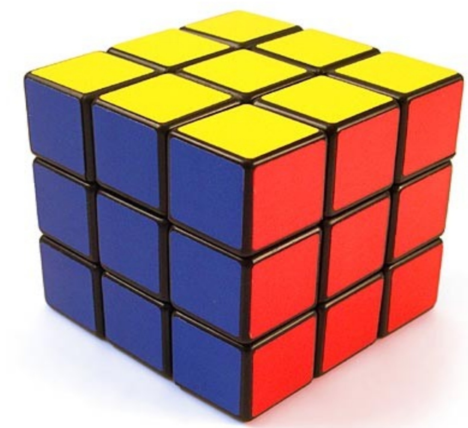
Heuristic Search

Robert Platt
Northeastern University

Some images and slides are used from:
1. CS188 UC Berkeley
2. RN, AIMA

Recap: What is graph search?

Start state



Goal state

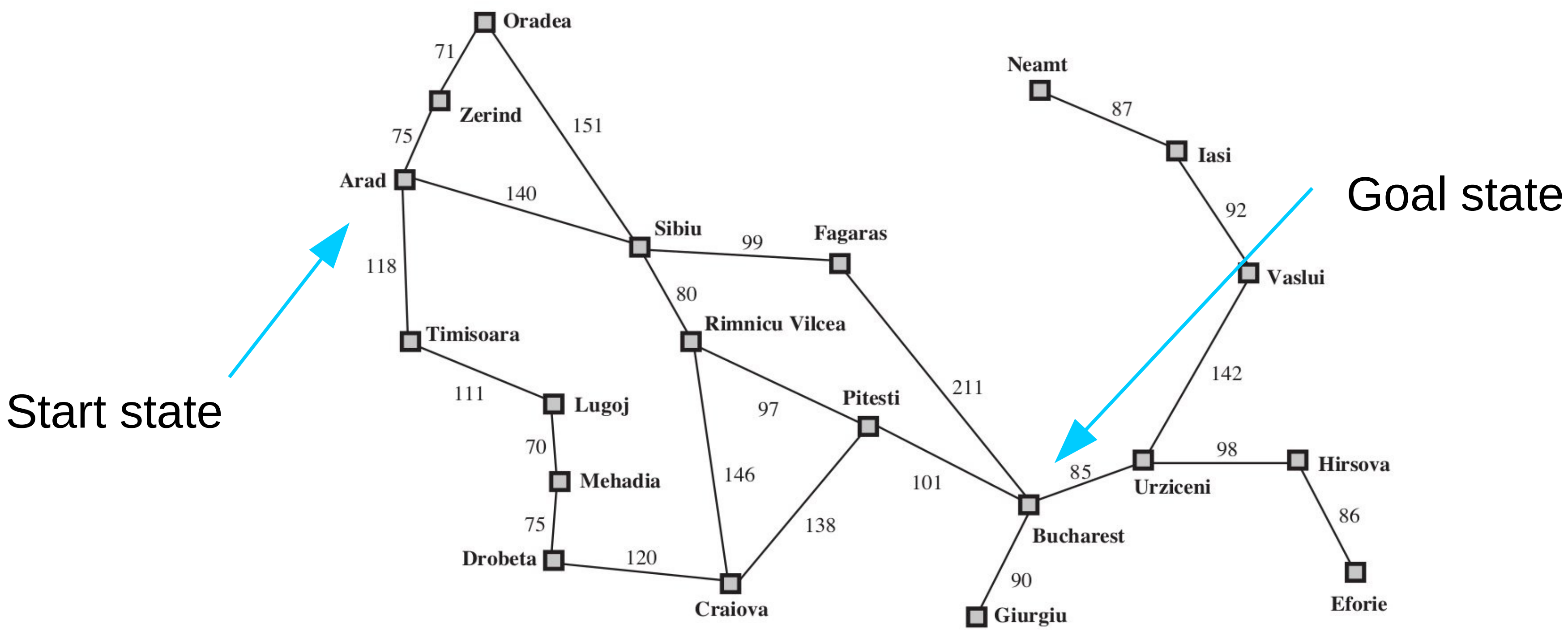
Graph search: find a path from start to goal

– what are the states?

– what are the actions (transitions)?

– how is this a graph?

Recap: What is graph search?



Graph search: find a path from start to goal

– what are the states?

– what are the actions (transitions)?

– how is this a graph?

Recap: BFS/UCS

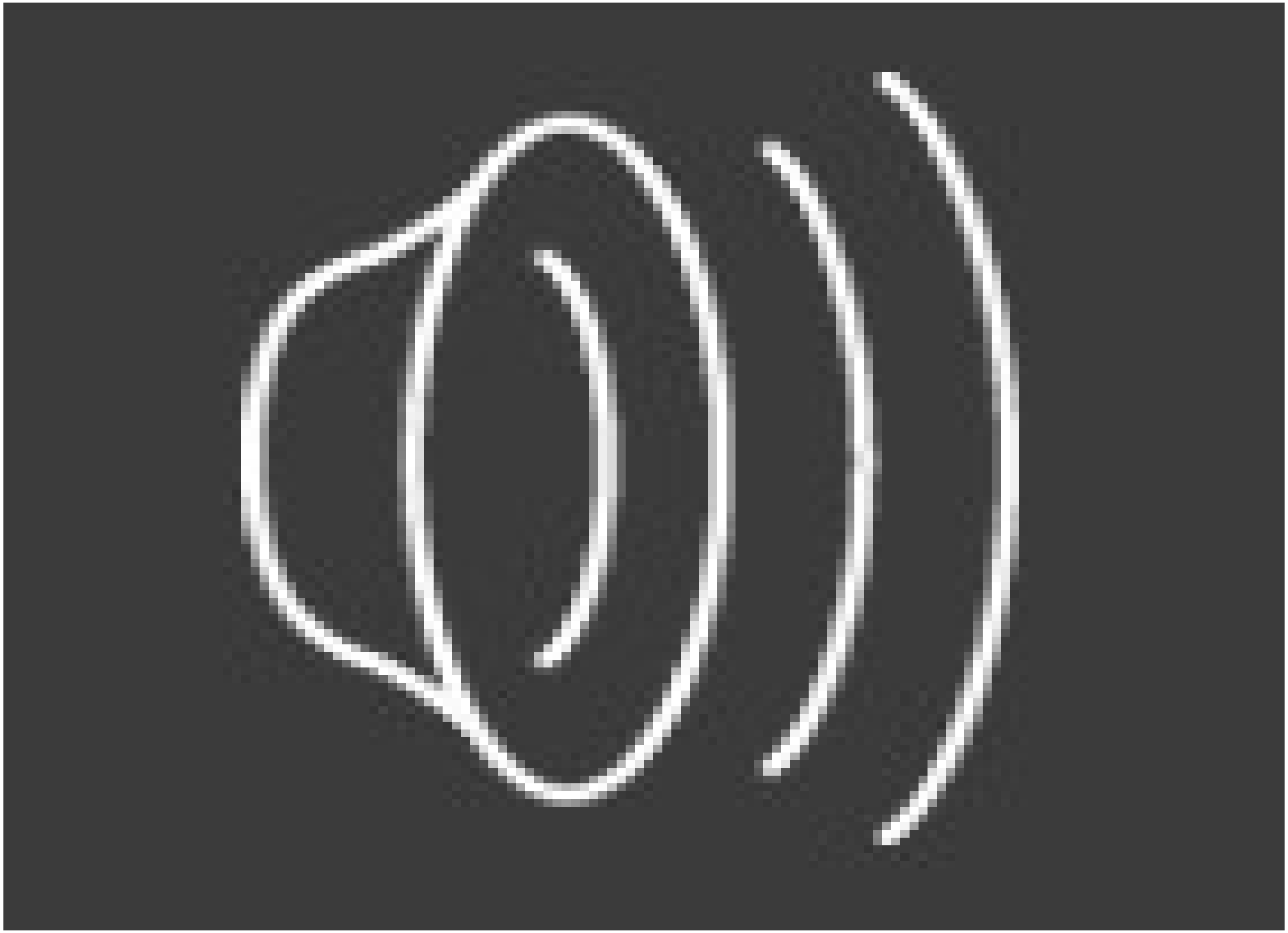


Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Recap: BFS/UCS

Notice that we search equally far in all directions...

It's like this

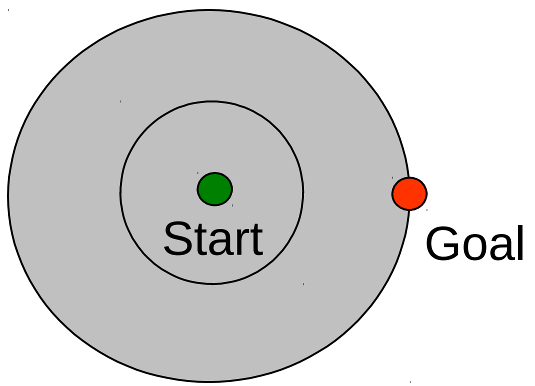


Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Idea

Is it possible to use additional information to decide which direction to search in?

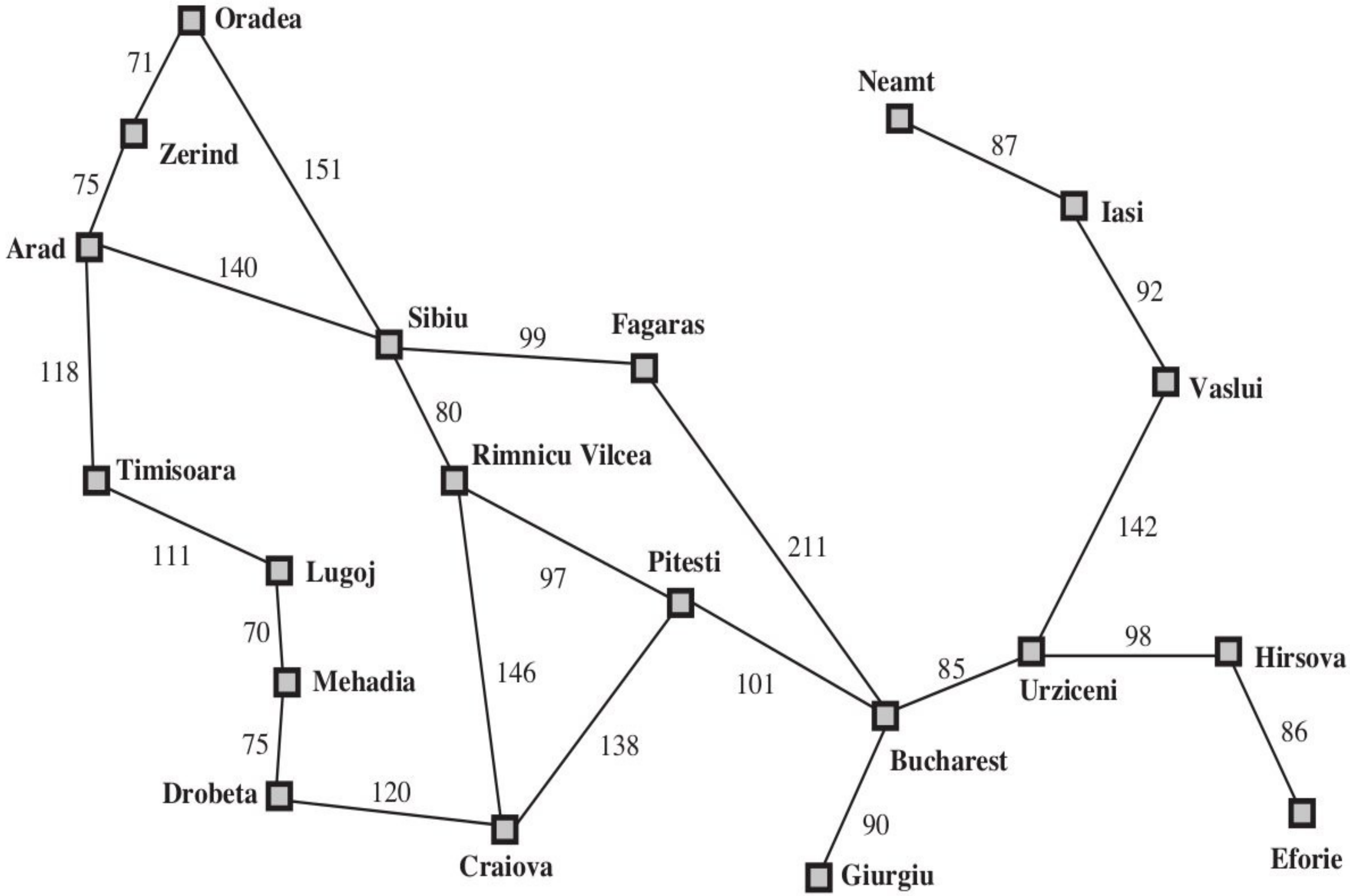
Idea

Is it possible to use additional information to decide which direction to search in?

Yes!

Instead of searching in all directions, let's bias search in the direction of the goal.

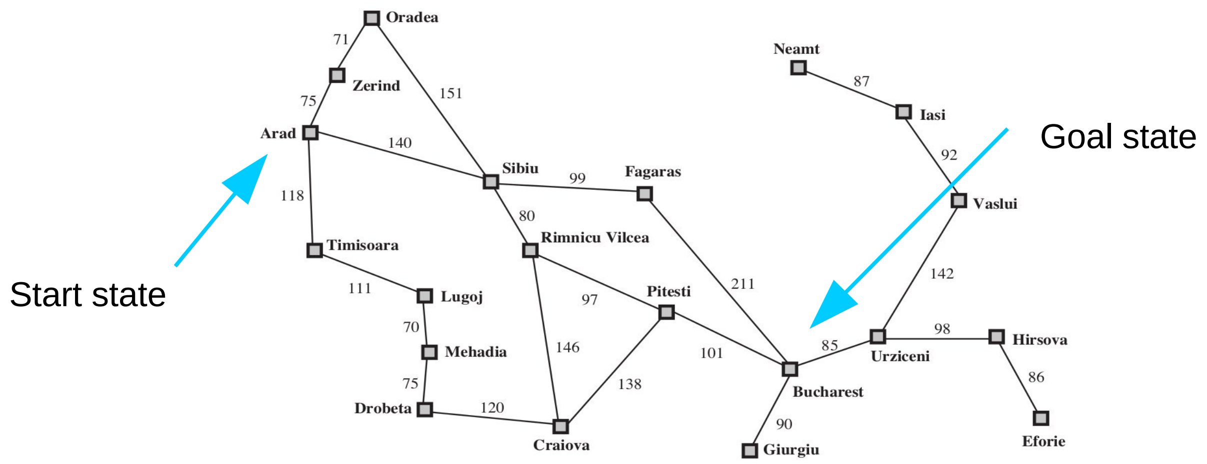
Example



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Stright-line distances to Bucharest

Example

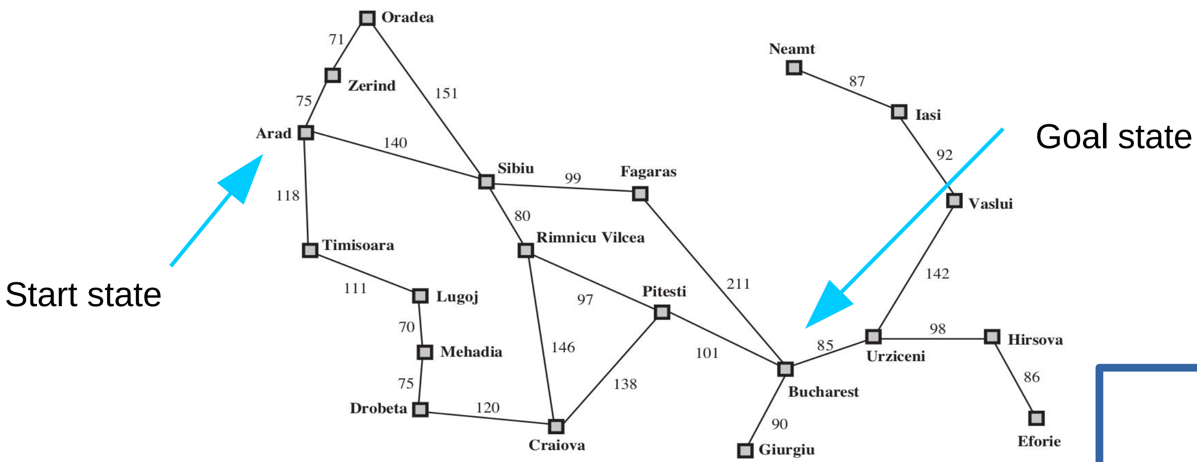


Expand states in order of their distance to the goal

– for each state that you put on the fringe: calculate straight-line distance to the goal

– expand the state on the fringe closest to the goal

Example



Heuristic: $h(s)$

Expand states in order of their distance to the goal

– for each state that you put on the fringe: calculate straight-line distance to the goal

– expand the state on the fringe closest to the goal

Greedy search

Greedy Search

Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Greedy Search

Each time you expand a state, calculate the heuristic for each of the states that you add to the fringe.

– heuristic: $h(s)$ ⬅ i.e. distance to Bucharest

– on each step, choose to expand the state with the lowest heuristic value.

Greedy Search

This is like a guess about how far the state is from the goal

Each time you expand a state, calculate the heuristic for each of the states that you add to the fringe.

– heuristic: $h(s)$ ← i.e. distance to Bucharest

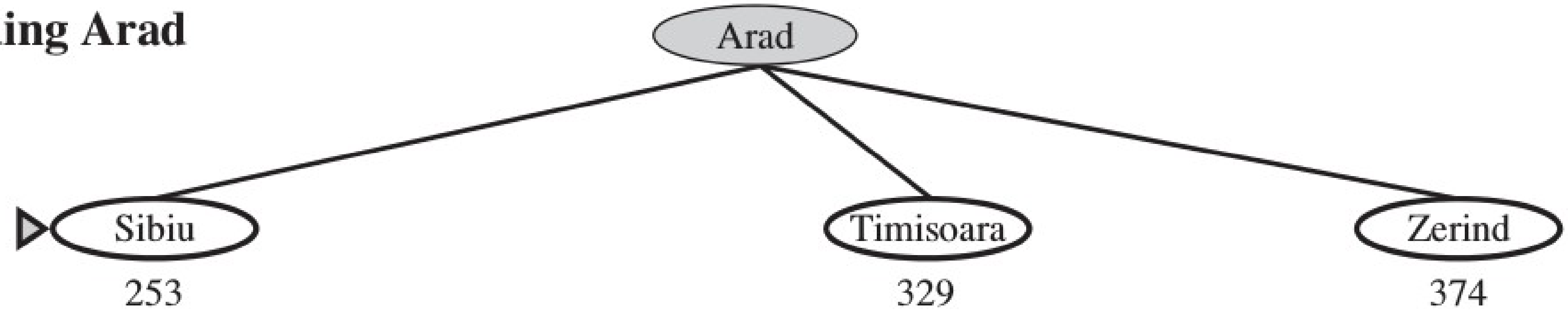
– on each step, choose to expand the state with the lowest heuristic value.

Example: Greedy Search

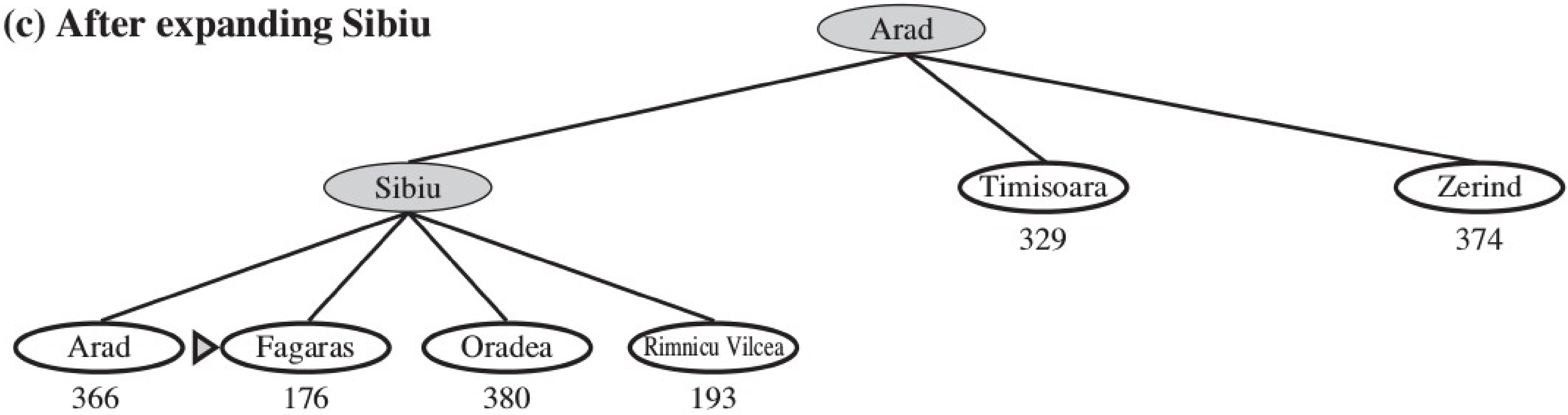
(a) The initial state

Example: Greedy Search

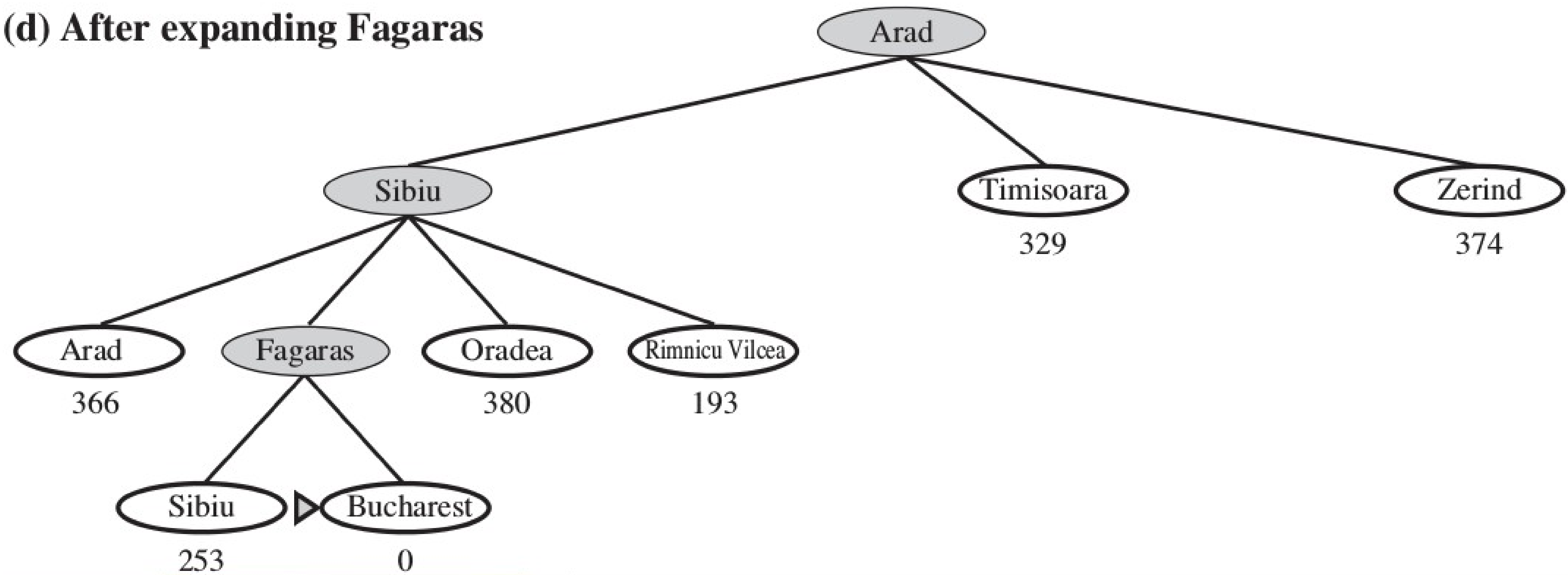
(b) After expanding Arad



Example: Greedy Search

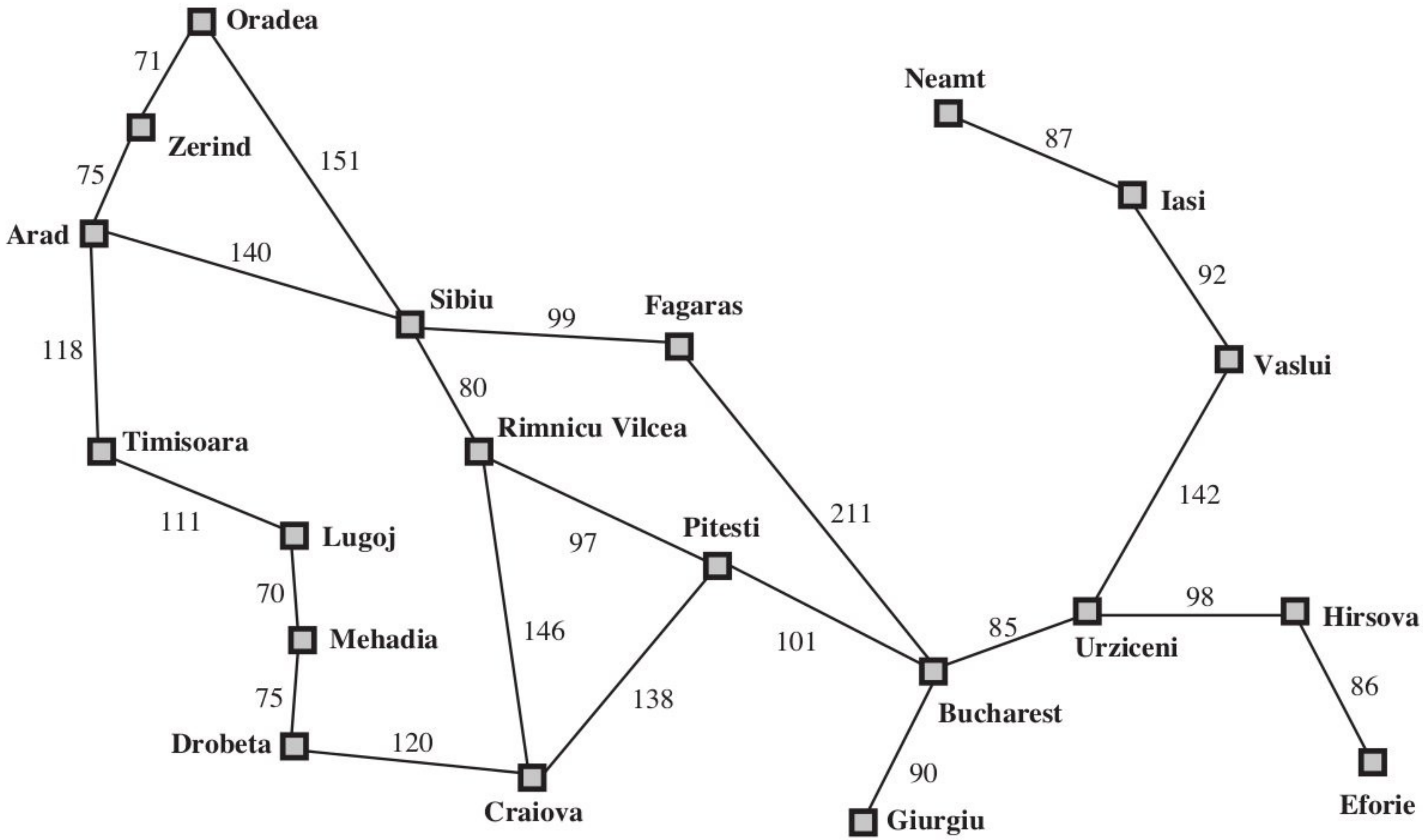


Example: Greedy Search



Path: A-S-F-B

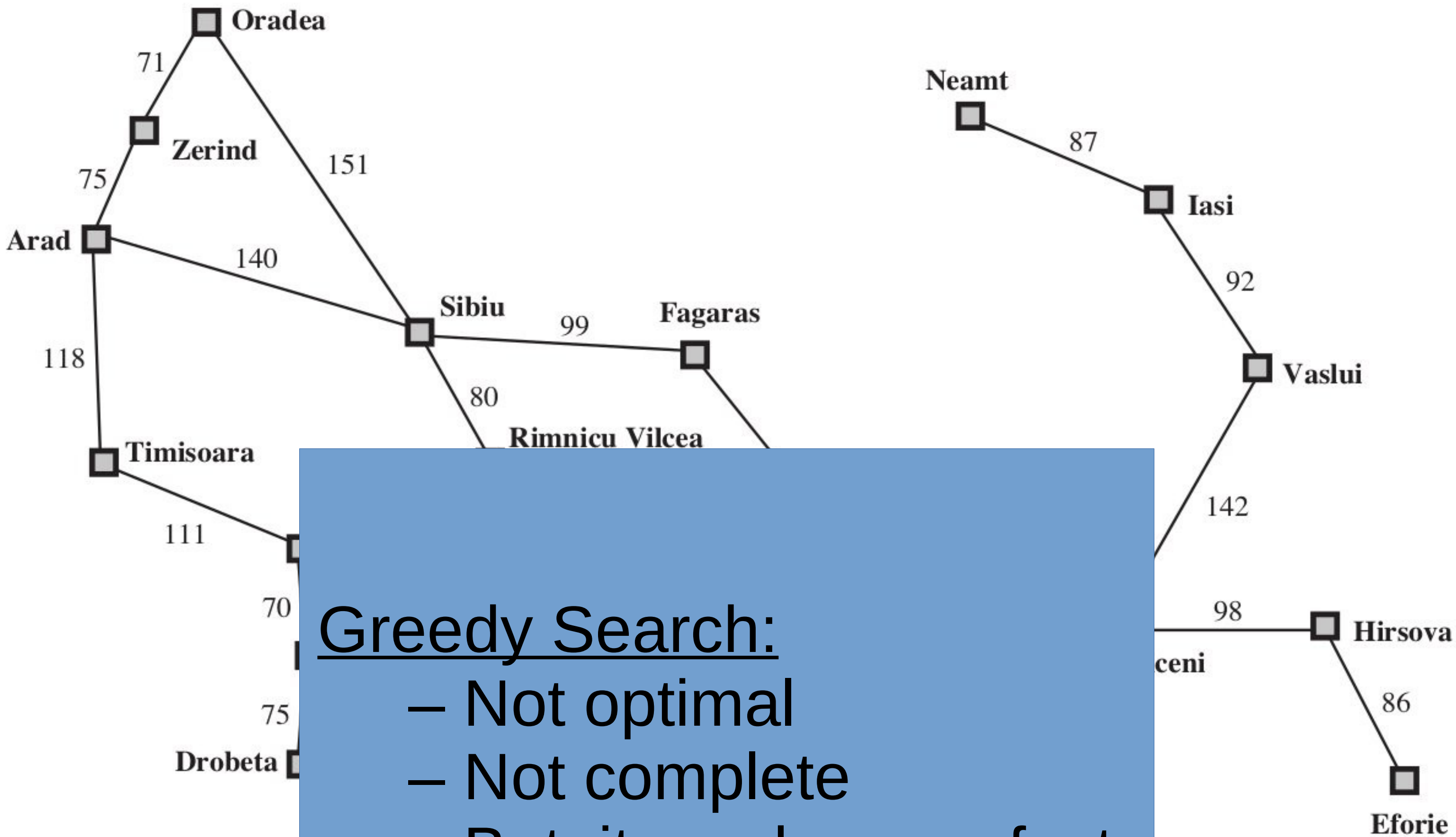
Example: Greedy Search



Path: A-S-F-B

<u>Notice that this is not the optimal path!</u>

Example: Greedy Search



<u>Greedy Search:</u>

– Not optimal
– Not complete
– But, it can be very fast

<u>Notice that this is not the optimal path!</u>

Greedy vs UCS

Greedy Search:

- Not optimal
- Not complete
- But, it can be very fast

UCS:

- Optimal
- Complete
- Usually very slow

Greedy vs UCS

Greedy Search:

- Not optimal
- Not complete
- But, it can be very fast

UCS:

- Optimal
- Complete
- Usually very slow

Can we combine greedy and UCS???

Greedy vs UCS

<u>Greedy Search:</u>

- Not optimal
- Not complete
- But, it can be very fast

<u>UCS:</u>

- Optimal
- Complete
- Usually very slow

Can we combine greedy and UCS???

<u>YES: A*</u>

Greedy vs UCS

UCS

Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Greedy vs UCS

UCS

Greedy

Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Greedy vs UCS

UCS

Greedy

A*

Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

A*

Image: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

A*

s : a state

$g(s)$: minimum cost from start to

$h(s)$: heuristic at (*i.e.* an estimate of remaining cost-to-go)

UCS: expand states in order of $g(s)$

Greedy: expand states in order of $h(s)$

A*: expand states in order of $f(s) = g(s) + h(s)$

A*

<u>What is “cost-to-go”?</u>

s : a state

$g(s)$: minimum cost from start to s

$h(s)$: heuristic at s (*i.e.* an estimate of remaining cost-to-go)

<u>UCS</u>: expand states in order of $g(s)$

<u>Greedy</u>: expand states in order of $h(s)$

<u>A*</u>: expand states in order of $f(s) = g(s) + h(s)$

A*

s : a state

What is "cost-to-go"?
– minimum cost required to reach a goal state

$g(s)$: minimum cost from start to s

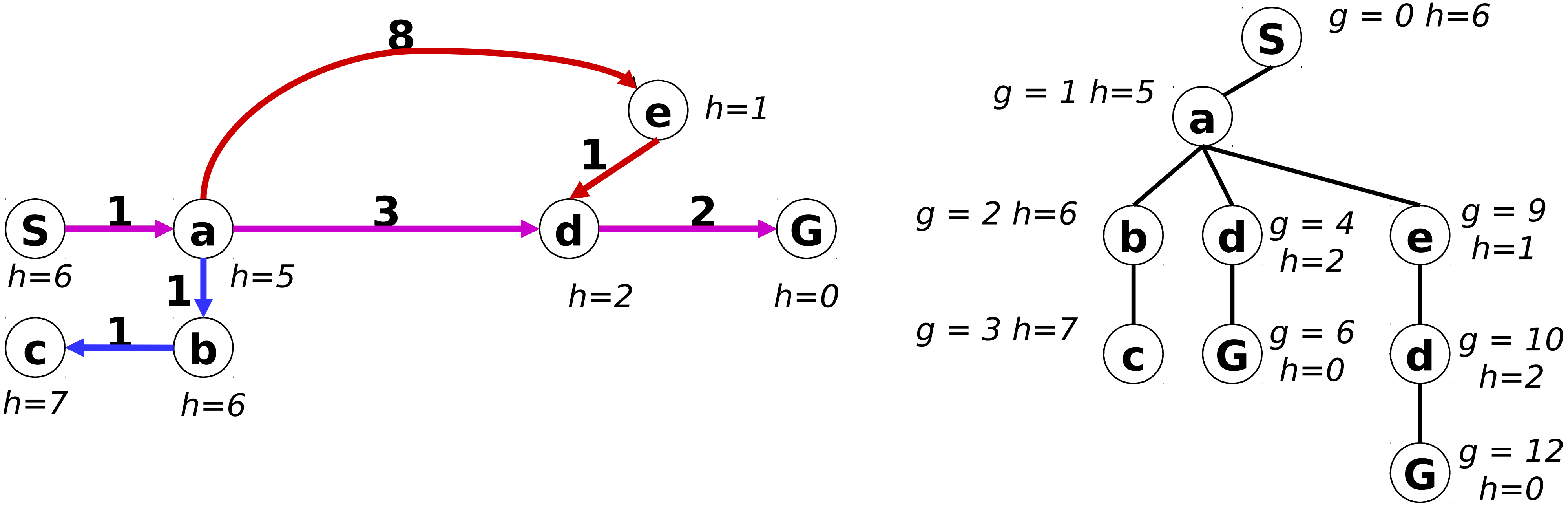
$h(s)$: heuristic at s (*i.e.* an estimate of remaining cost-to-go)

UCS: expand states in order of $g(s)$

Greedy: expand states in order of $h(s)$

A*: expand states in order of $f(s) = g(s) + h(s)$

A*

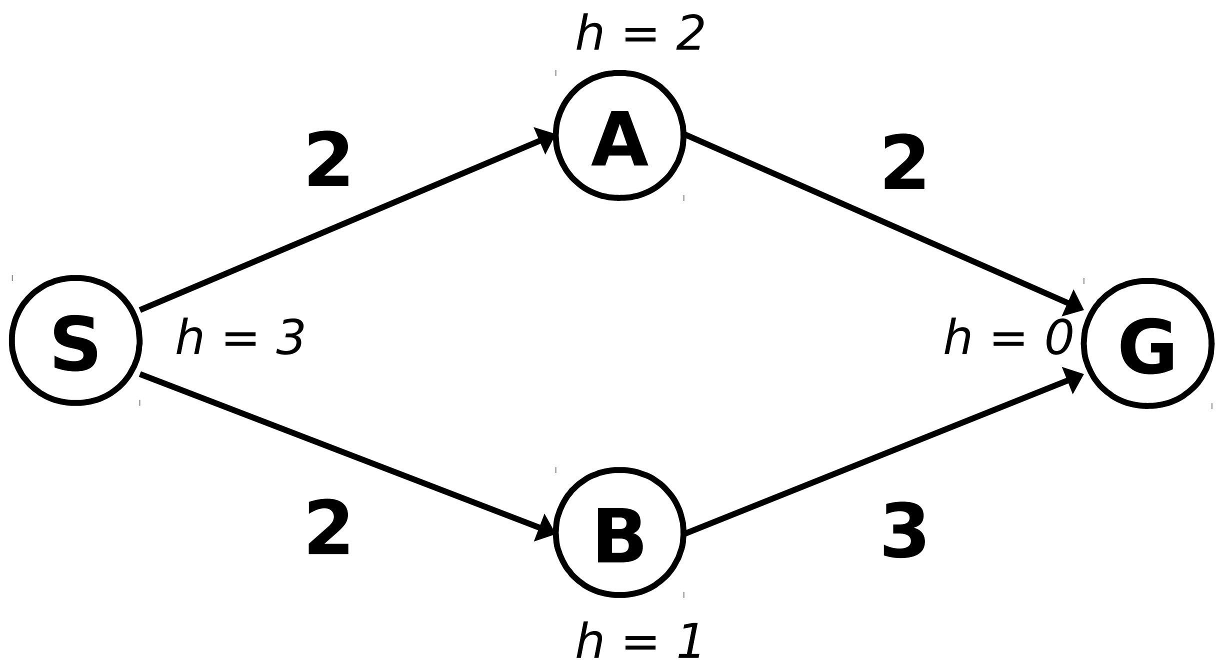


<u>A*</u>: expand states in order of $f(s) = g(s) + h(s)$

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

When should A* terminate?

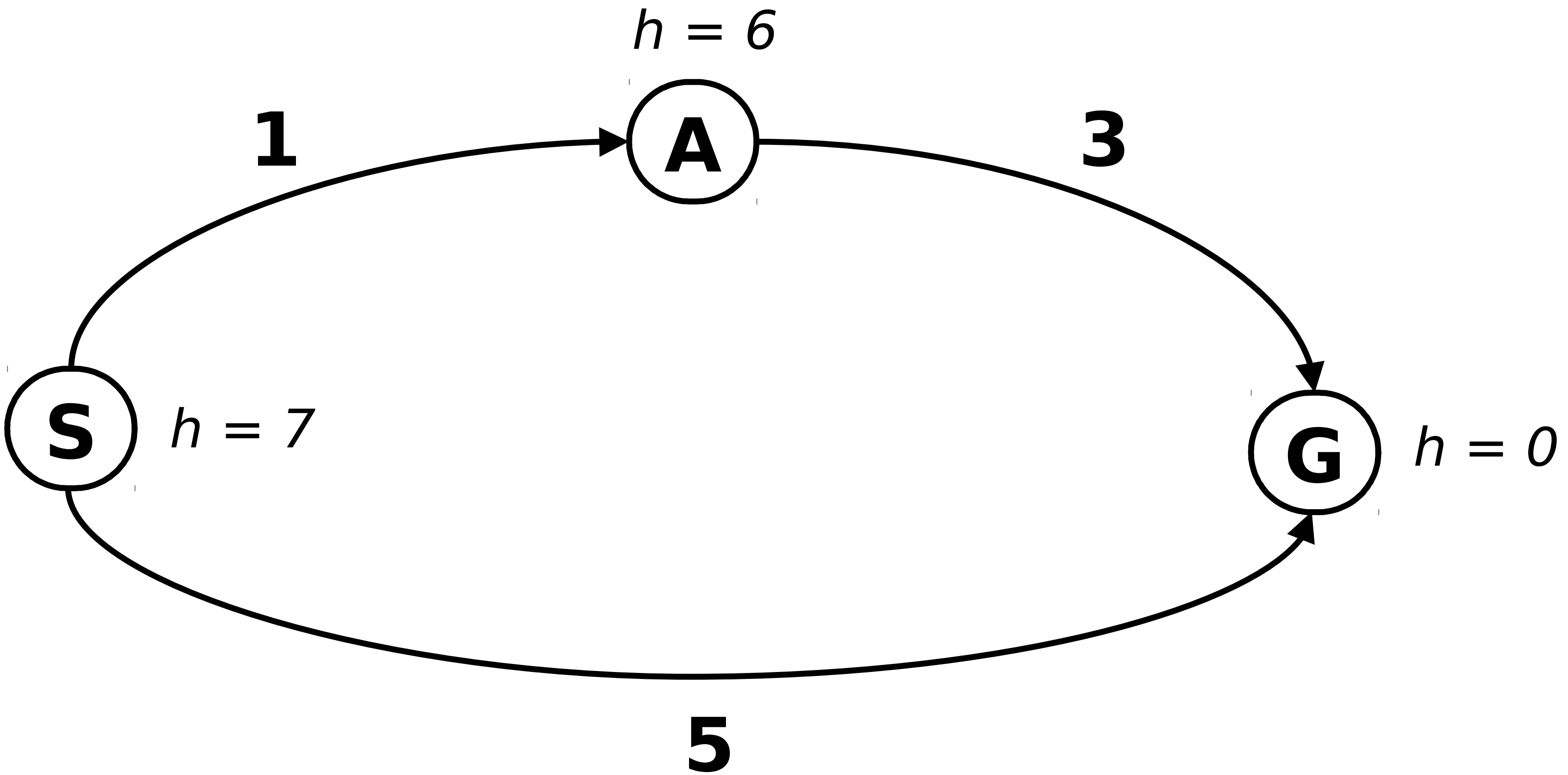
Should we stop when we enqueue a goal?



No: only stop when we dequeue a goal

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Is A* optimal?



What went wrong?
Actual cost-to-go < heuristic
The heuristic must be less than the actual cost-to-go!

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

When is A* optimal?

It depends on whether we are using the tree search or the graph search version of the algorithm.

Recall:

– in tree search, we do not track the explored set
– in graph search, we do

Recall: Breadth first search (BFS)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

What is the purpose of the *explored* set?

When is A* optimal?

It depends on whether we are using the <u>tree search</u> or the <u>graph search</u> version of the algorithm.

Optimal if h is <u>consistent</u>

Optimal if h is <u>admissible</u>

When is A* optimal?

It depends on whether we are using the tree search or the graph search version of the algorithm.

Optimal if h is consistent

– h(s) is an underestimate of the cost of each arc.

Optimal if h is admissible

– *h(s)* is an underestimate of the true cost-to-go.

When is A* optimal?

It depends on whether we are using the <u>tree search</u> or the <u>graph search</u> version of the algorithm.

Optimal if h is <u>consistent</u>

– h(s) is an underestimate of the cost of each arc.

Optimal if h is <u>admissible</u>

– *h(s)* is an underestimate of the true cost-to-go.

<u>What is "cost-to-go"?</u>

– minimum cost required to reach a goal state

When is A* optimal?

It depends on whether we are using the tree search or the graph search version of the algorithm.

Optimal if h is consistent

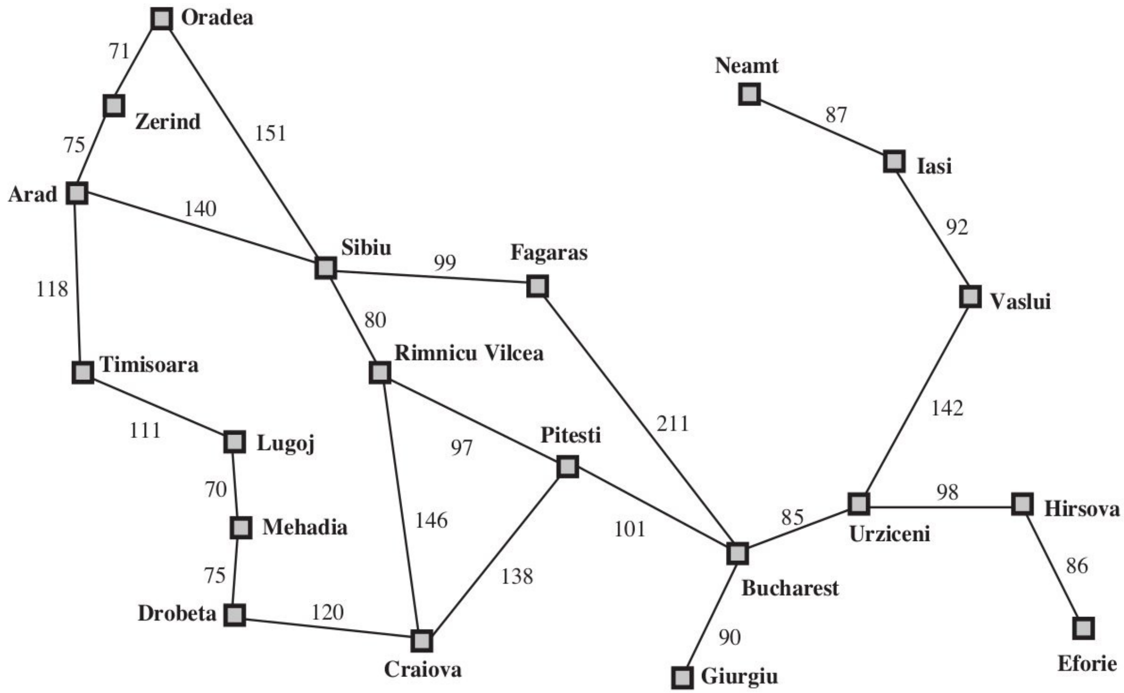
– h(s) is an underestimate of the cost of each arc.

Optimal if h is admissible

– *h(s)* is an underestimate of the true cost-to-go.

More on this later...

Admissibility: Example

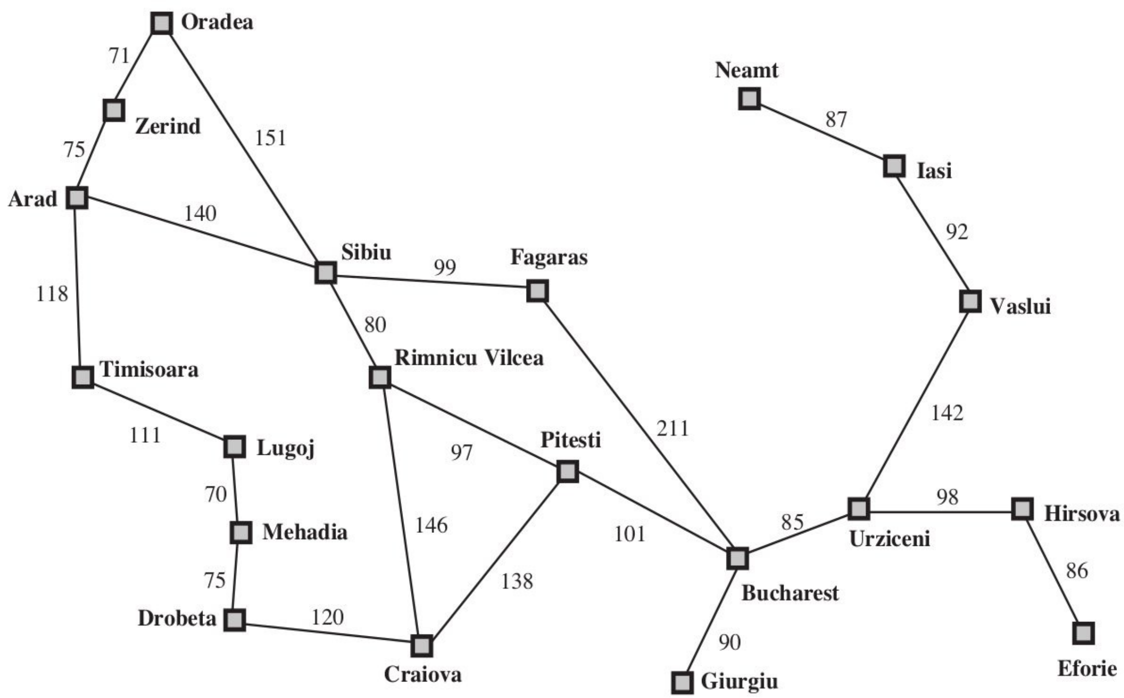


Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Stright-line distances to Bucharest

h(s) = straight-line distance to goal state (Bucharest)

Admissibility



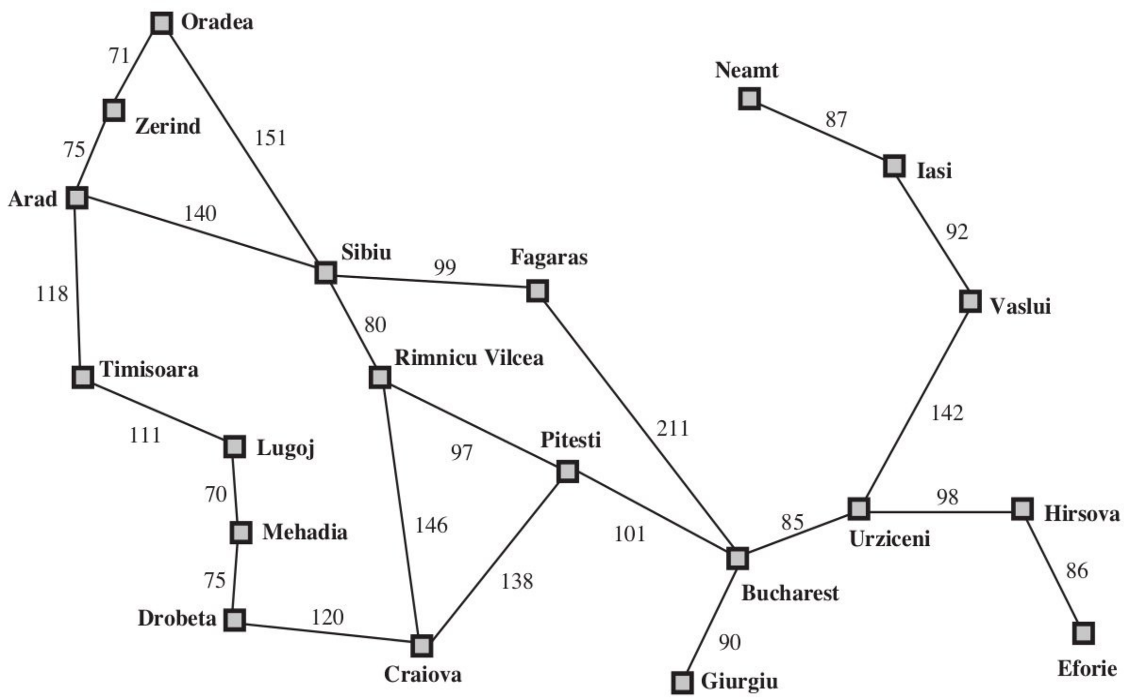
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Stright-line distances to Bucharest

h(s) = straight-line distance to goal state (Bucharest)

Is this heuristic admissible???

Admissibility



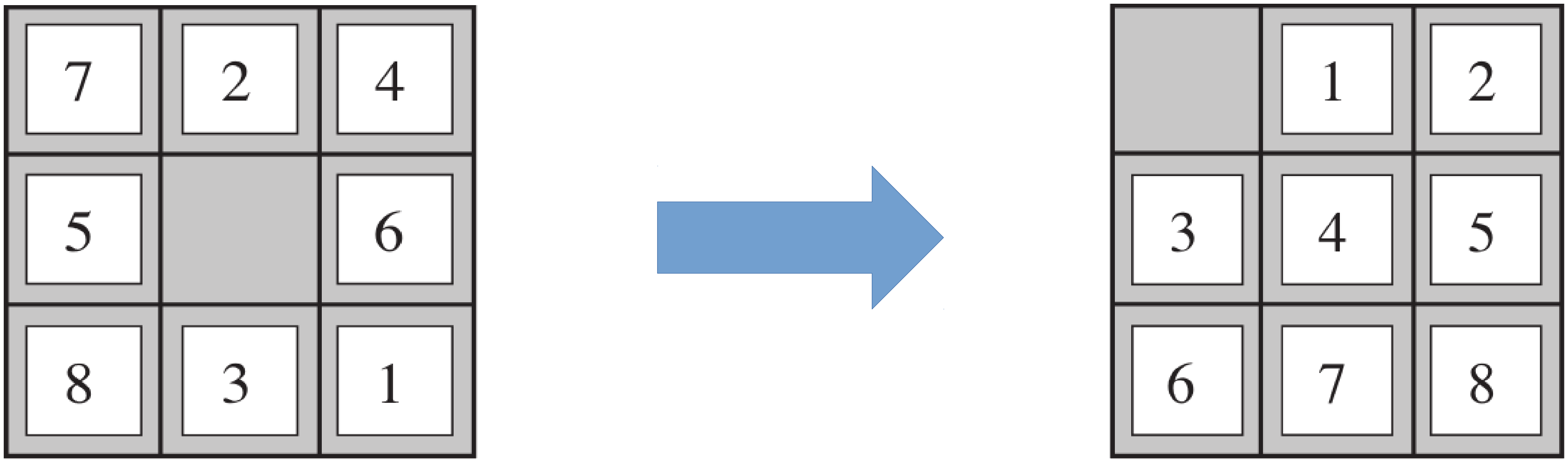
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Stright-line distances to Bucharest

h(s) = straight-line distance to goal state (Bucharest)

Is this heuristic admissible???
YES! Why?

Admissibility: Example



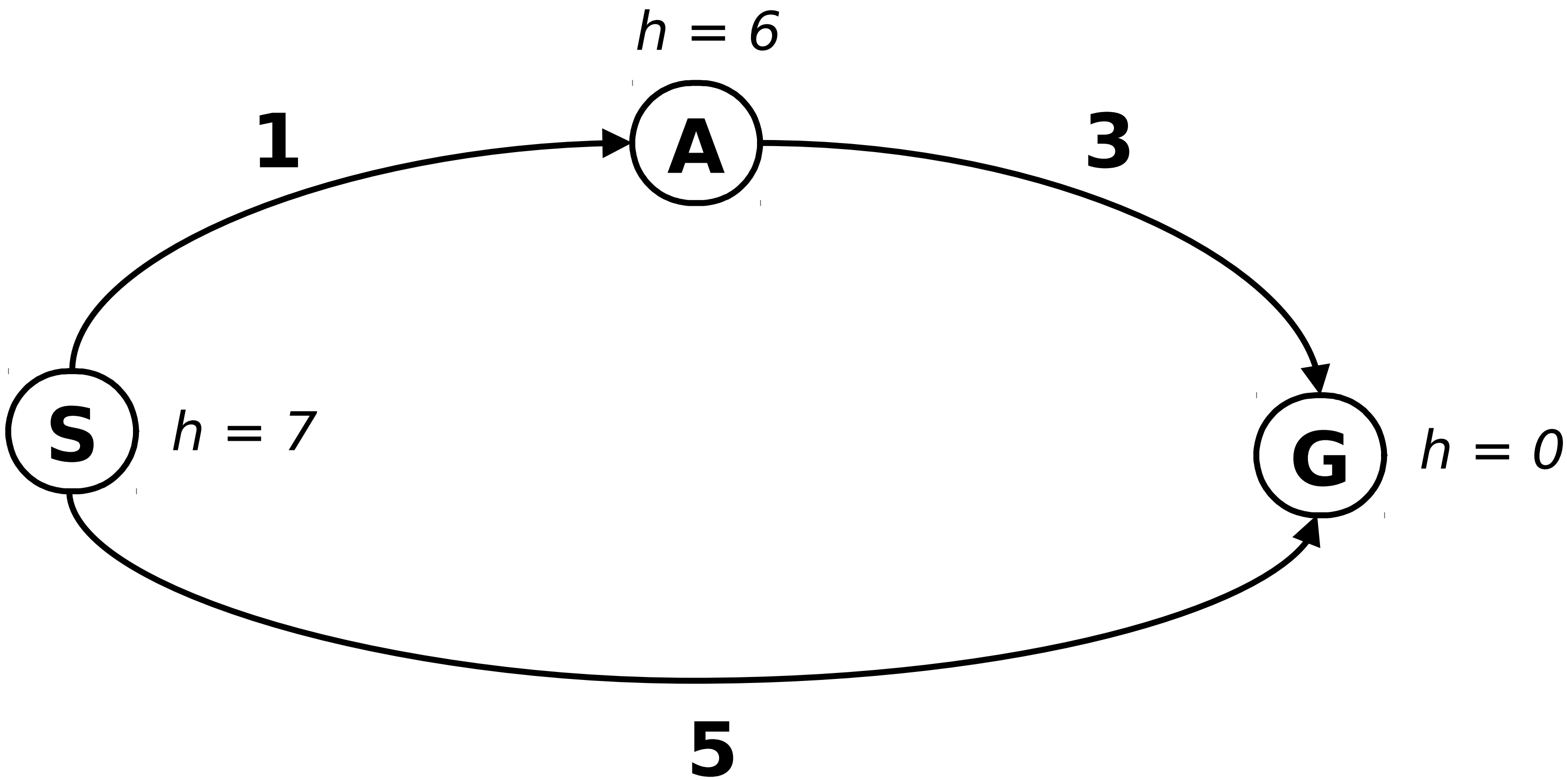
Start state

Goal state

h(s) = ?

Can you think of an admissible heuristic for this problem?

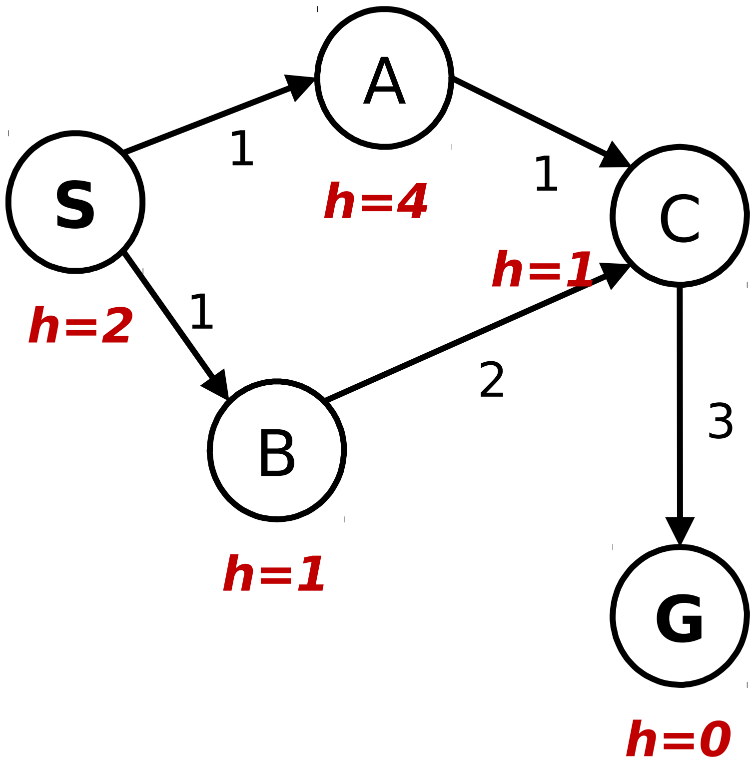
Admissibility



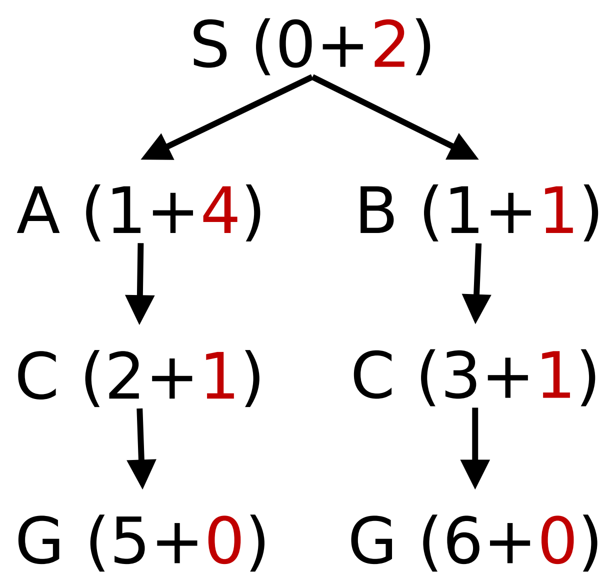
Why isn't this heuristic admissible?

Consistency

State space graph



Search tree



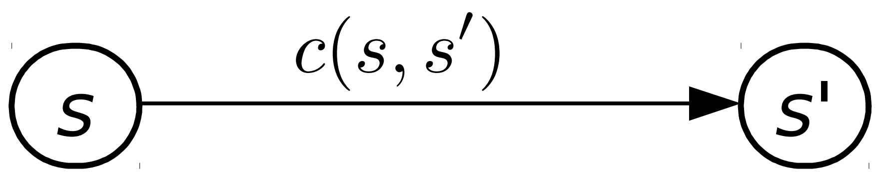
What went wrong?

Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Consistency

$$h(s) \leq c(s, s') + h(s')$$

Cost of going from *s* to *s′*



Consistency

$$h(s) \leq c(s, s') + h(s')$$

$$h(s) - h(s') \leq c(s, s')$$ ⟵ Rearrange terms

Consistency

$$h(s) \leq c(s, s') + h(s')$$

$$\underbrace{h(s) - h(s')}_{} \leq c(s, s')$$

Cost of going from s to s'
implied by heuristic

Actual cost of
going from *s* to *s'*

Consistency

$$h(s) \leq c(s, s') + h(s')$$

$$h(s) - h(s') \leq c(s, s')$$

Cost of going from s to s'
implied by heuristic

Actual cost of
going from *s* to *s'*

Consistency

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

Consistency implies that the “f-cost” never decreases along any path to a goal state.
– the optimal path gives a goal state its lowest f-cost.

A* expands states in order of their f-cost.

Given any goal state, A* expands states that reach the goal state optimally before expanding states the reach the goal state suboptimally.

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T) + h(s_T)$

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T)$

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T)$ ← admissible

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T)$

$h(s_{T-2}) \leq c(s_{T-2}, s_{T-1}) + h(s_{T-1})$

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T)$

$$h(s_{T-2}) \leq c(s_{T-2}, s_{T-1}) + h(S_{T-1})$$

admissible

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T)$

$h(s_{T-2}) \leq c(s_{T-2}, s_{T-1}) + h(S_{T-1})$

admissible admissible

Consistency implies admissibility

Suppose: $\forall s_t, s_{t+1} : h(s_t) \leq c(s_t, s_{t+1}) + h(s_{t+1})$

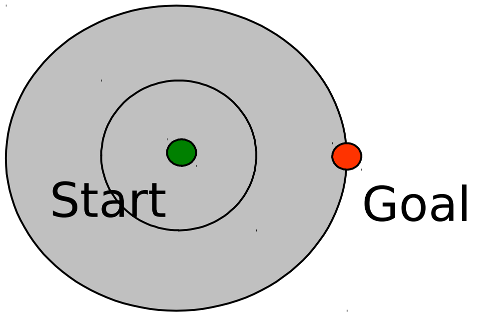
Then: $h(s_{T-1}) \leq c(s_{T-1}, s_T)$

$$h(s_{T-2}) \leq c(s_{T-2}, s_{T-1}) + h(S_{T-1})$$

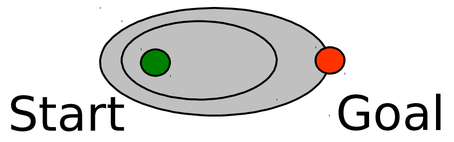
• • •

A* vs UCS

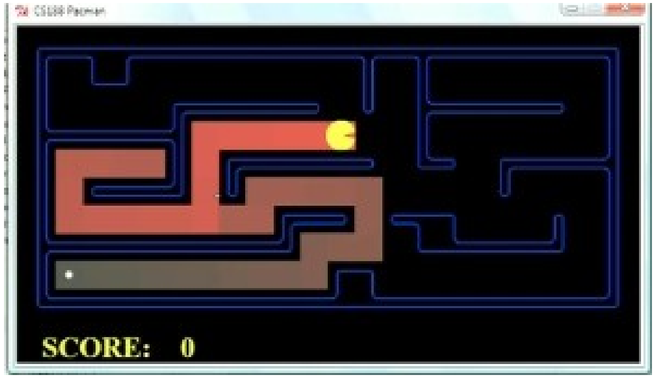
- Uniform-cost expands equally in all “directions”



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



A* vs UCS



Greedy

UCS

A*

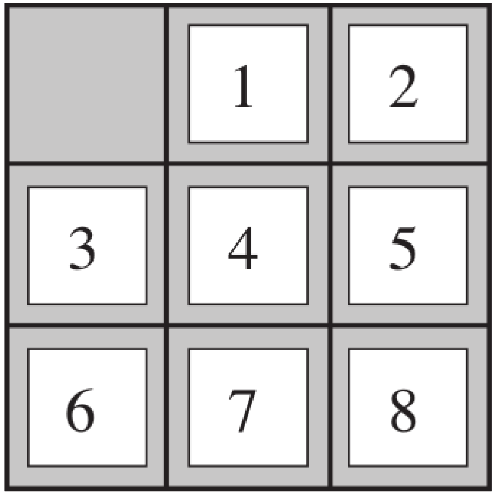
Slide: Adapted from Berkeley CS188 course notes (downloaded Summer 2015)

Choosing a heuristic

The right heuristic is often problem-specific.

But it is very important to select a good heuristic!

Choosing a heuristic



Consider the 8-puzzle:

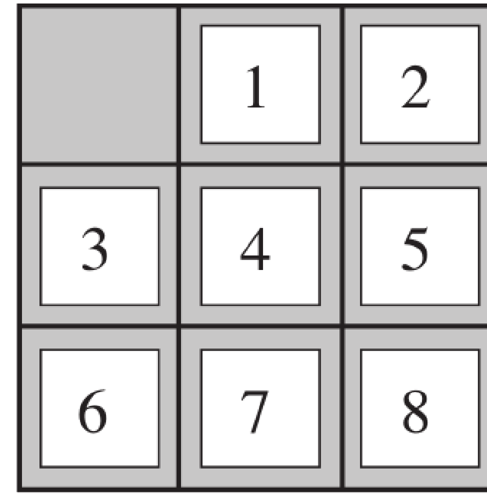
h_1 : number of misplaced tiles

h_2 : sum of manhattan distances between each tile and its goal.

How much better is h_2 ?

Choosing a heuristic

<u>Consider the 8-puzzle:</u>



h_1 : number of misplaced tiles

h_2 : sum of manhattan distances between each tile and its goal.

Average # states expanded on a random depth-24 puzzle:

$$A^*(h_1) = 39k$$

$$A^*(h_2) = 1.6k$$

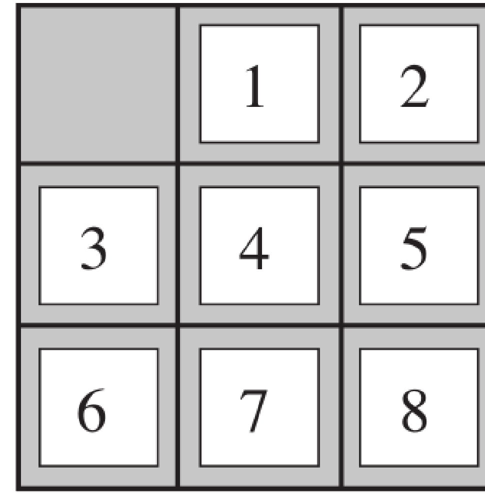
$$IDS = 3.6M$$ (by depth 12)

Choosing a heuristic

<u>Consider the 8-puzzle:</u>

h_1 : number of misplaced tiles

h_2 : sum of manhattan distances between each tile and its goal.

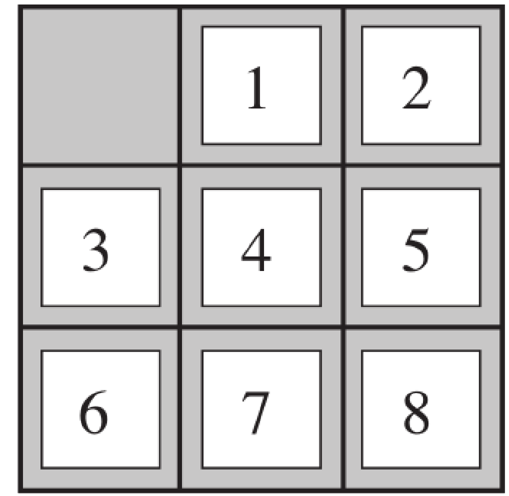


zle:

So, getting the heuristic right can speed things up by multiple orders of magnitude!

$IDS = 3.6M$ (by depth 12)

Choosing a heuristic



Consider the 8-puzzle:

h_1 : number of misplaced tiles

h_2 : sum of manhattan distances between each tile and its goal.

Why not use the actual cost to goal as a heuristic?

How to choose a heuristic?

Nobody has an answer that always works.

A couple of best-practices:
– solve a relaxed version of the problem
– solve a subproblem