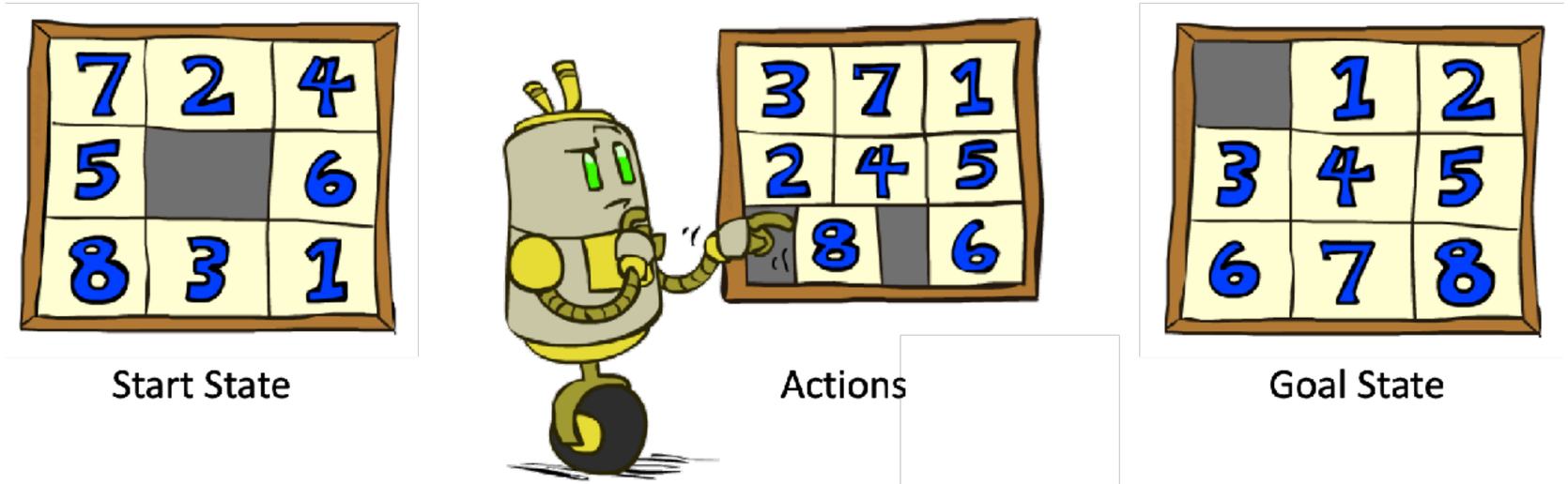
366





- Inadmissible heuristics are often useful too

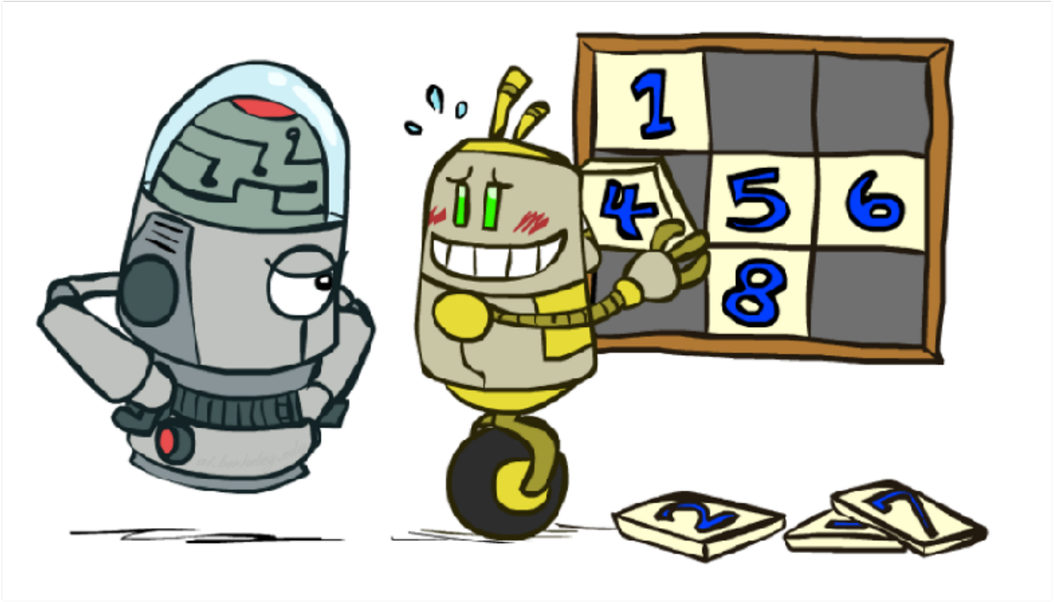
Example: 8 Puzzle

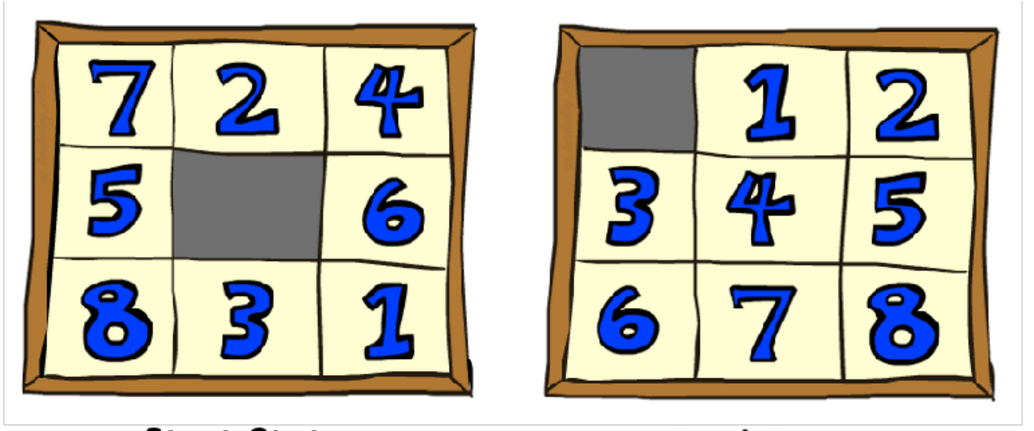


- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

50 Example: 8 Puzzle

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a **relaxed-problem** heuristic



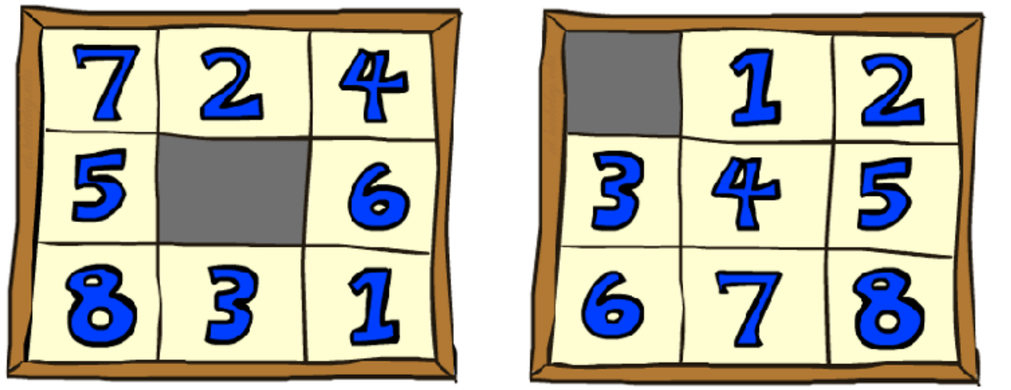


	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

Statistics from Andrew Moore

Example: 8 Puzzle

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total Manhattan distance
- Why is it admissible?
- h(start) = 3+1+2+… = 18



Start State Goal State

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
TILES	13	39	227
MANHATTAN	12	25	73

Example: 8 Puzzle

- How about using the actual cost as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?

- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Trivial Heuristics, Dominance

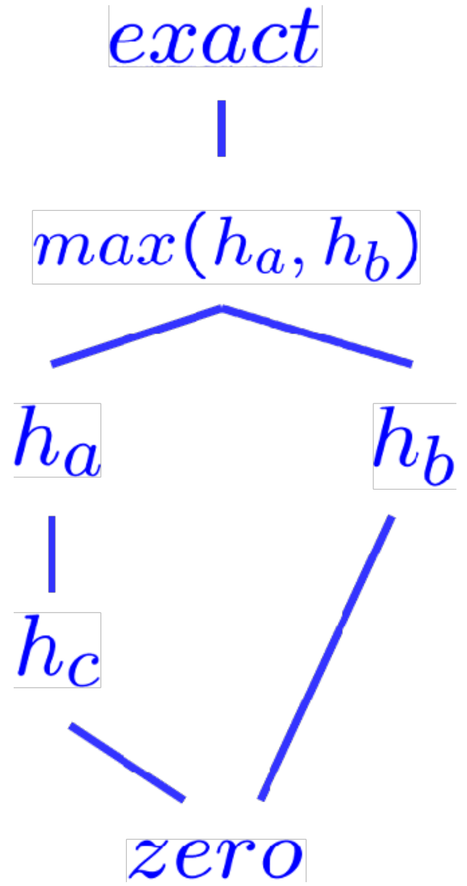
- Dominance: ha ≥ hc if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

$$h(n) = max(h_a(n), h_b(n))$$

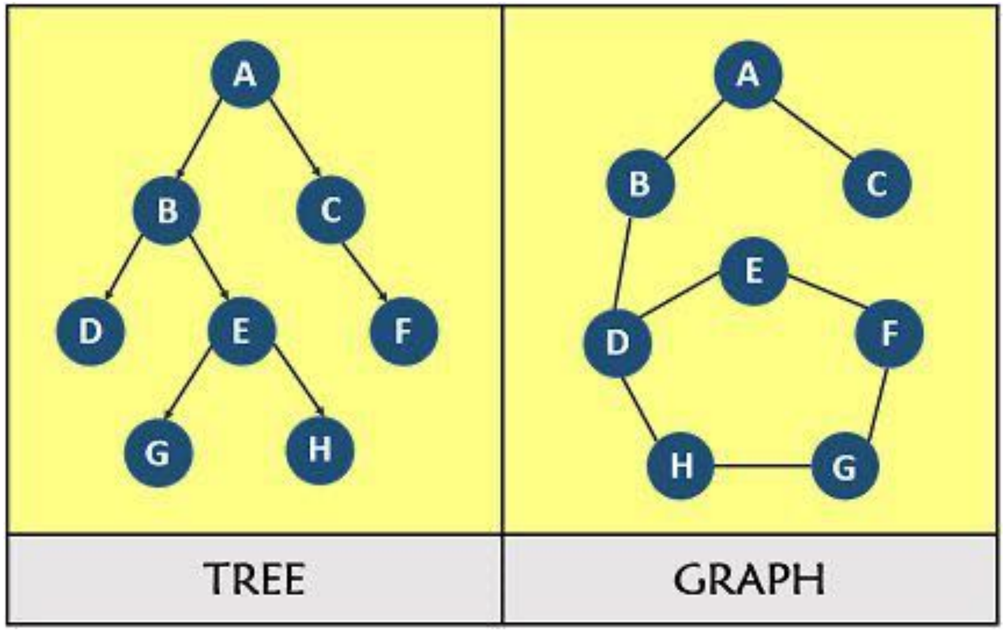
- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



Graph Search

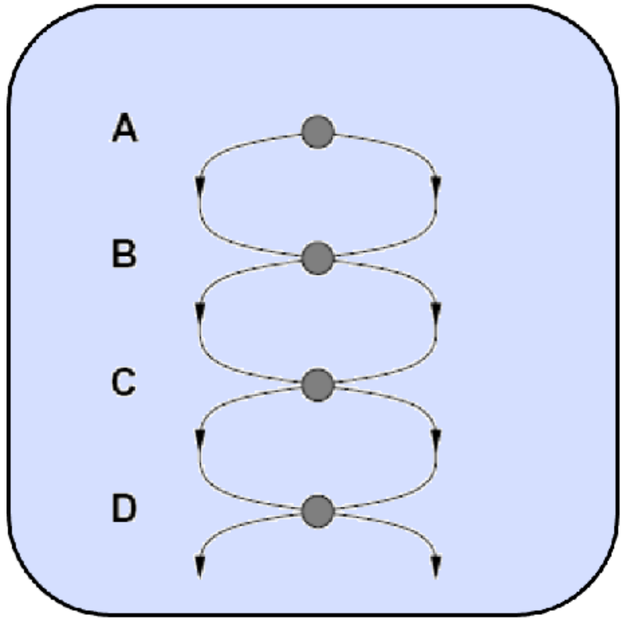
Graph Search vs. Tree Search

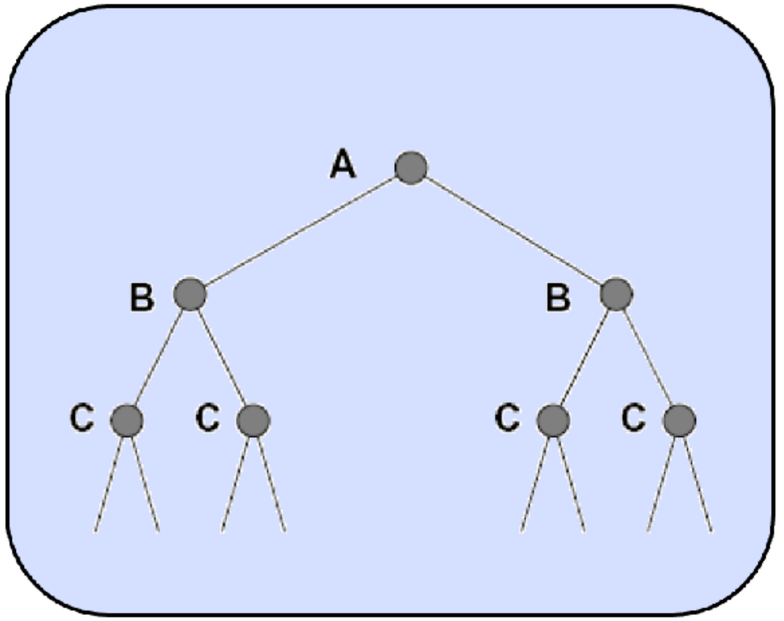
- **The problem is always a graph**
  - The distinction between tree search and graph search is not rooted in the fact whether the problem graph is a tree or a general graph.
  - The distinction lies in the traversal pattern that is used to search through the graph, which can be graph-shaped or tree-shaped.



Tree Search

- **Tree Search**
  - Tree search will visit a state of the underlying problem graph multiple times, if there are multiple directed paths to it rooting in the start state.
  - Failure to detect repeated states can cause exponentially more work.





Graph Search

- **Idea: never expand a state twice**

- **How to implement:**
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but...
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set

- Important: store the closed set as a set, not a list

- Can graph search wreck completeness? Why/why not?

- How about optimality?

Tree Search Pseudo-Code

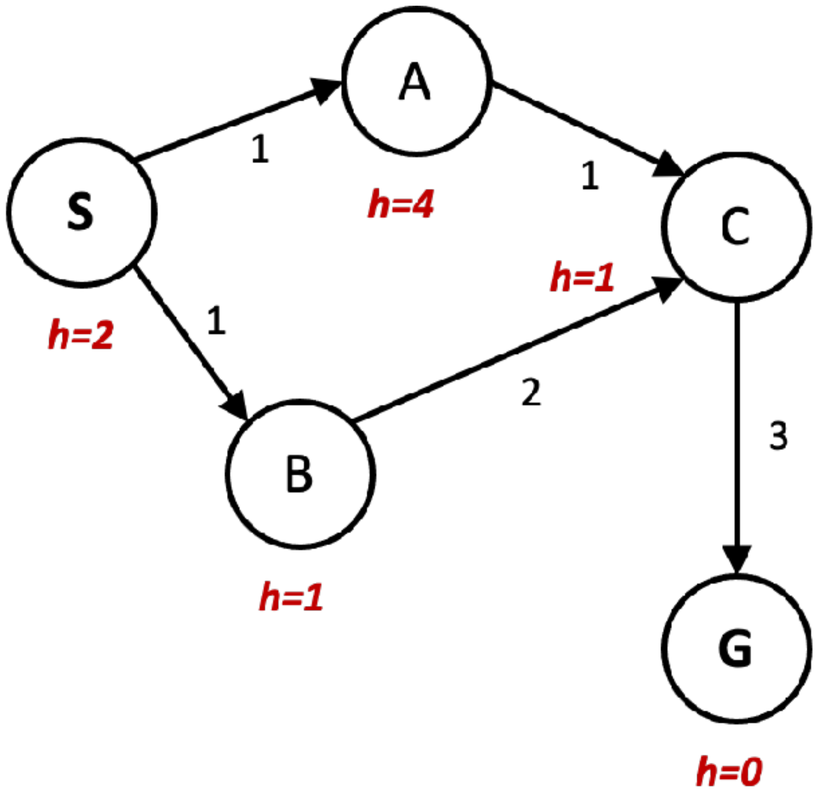
```
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
```

Graph Search Pseudo-Code

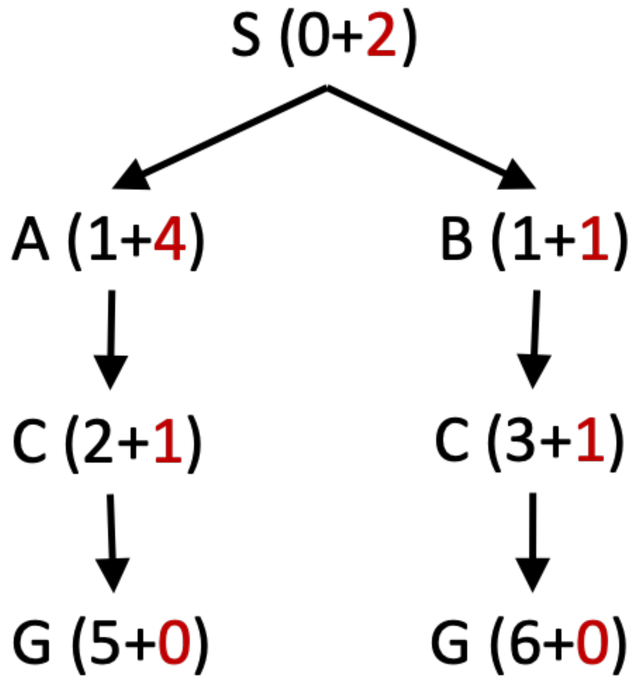
```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

A* Graph Search Gone Wrong?

State space graph



Search tree



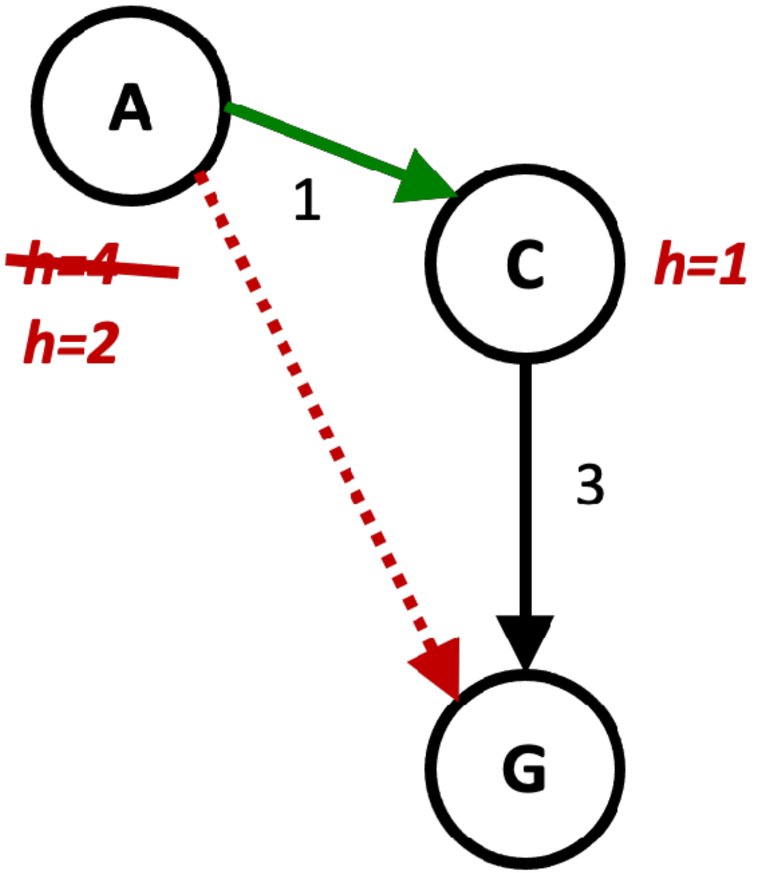
S->B->C->G
Back to ->A-> C
But C is already visited, so cannot do it.

Graph Search

- **Main idea: estimated heuristic costs ≤ actual costs**
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

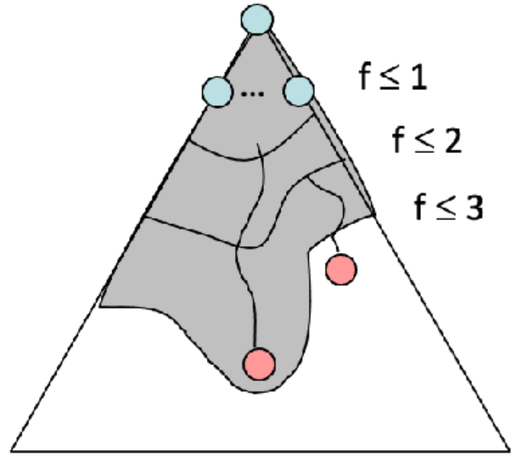
    h(A) – h(C) ≤ cost(A to C)



- **Consequences of consistency:**
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality of A* Graph Search



Consider what A* does with a consistent heuristic:

- **Fact 1:** In tree search, A* expands nodes in increasing total f value (f-contours)

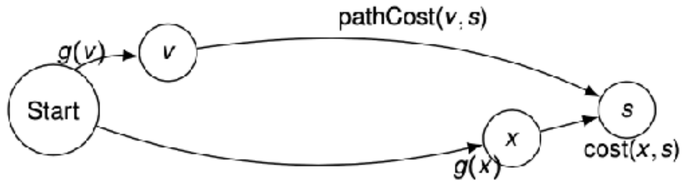
Claim: If y is expanded due to x, $f(y) \geq f(x)$.
Proof:

$$f(y) = g(x) + cost(x,y) + h(y)$$
$$\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)$$

The "estimate" of plan cost keeps rising as you progress.

- **Fact 2:** For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally

- **Result:** A* graph search is optimal



Fact 2: The optimal path is discovered to every state s.

Proof: Consider first error.
State s discovered from x.
Optimal path is from $y \neq x$.
There is a vertex v in the optimal path to y in fringe.
s in fringe with key $f(s) = g(x) + cost(x,s) + h(s)$.
v in fringe with key $f(v) = g(v) + h(v)$.
$h(v) - h(s) \leq pathCost(v,s)$ by induction.
$g(v) + pathCost(v,s) < g(x) + cost(x,s)$
$\Longrightarrow f(v) < f(s)$.
But then v would have been expanded before s!

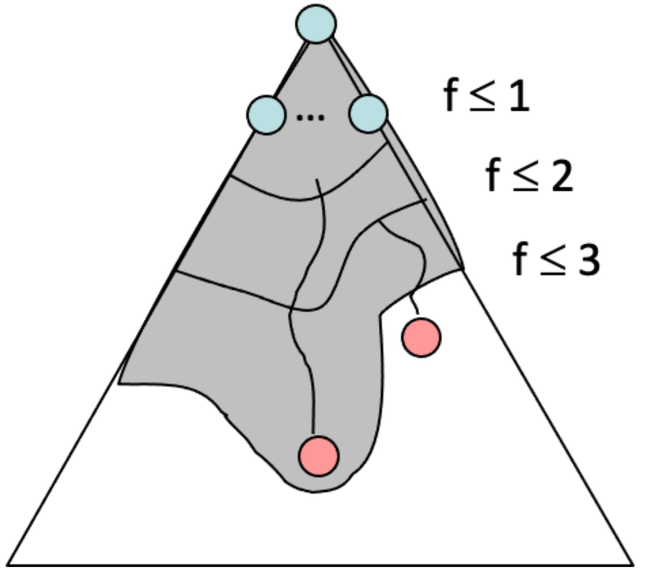
Optimality of A* Graph Search

- **Tree search:**
  - A* is optimal if heuristic is *admissible*
  - UCS is a special case (h = 0)

- **Graph search:**
  - A* optimal if heuristic is *consistent*
  - UCS optimal (h = 0 is consistent)

- **Consistency implies admissibility**

- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

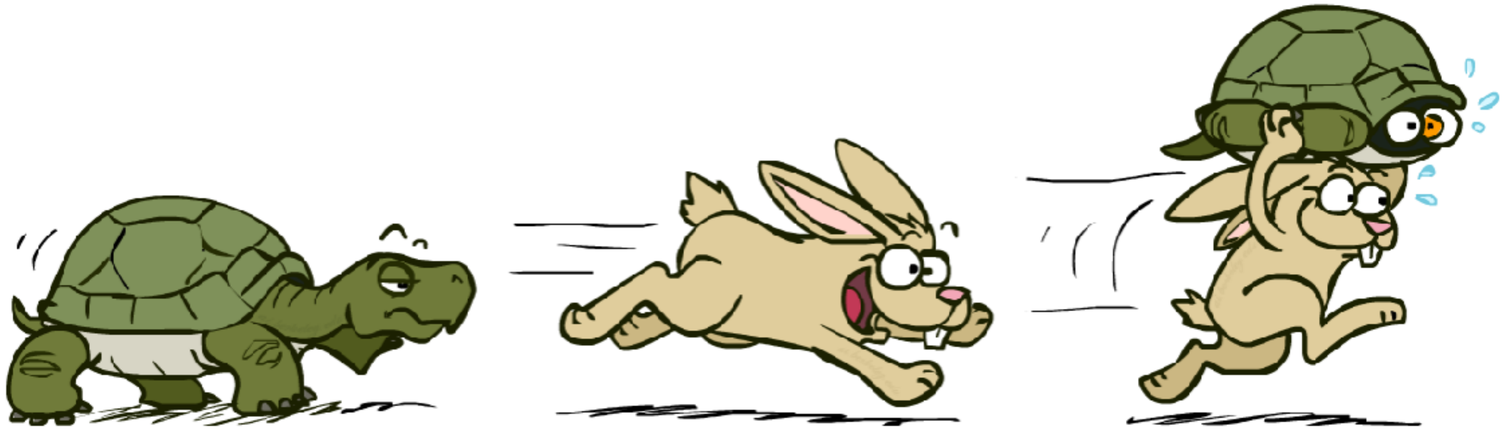


A* Variants

- Iterative Deepening A*
- Recursive best first search (incorporates A* idea, despite name)
- Memory Bounded A*
- Simplified Memory Bounded A*
- *See AIMA 3.5.4, 3.5.5 if you are interested in these topics*

A* Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Thanks! Q&A


Université
de Montréal


rali
recherche appliquée en
linguistique informatique


Mila