Lecture 3: Informed Search

THESE SLIDES ARE ADOPTED FROM BERKELEY COURSE MATERIALS AND RUSSELL AND NORVIG TEXTBOOK

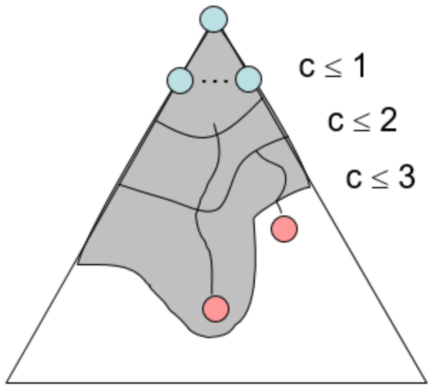
Search

Search problem:
- States (configurations of the world)
- Initial state
- Actions and costs
- Transition model (Successor function describing world dynamics)
- Goal test

Search algorithms:
- Systematically builds a search tree
- Chooses an ordering of the frontier
- Optimal: finds least path cost solution

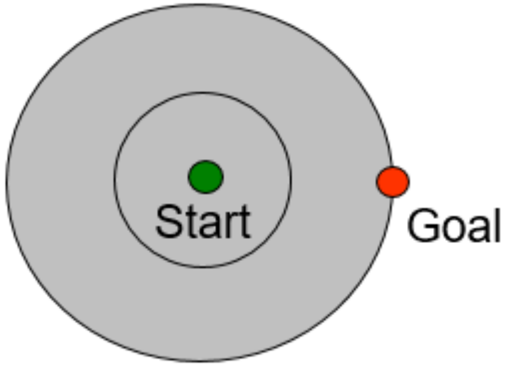
Uniform cost Search

Strategy: expand the lowest path cost



Advantages:

UCS is complete and optimal!



Problems:

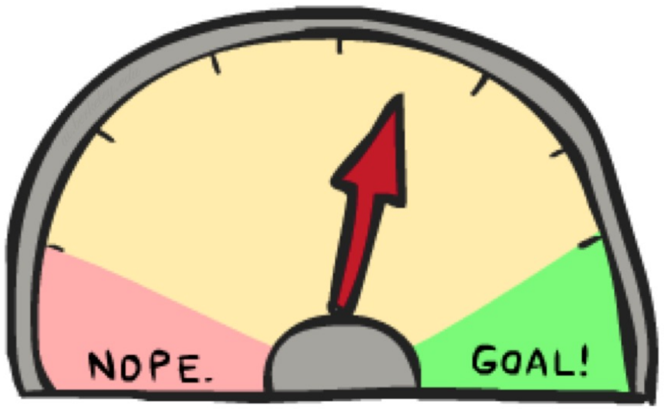
- Explores options in every “direction”
- No information about goal location

Video of Demo Contours UCS

Video of Demo Contours UCS Pacman Small Maze

Informed Search

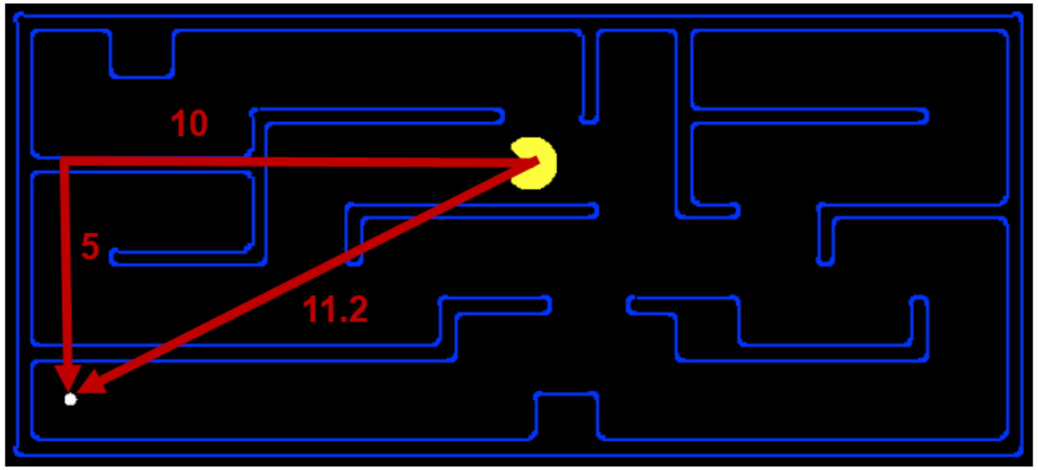
- **Informed search** uses problem-specific knowledge beyond the definition of the problem itself, so it can find solutions more efficiently than uninformed search.
- Problem-specific knowledge beyond the definition of the problem itself is represented using heuristic functions.
- Informed Search algorithms:
  - **Greedy best-first search**
  - **Astar (A*) search**



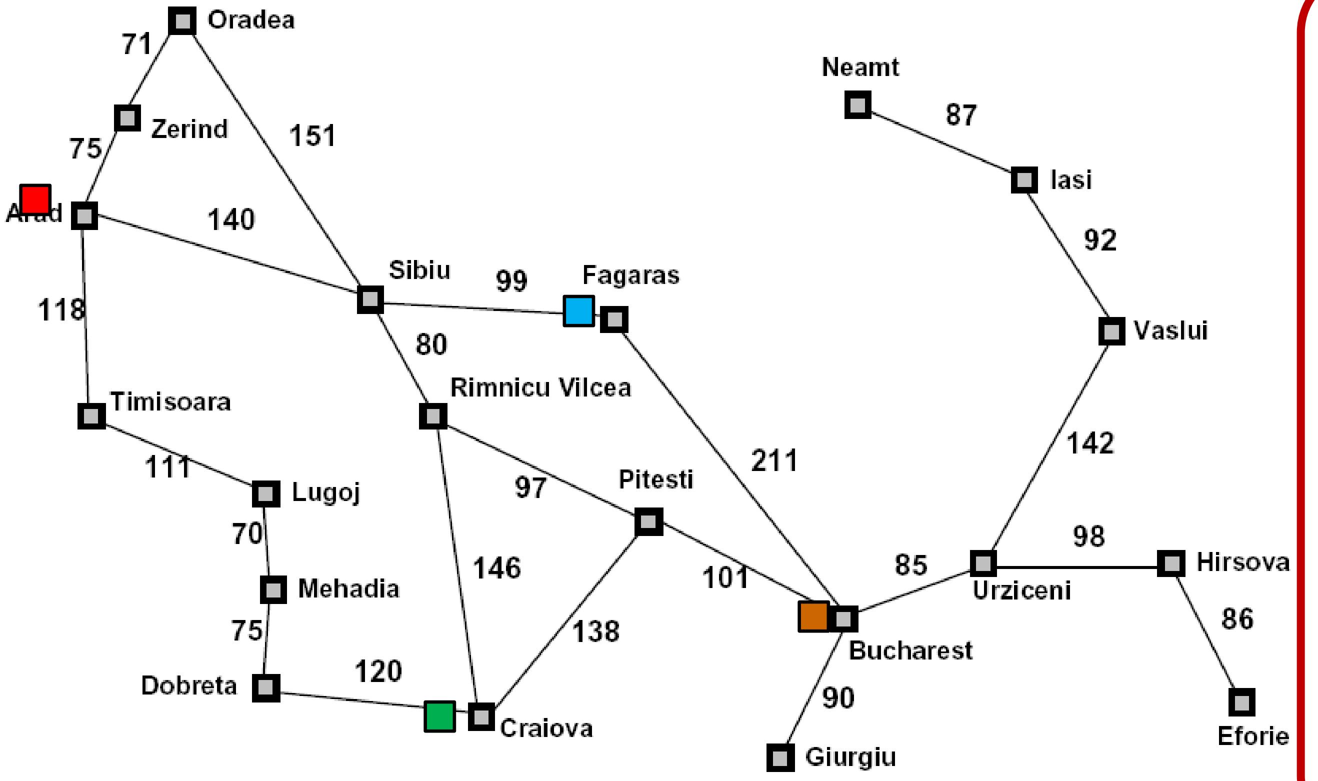
Search Heuristics

A heuristic function h(n) is:

- A function that *estimates* how close a state is to a goal
- h(n) = estimated cost of the cheapest path from the state at node *n* to a goal state.
- Designed for a particular search problem.
- Examples for Pacman pathing problem: Manhattan distance and Euclidean distance.



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

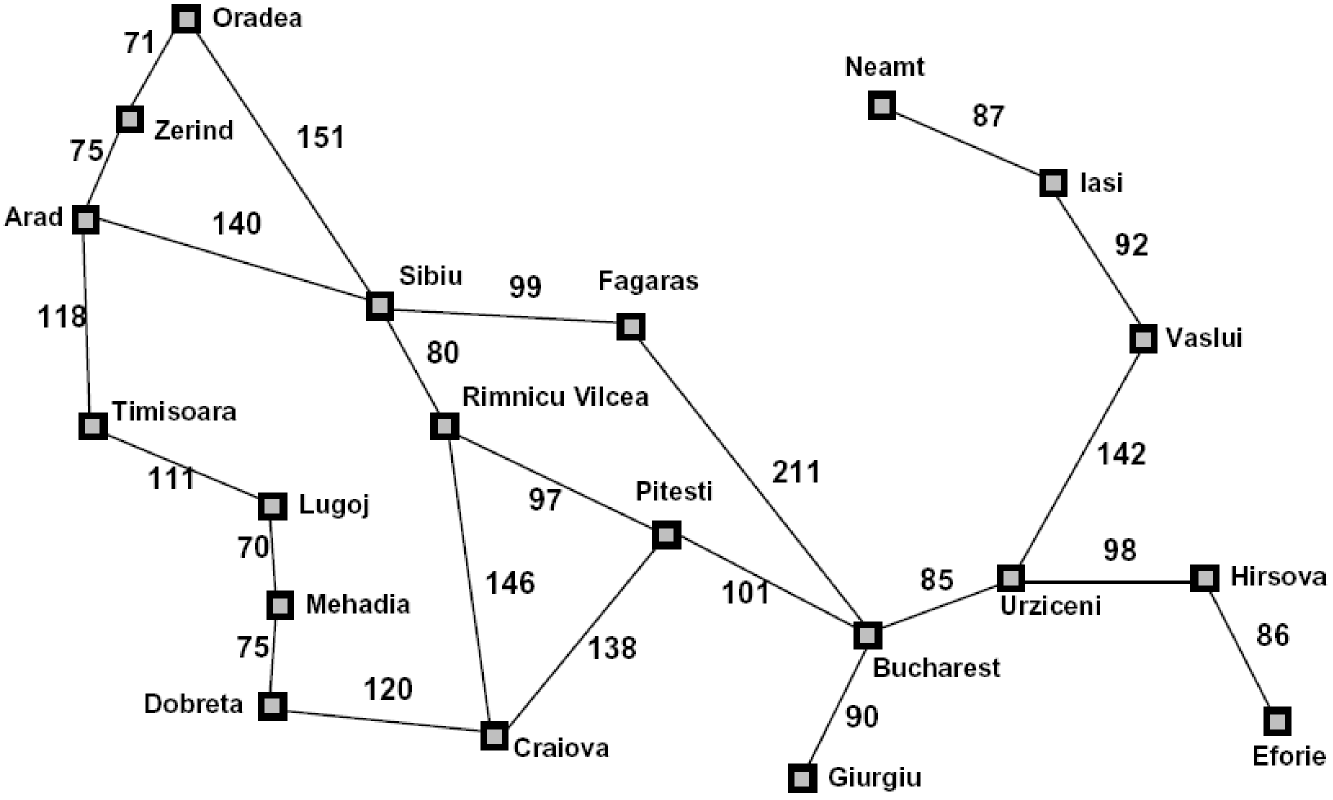
h(x)

Greedy best-first search

- Expand the node that *seems closest* to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes only using the heuristic function h(n).

- Do you think that geedy best-first search is optimal? Why?

Example: Heuristic Function

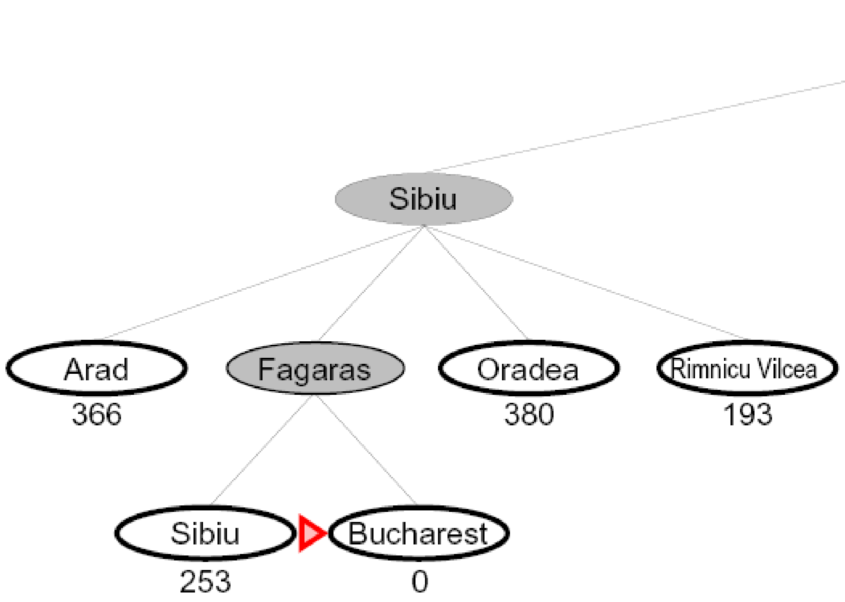


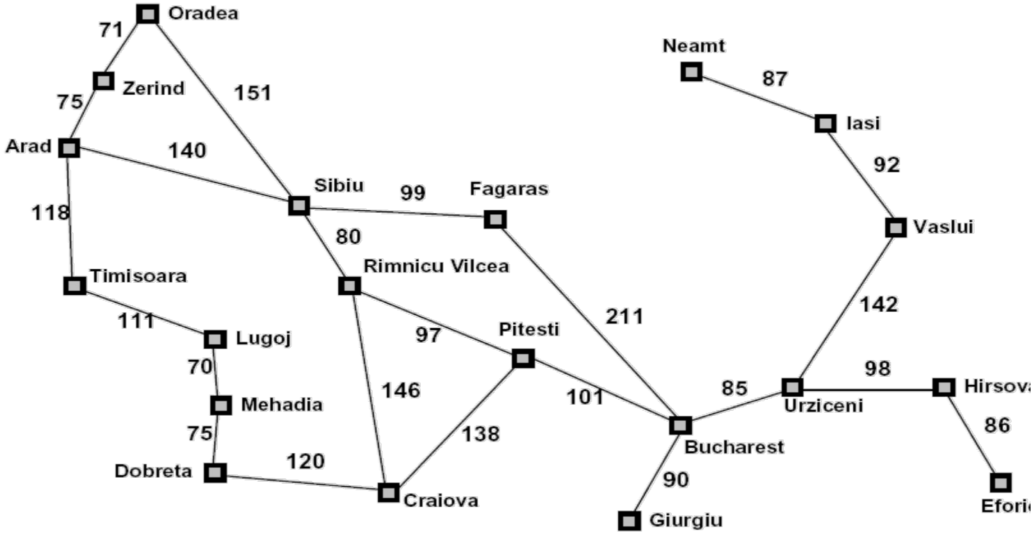
Straight−line distance to Bucharest

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 best first tree search: Example





Is it optimal?

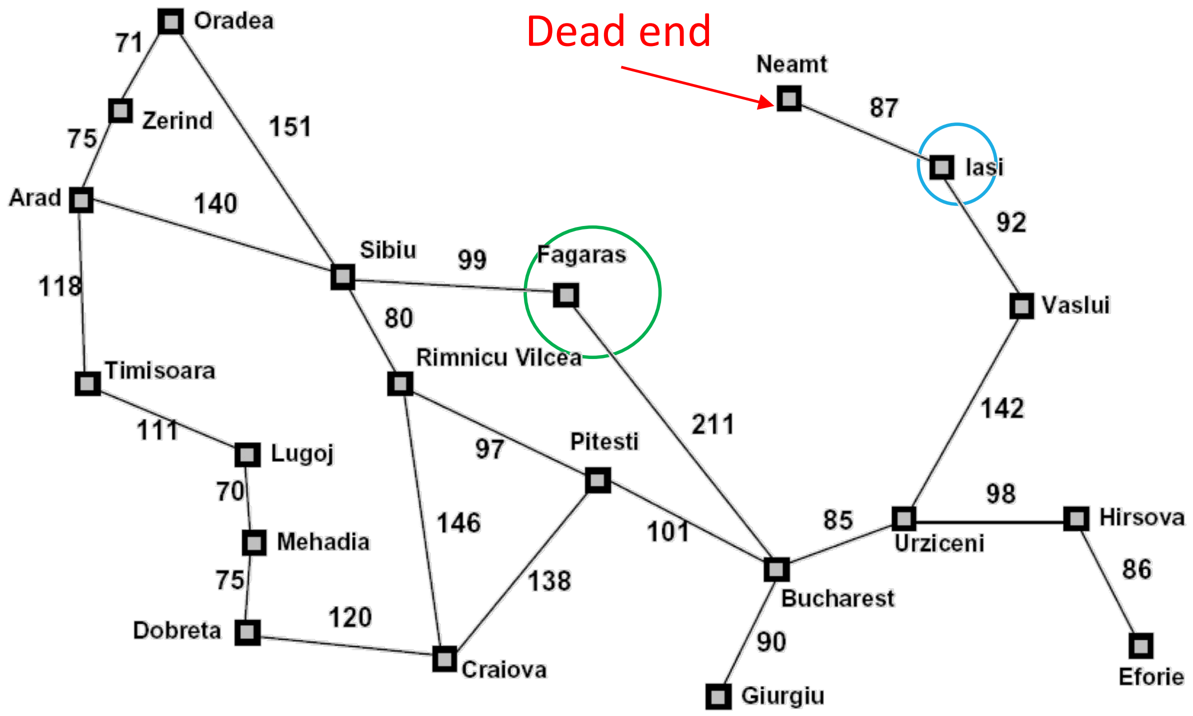
- No. Resulting path to Bucharest is not the shortest!

Greedy best-first Search

- Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.

Example: Go from Iasi to Fagaras.

- The heuristic function (straight line distance) suggests that Neamt is closer to the goal, but it is a dead end.



- Greedy best-first graph search is complete in finite state spaces, but not in infinite ones.

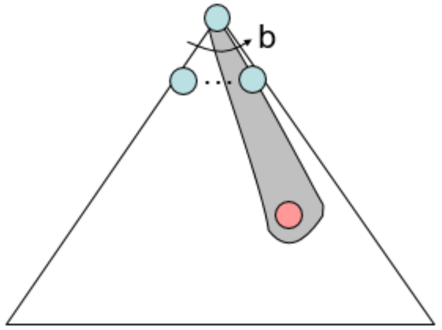
Greedy best-first Search

- The worst-case time and space complexity for the tree version is $O(b^m)$, where m is the maximum depth of the search space.
- Using a good heuristic function can substantially reduce the complexity.
- The amount of the reduction depends on the particular problem and on the quality of the heuristic.

Greedy best-first Search

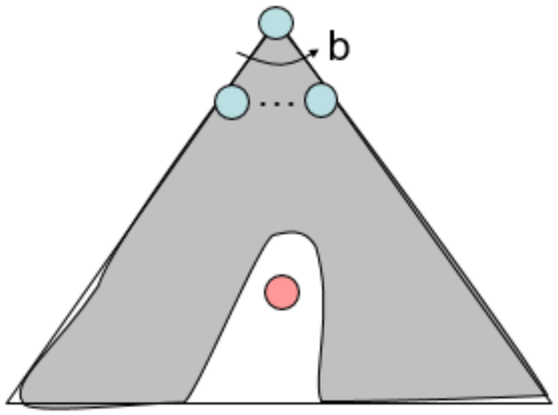
Strategy: expand a node that seems to be closest to a goal state

- Heuristic: estimate of distance to nearest goal for each state



A common case:

- Best-first takes you straight to the (wrong) goal



Worst-case: like a badly-guided DFS

Video of Demo Contours Greedy- straight line distance heuristic

Video of Demo Contours Greedy (Pacman Small Maze)-Manhattan distance heuristic

A* Search

A* Search

- A* search **minimizes the total estimated solution cost f(n).**

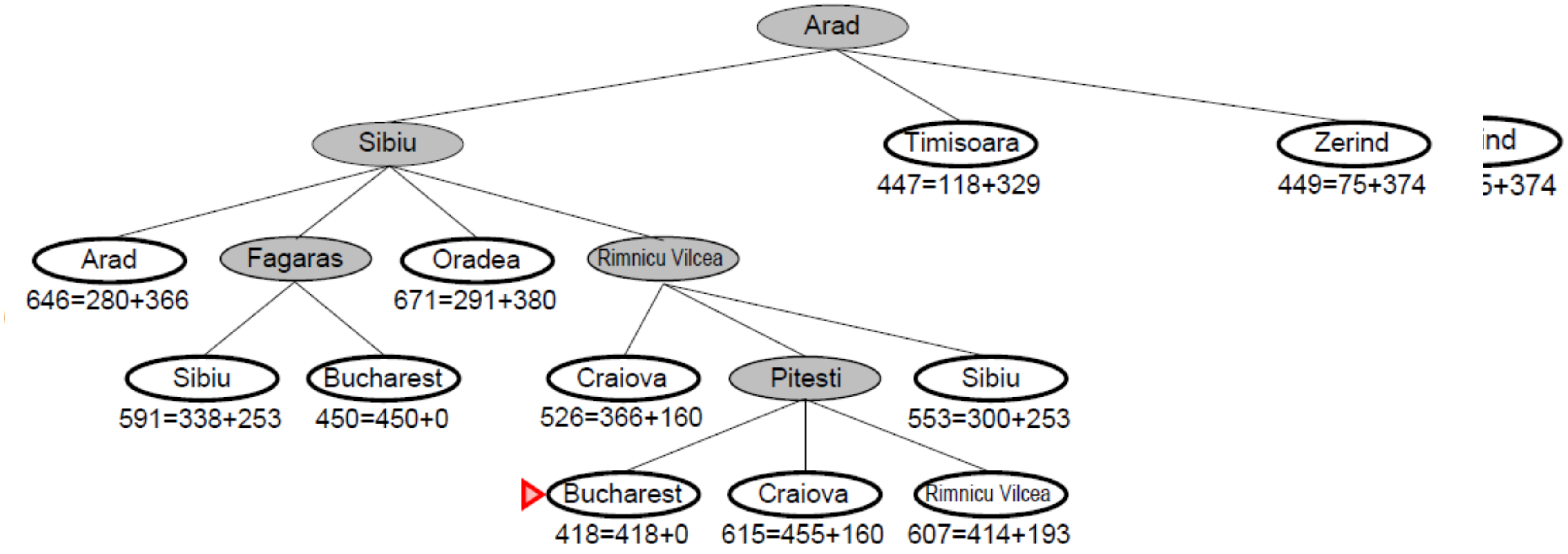
f(n) = g(n) + h(n)

g(n): the actual path cost from the initial state to state n. (backward cost)

h(n): the estimated heuristic function from the node n to the goal. (forward estimated cost)

- Thus, f(n) = estimated cost of the cheapest solution through n .

A*search- Romanian Map Example



A* Search Contours

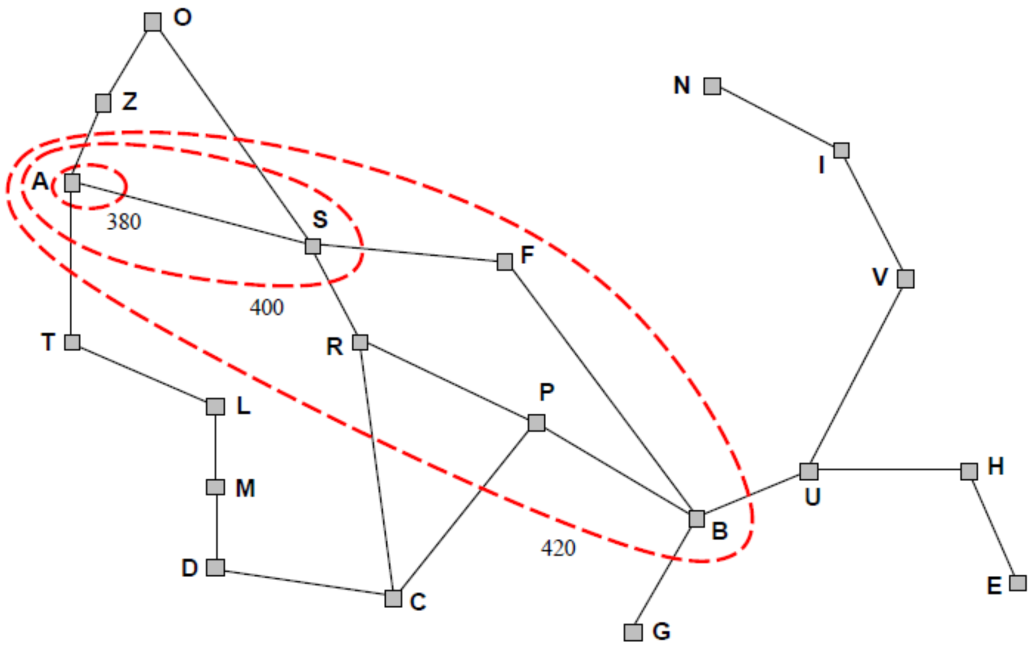
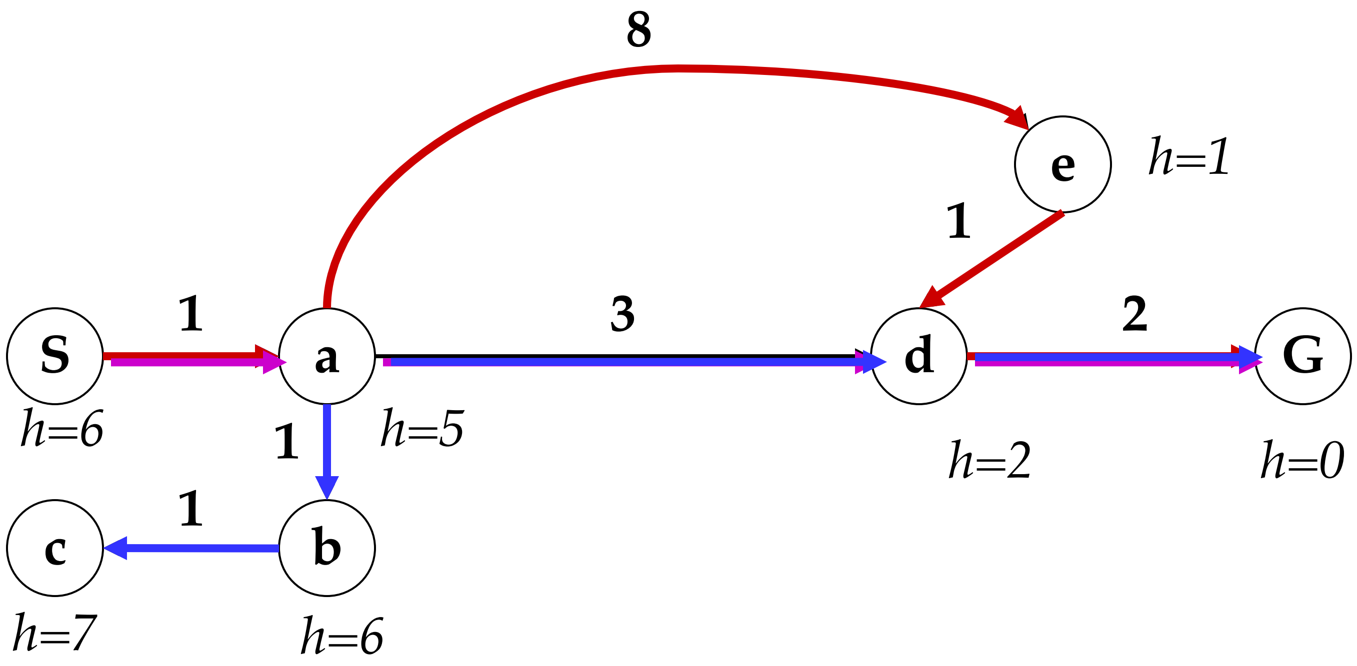


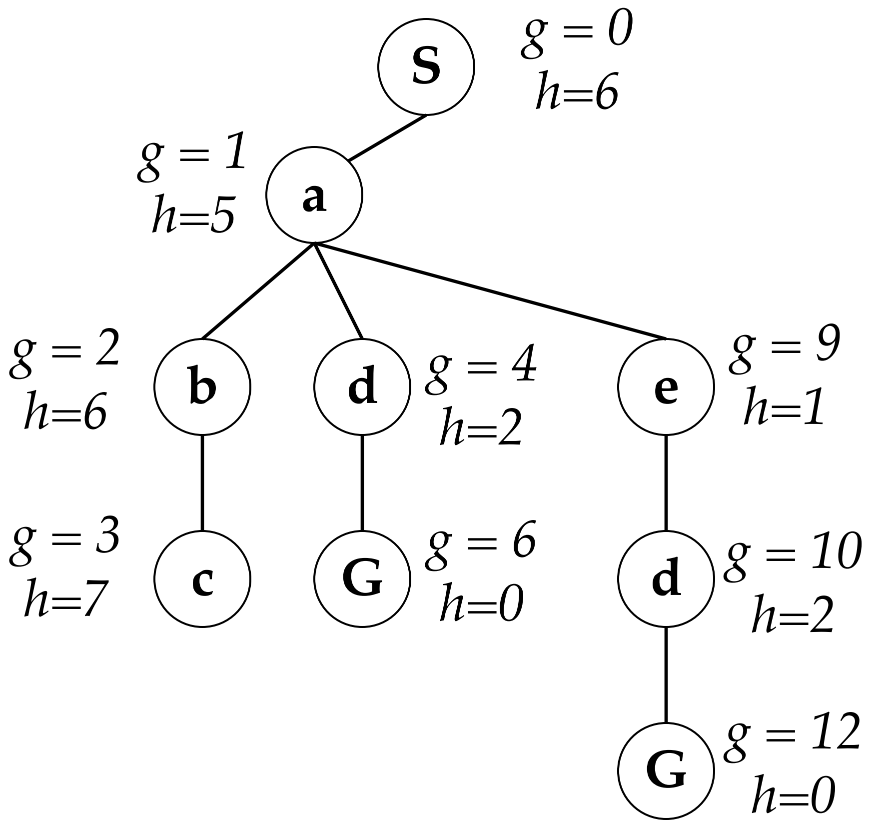
Figure 3.25 ap of Romania showing contours at *f*=380, *f*=400 and *f*=420, with Arad as the start state. Nodes inside a given contour have *f*-costs lower than the contour value.

A* Search- Example 2

Uniform-cost orders by path cost, or *backward cost* g(n)

Greedy orders by goal proximity, or *forward cost* h(n)

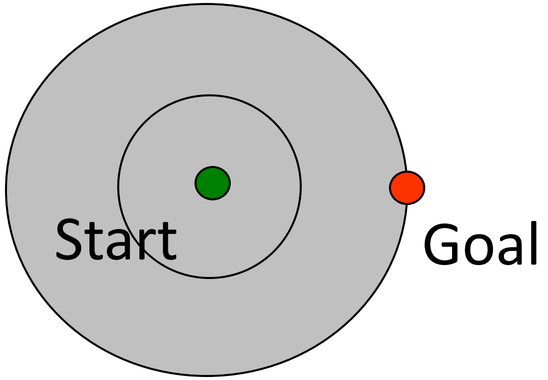




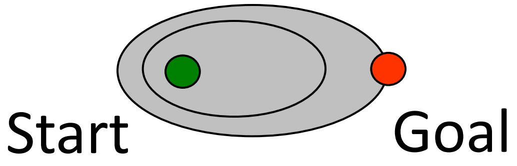
A* Search orders by the sum: f(n) = g(n) + h(n)

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



Video of Demo Contours (Empty) -- UCS

Video of Demo Contours (Empty) -- Greedy

Video of Demo Contours (Empty) – A*

Video of Demo Contours (Pacman Small Maze) – A*

Comparison

Greedy

Uniform Cost

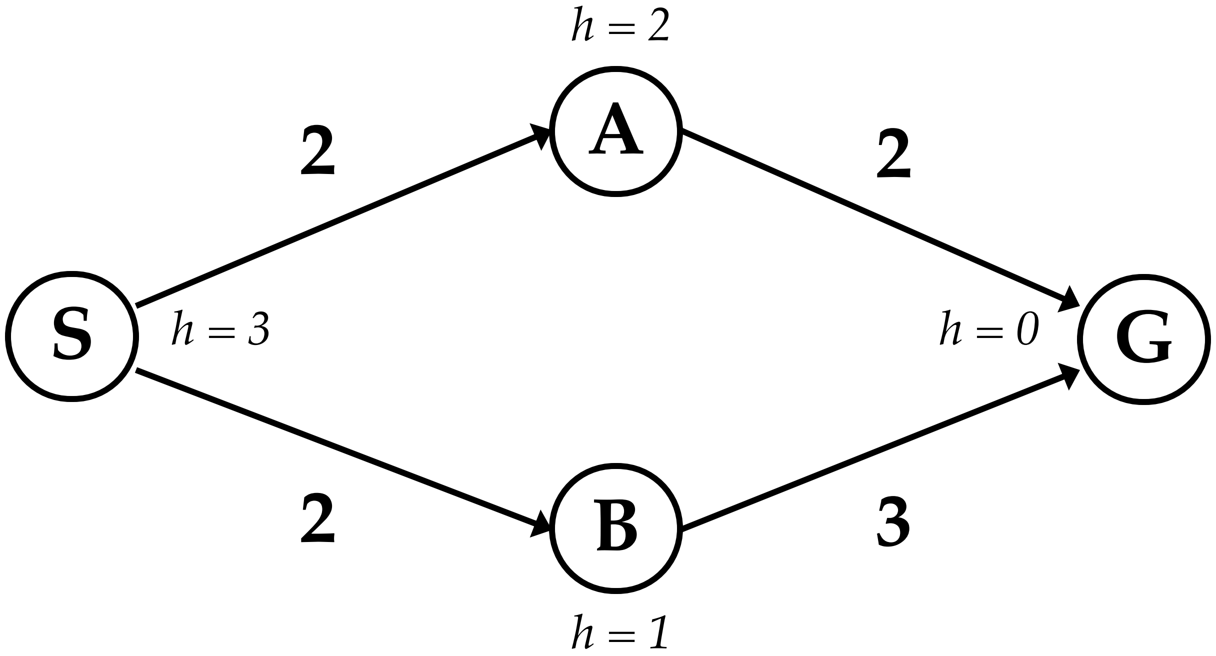
A*

A* Search Properties

- *A* search is complete.*
- *A* tree-search version is optimal if* h(n) *is admissible, while the graph-search version is optimal if* h(n) *is consistent.*

When should A* terminate?

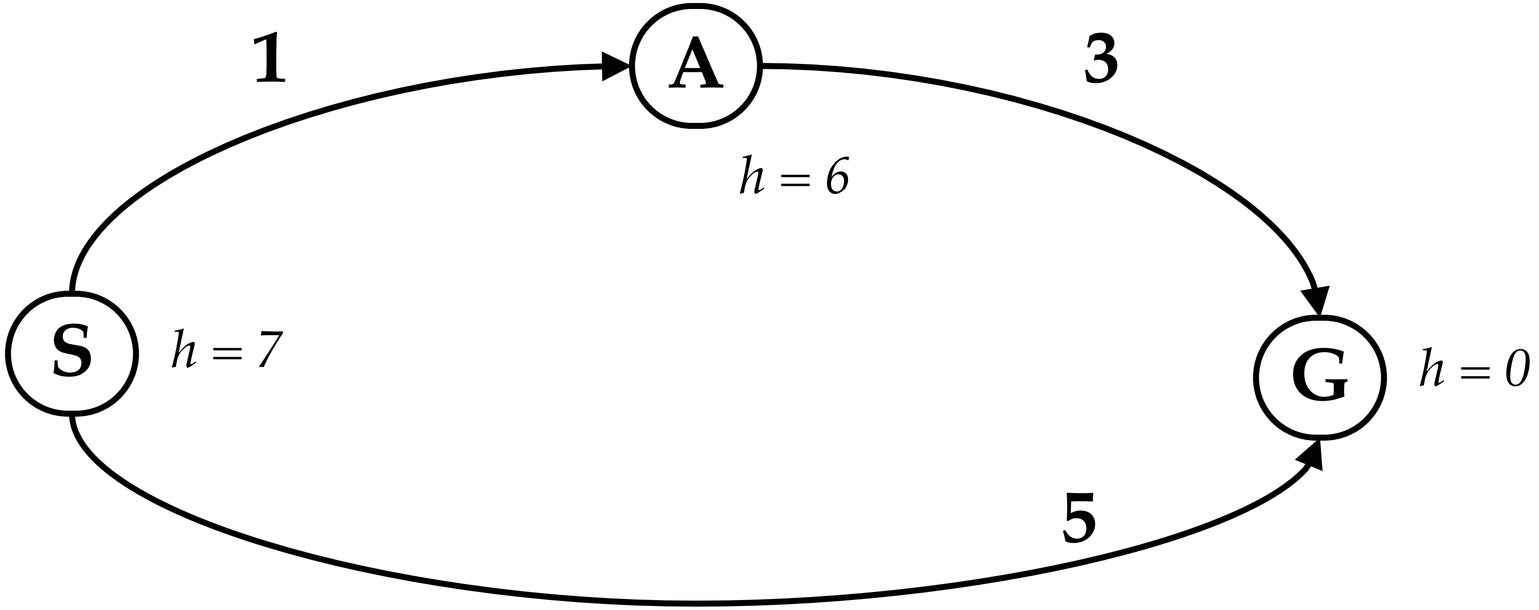
Should we stop when we insert a goal to the frontier?



Note : only stop when we dequeue (pick the node for expansion) a goal.

	g	h	+
~~S~~	~~0~~	~~3~~	~~3~~
~~S->A~~	~~2~~	~~2~~	~~4~~
~~S->B~~	~~2~~	~~1~~	~~3~~
S->B->G	5	0	5
$\boxed{\text{S->A->G}}$	$\boxed{4}$	$\boxed{0}$	$\boxed{4}$

Is A* Optimal?



	g	h	+
~~S~~	~~0~~	~~7~~	~~7~~
S->A	1	6	7
S->G	5	0	5

What went wrong?

Actual goal cost < estimated goal cost

We need estimates to be less than actual costs!

A* search optimality

- *A* tree-search version is optimal if* h(n) *is admissible, while the graph-search version is optimal if* h(n) *is consistent.*

- An admissible heuristic *never overestimates* the cost to reach the goal.
- g(n): the actual cost to reach n along the current path
- f(n)=g(n) + h(n)
- For admissible heuristics, f(n) never overestimates the true cost of a solution along the current path through n.
- Admissible heuristics are optimistic because they think the cost of solving the problem is less than it actually is.

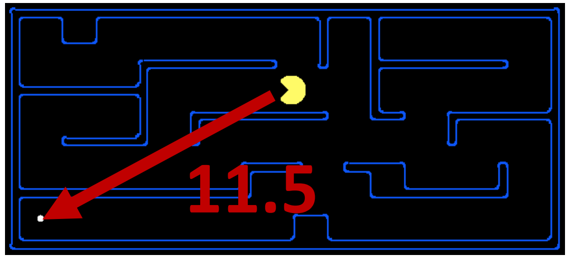
Admissible Heuristics

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

Example: Euclidean distance



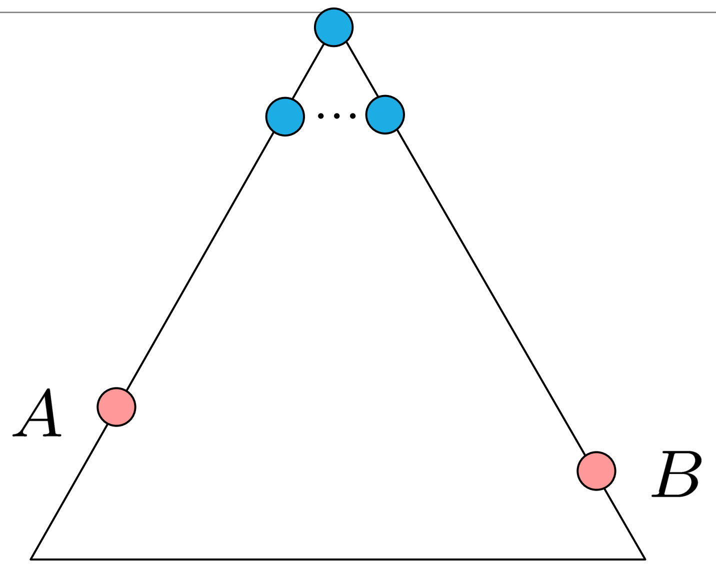
Optimality of A* Tree Search

Assume:

A is an optimal goal node

B is a suboptimal goal node

h is admissible



Claim:

A will exit the frontier before B

Optimality of A* Tree Search

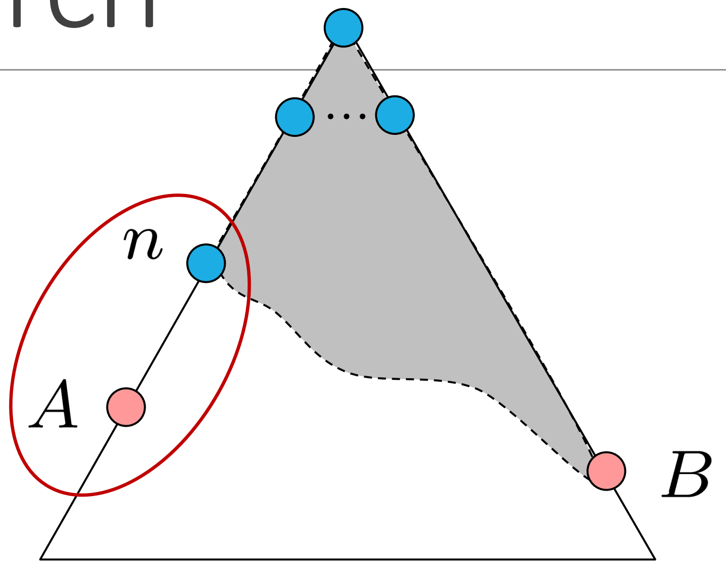
Proof:

Imagine B is on the frontier

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

Claim: *n* will be expanded before B

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



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

Optimality of A* Tree Search

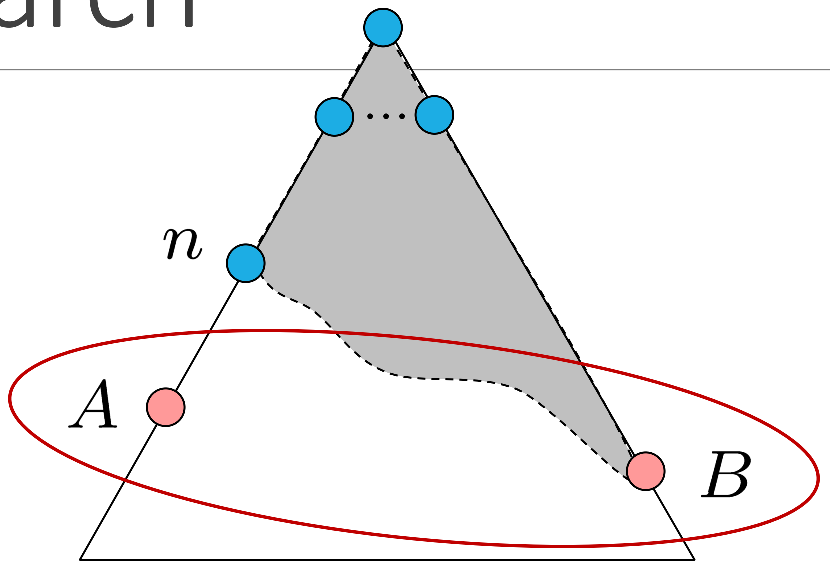
Proof:

Imagine B is on the frontier

Some ancestor *n* of A is on the frontier, 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

Proof:

Imagine B is on the frontier

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

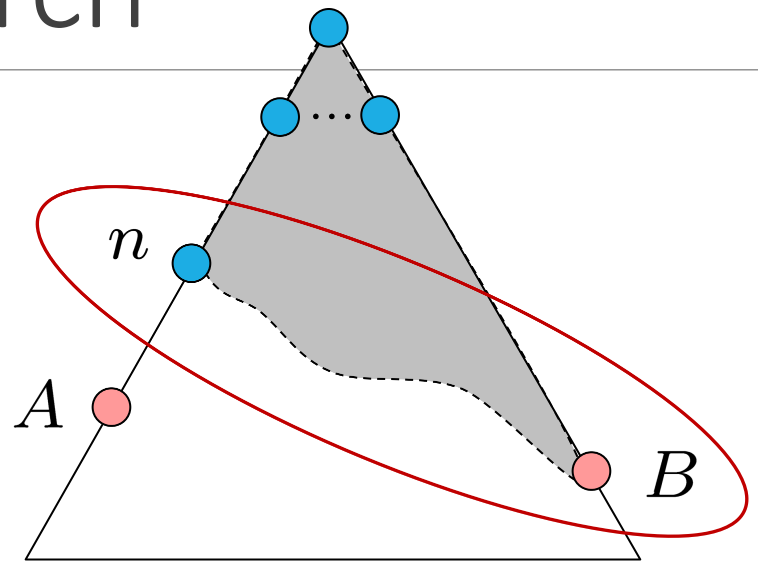
Claim: *n* will be expanded before B

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

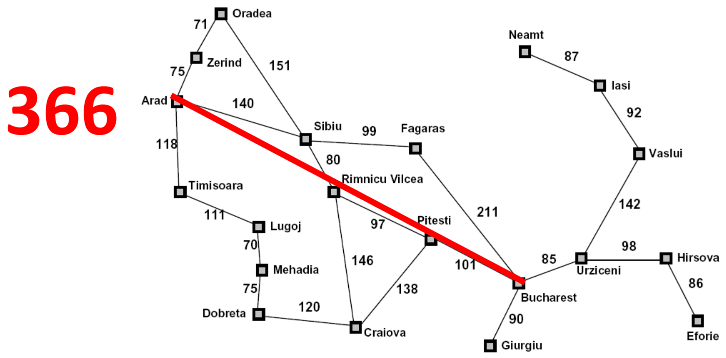


$$f(n) \leq f(A) < f(B)$$

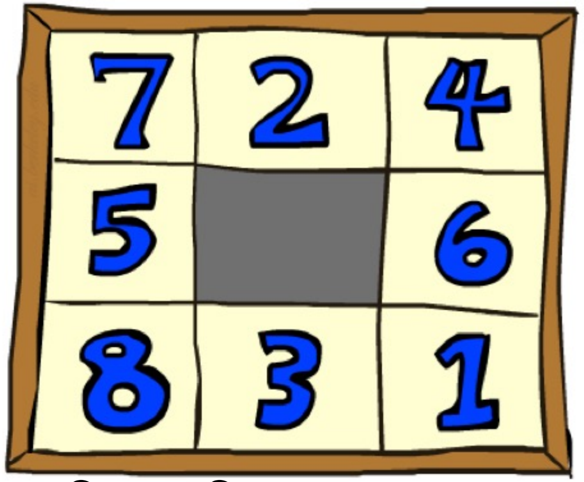
Creating Admissible Heuristics

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

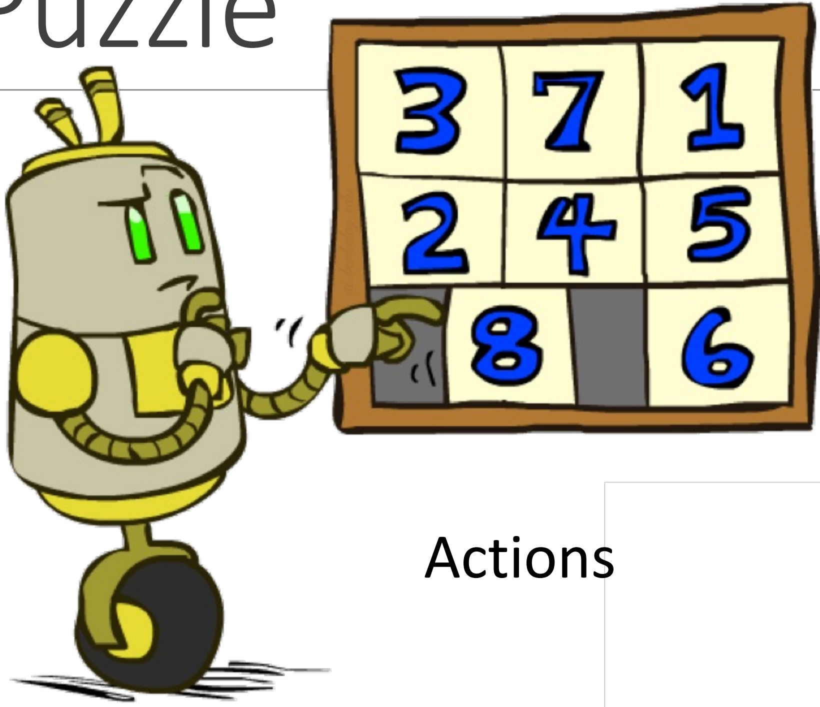
Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



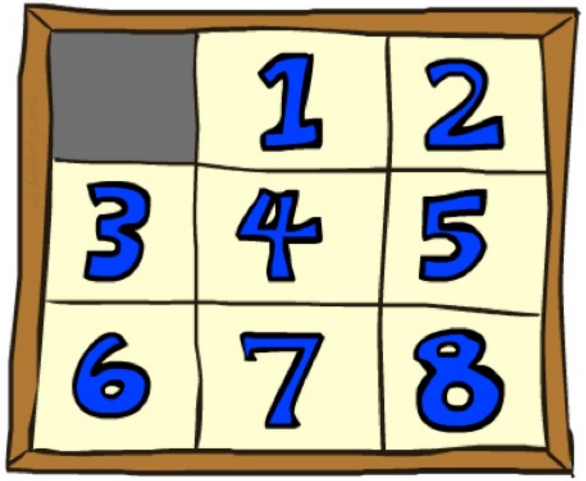
Example: 8 Puzzle



Start State



Actions



Goal State

What are the states?

How many states?

What are the actions?

How many successors from the start state?

What should the costs be?

Admissible heuristics?

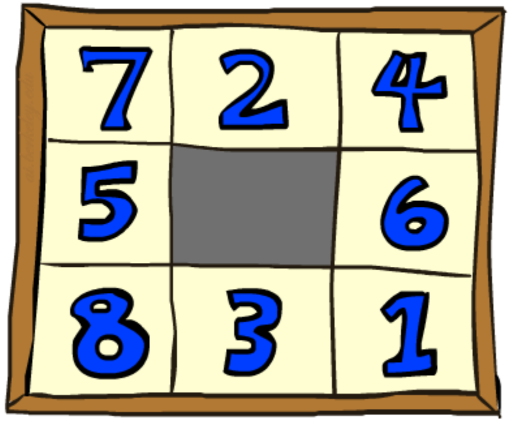
8 Puzzle I

Heuristic: Number of misplaced tiles

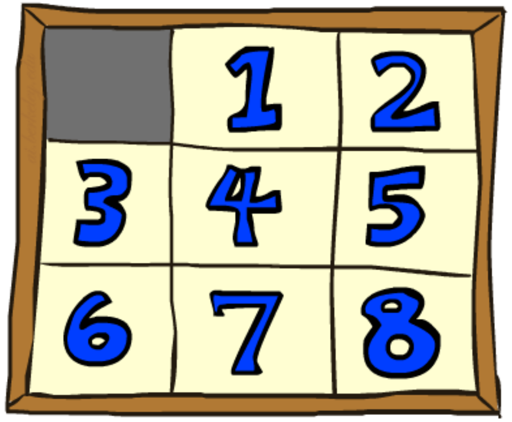
Why is it admissible?

h(start) = 8

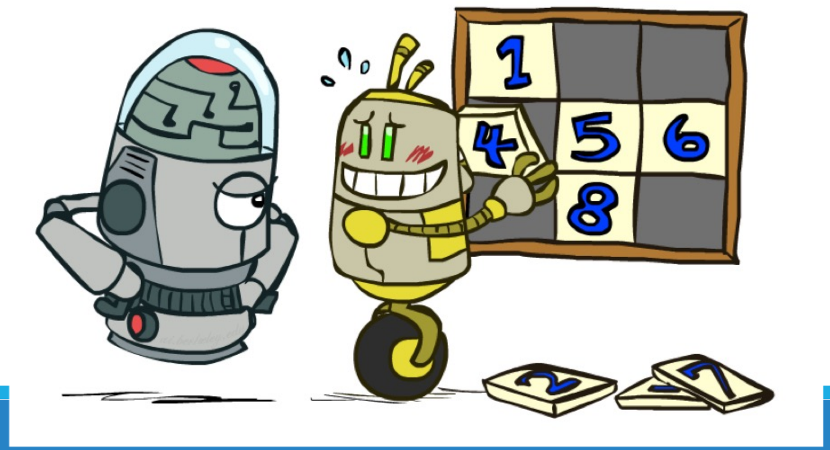
This is a *relaxed-problem* heuristic



Start State



Goal State



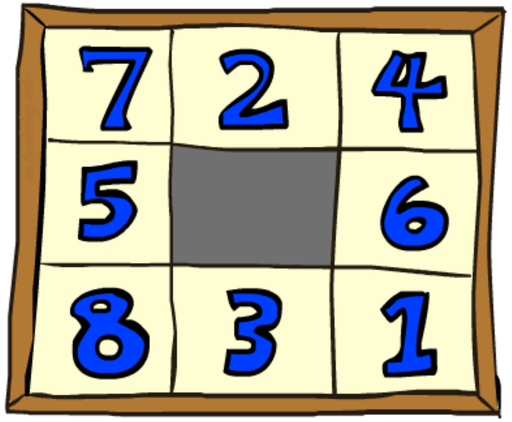
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6 x 10^6
TILES	13	39	227

Statistics from Andrew Moore

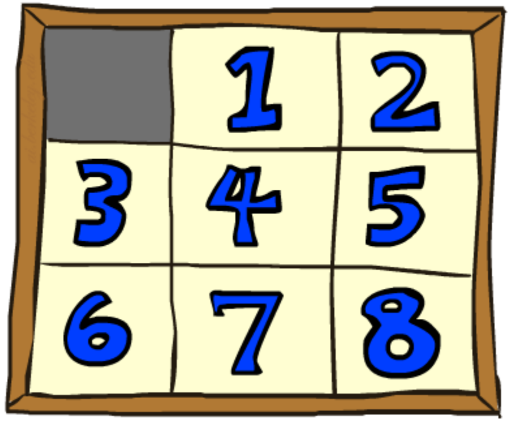
8 Puzzle II

Heuristic 2: The sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances.

This is sometimes called the **city block distance** or **Manhattan distance**.



Start State



Goal State

Is it admissible?

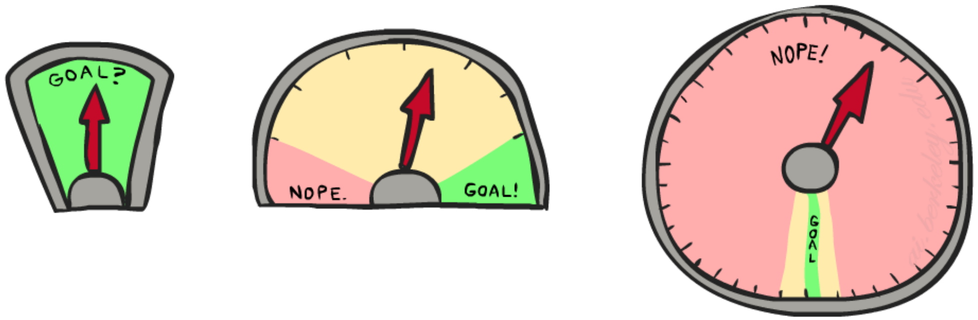
h(start) = $3 + 1 + 2 + \ldots = 18$

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

8 Puzzle III

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

Dominance of heuristics

Dominance: $h_a \geq h_c$ if

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

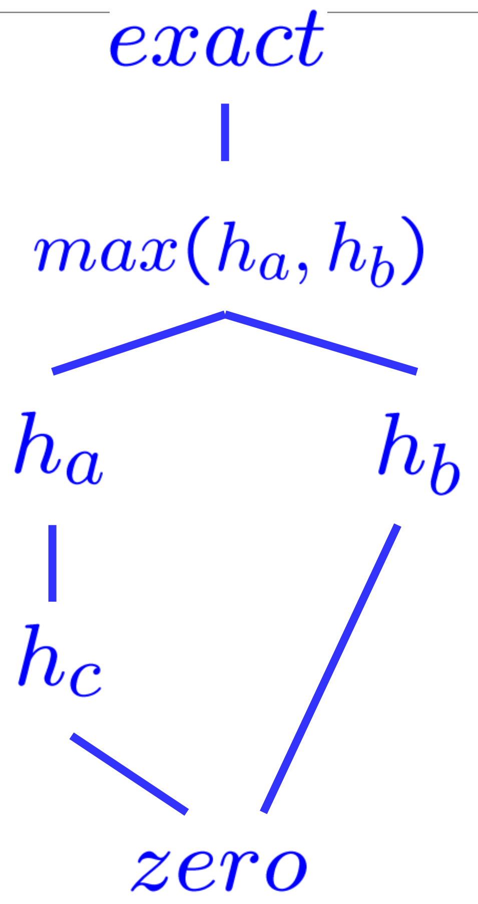
Heuristics form a semi-lattice:

- 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



Learning heuristics from experience

- “Experience” here means solving lots of 8-puzzles.
- Each optimal solution to an 8-puzzle problem provides examples from which h(n) can be learned.
- Each example consists of a state from the solution path and the actual cost of the solution from that point.
- From these examples, a learning algorithm can be used to construct a function h(n) that can (with luck) predict solution costs for other states that arise during search.

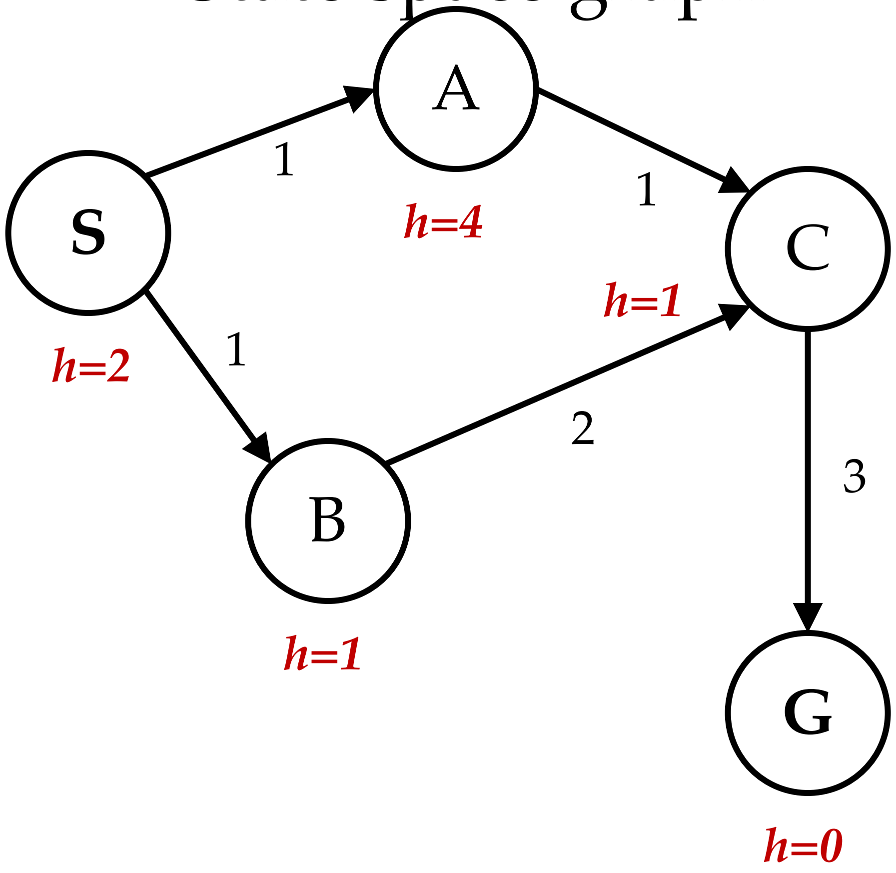
- For example, h(n) can be represented as a linear combination of features as follows:

$$h(n) = c_1 x_1(n) + c_2 x_2(n) .$$

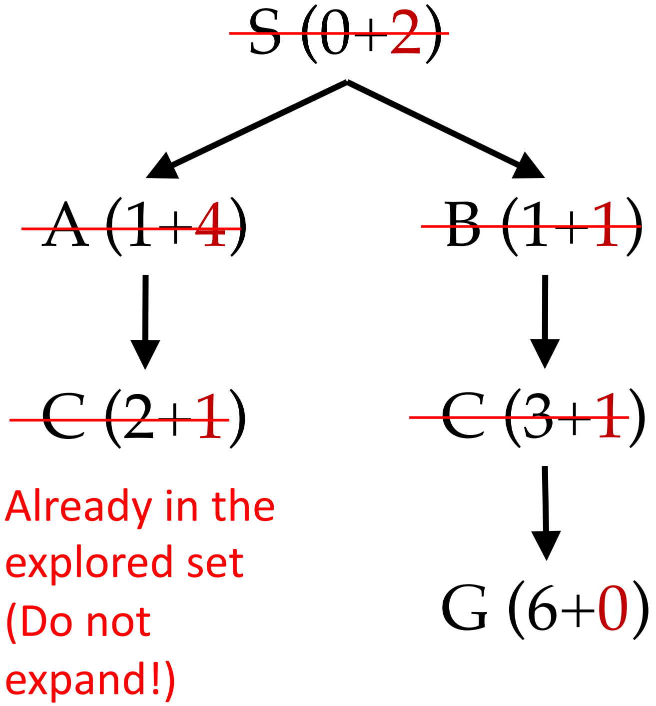
  - x_1 is the number of misplaced tiles and x_2 is the number of pairs of adjacent tiles that are not adjacent in the goal state.
  - The constants c_1 and c_2 are adjusted to give the best fit to the actual data on solution costs.

A* Graph Search Gone Wrong?

State space graph

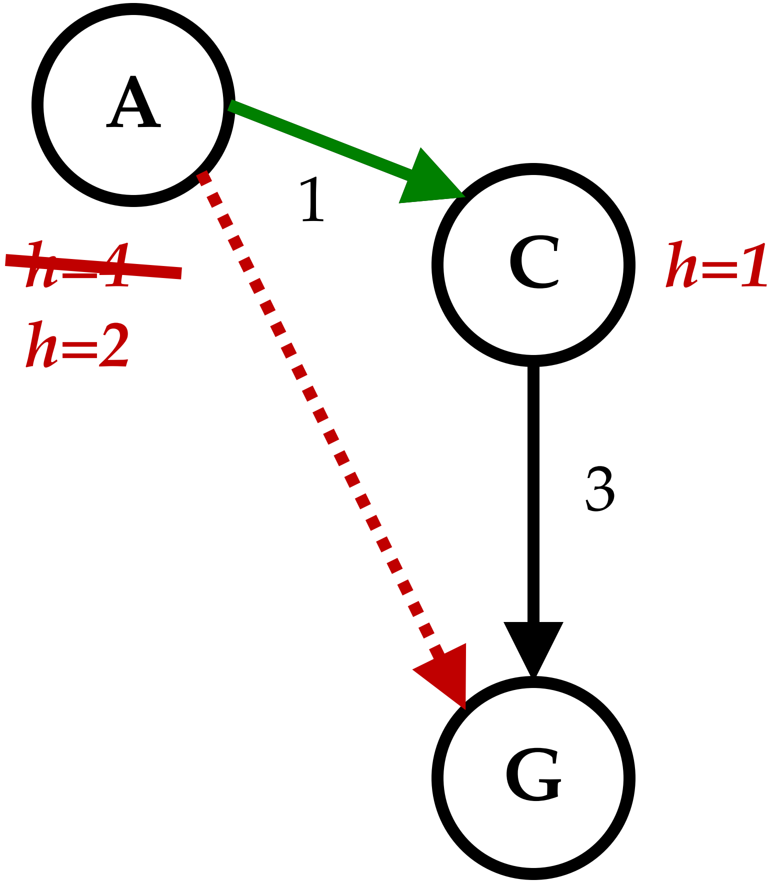


Search tree



Explored Set: S B C A

Consistency of Heuristics



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

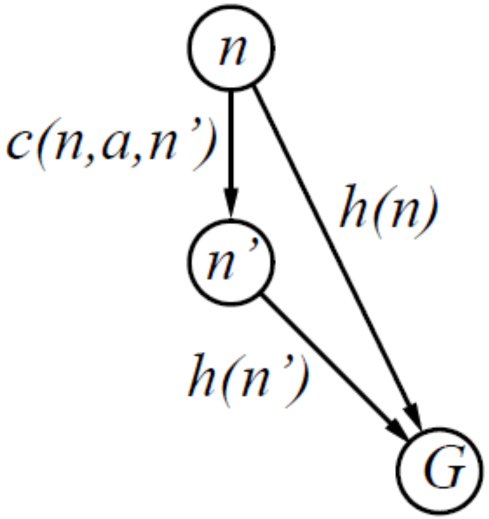
  h(A) ≤ cost(A to C) + h(C)

- A* graph search is optimal

Optimality of A* Graph Search

- *The A* graph-search is optimal if* h(n) *is consistent.*
- If h(n) is consistent:

h(n) ≤ c(n; a; n’) + h(n’)



If h is consistent, we have:

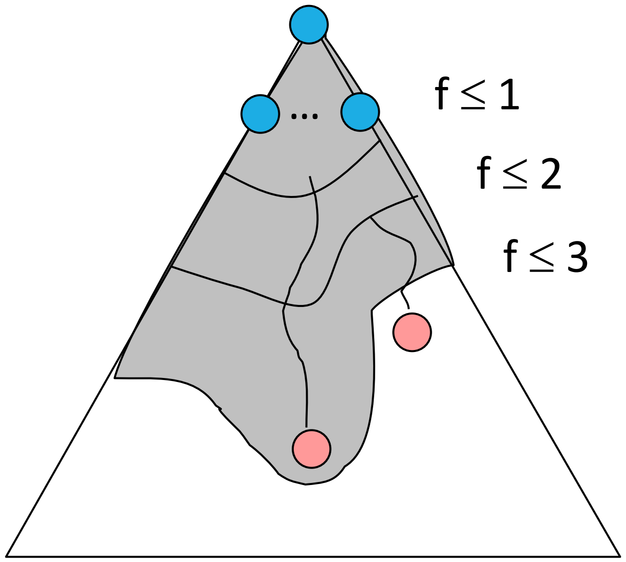
f(n’) = g(n’) + h(n’)
= g(n) + c(n; a; n’) + h(n’) as g(n’)=g(n)+c(n;a;n’)
>=g(n) + h(n) since h(n) is consistent
Then, f(n’)>= f(n)

Accordingly, *the values of f*(n) *along any path are nondecreasing.*

Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:

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



Optimality of A* Graph Search

Fact2 Proof:

New possible problem: some *n* on path to G* isn't in queue when we need it, because some worse *n'* for the same state dequeued and expanded first (disaster!)

Take the highest such *n* in tree

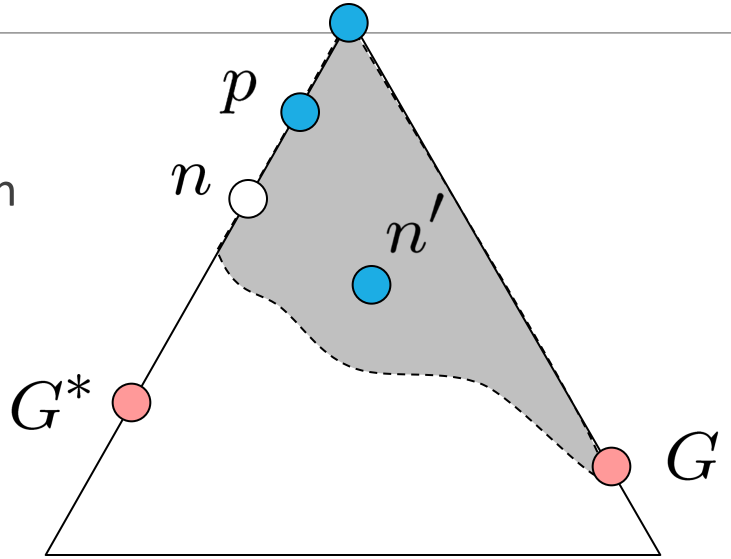
Let *p* be the ancestor of *n* that was on the queue when *n'* was popped

f(p) < f(n) because of consistency

f(n) < f(n') because *n'* is suboptimal

p would have been expanded before *n'*

Contradiction!



Optimality

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

Optimality of A* Search

With a admissible heuristic, Tree A* is optimal.

With a consistent heuristic, Graph A* is optimal.

A* Search Complexity

- The number of states within the goal contour search space is still exponential in the length of the solution.

- For constant step costs, we can write this as $O(b^{\varepsilon d})$, where d is the solution depth and ε is the relative error of the heuristic function.

$$\epsilon \equiv (h^* - h)/h^*.$$

- A* keeps all generated nodes in memory, so it has a space complexity problem.
- So, A* has exponential time and space complexity.

Memory-bounded heuristic search

- The simplest way to reduce memory requirements for A$*$ is to adapt the idea of iterative deepening to the heuristic search context, resulting in the **iterative-deepening A*** (IDA *) algorithm.

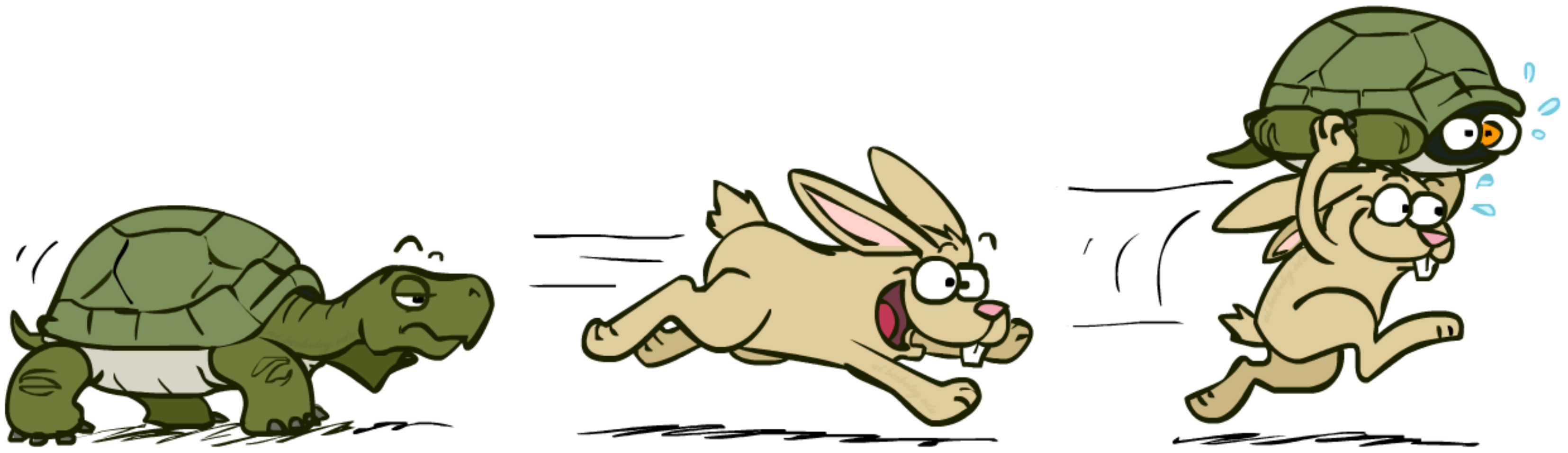
- The main difference between IDA * and the standard iterative deepening is that the cutoff used is the f-cost (g+h) rather than the depth; at each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.

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



A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Summary

- Heuristic (Informed Search)
- Greedy search
- A* search
- Heuristic functions
- Admissible heuristics
- Consistent heuristics
- Optimality of A* search
- Complexity of A* search