15-381: AI
Informed Search

Fall 2009

Manuela M. Veloso

Chapter 4, Russell and Norvig

Thanks to all past 381 instructors, and
http://www.cs.cmu.edu/~awm/tutorials

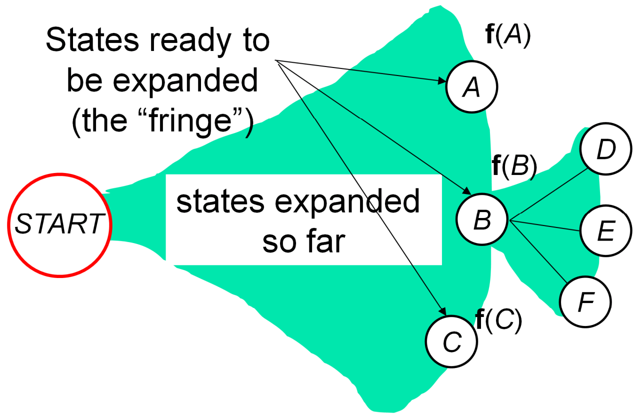
Carnegie Mellon

Uninformed Search Complexity

- N = Total number of states
- B = Average number of successors (branching factor)
- L = Length for start to goal with smallest number of steps
- Q = Average size of the priority queue
- $Lmax$ = Length of longest path from *START* to any state

	Algorithm	Complete	Optimal	Time	Space
BFS	Breadth First Search	Y	Y, If all trans. have same cost	$O(B^L)$	$O(B^L)$
BIBFS	Bi- Direction. BFS	Y	Y, If all trans. have same cost	$O(2B^{L/2})$	$O(2B^{L/2})$
PCDFS	Path Check DFS	Y	N	$O(B^{Lmax})$	$O(BL_{max})$
MEMDFS	Memorizing DFS	Y	N	$O(B^{Lmax})$	$O(B^{Lmax})$
IDS	Iterative Deepening	Y	Y, If all trans. have same cost	$O(B^L)$	$O(BL)$

General Search Revisited



1. Define some function **f**(*s*) at each state *s*
2. Choose the state with "lowest" **f** to expand next
3. Insert its successors

If **f**() is chosen carefully, we will eventually find the lowest-cost sequence

Note: In these examples s is just a state, not necessarily a start state

How does DFS expand? How do you decide which next node? Use a stack…expand the states most recently added to the stack.

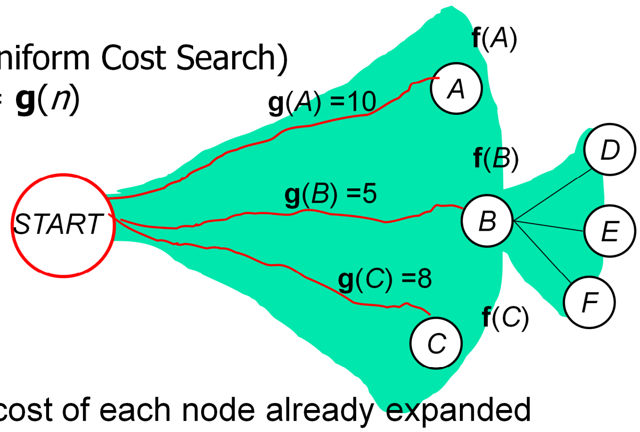
How does BFS expand? Use a queue…expand the states that were added first to the queue.

Uninformed vs Informed

- Uninformed – only guided by
  - *successor* relationships
  - topological structure (leftmost,…)
  - length as number of nodes

- Informed
  - assume *cost* of edges
  - more knowledge?

- UCS (Uniform Cost Search)
  f(*n*) = **g**(*n*)



- g(n) - cost of each node already expanded
  length of shortest path from START to n

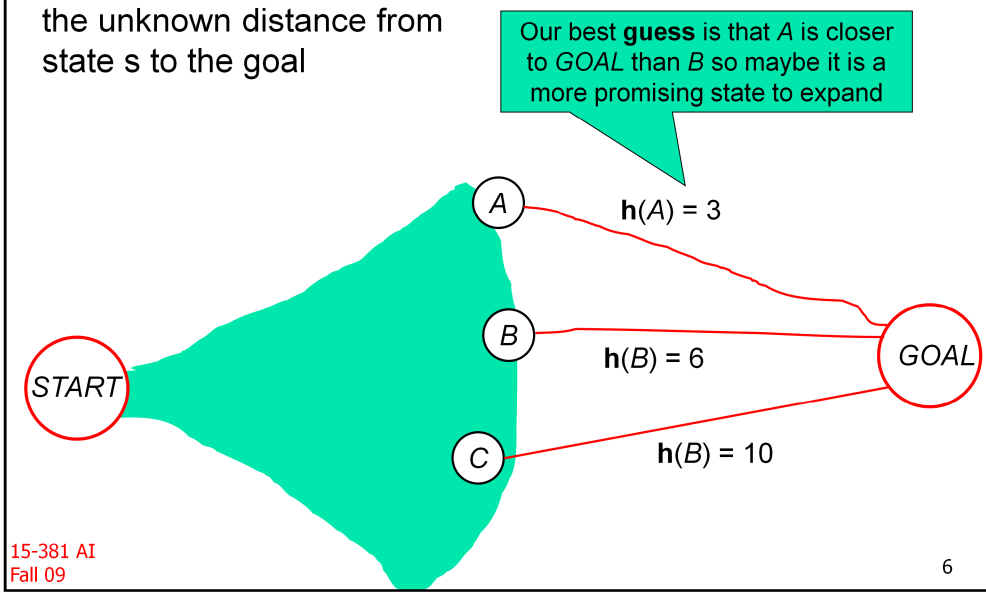
- Implementation – Store open successor states (waiting to be expanded) in a *priority queue* for efficient retrieval of minimum **f**

- Optimal → Guaranteed to find lowest cost sequence, *but guidance is about known path…*

Uninformed: has no information about how to get to the goal!

Estimate "Cost" to Goal

- Introduce a function **h**(s) **to estimate** the unknown distance from state s to the goal



We really want a "best guess"– as informed as possible.

Heuristic Functions

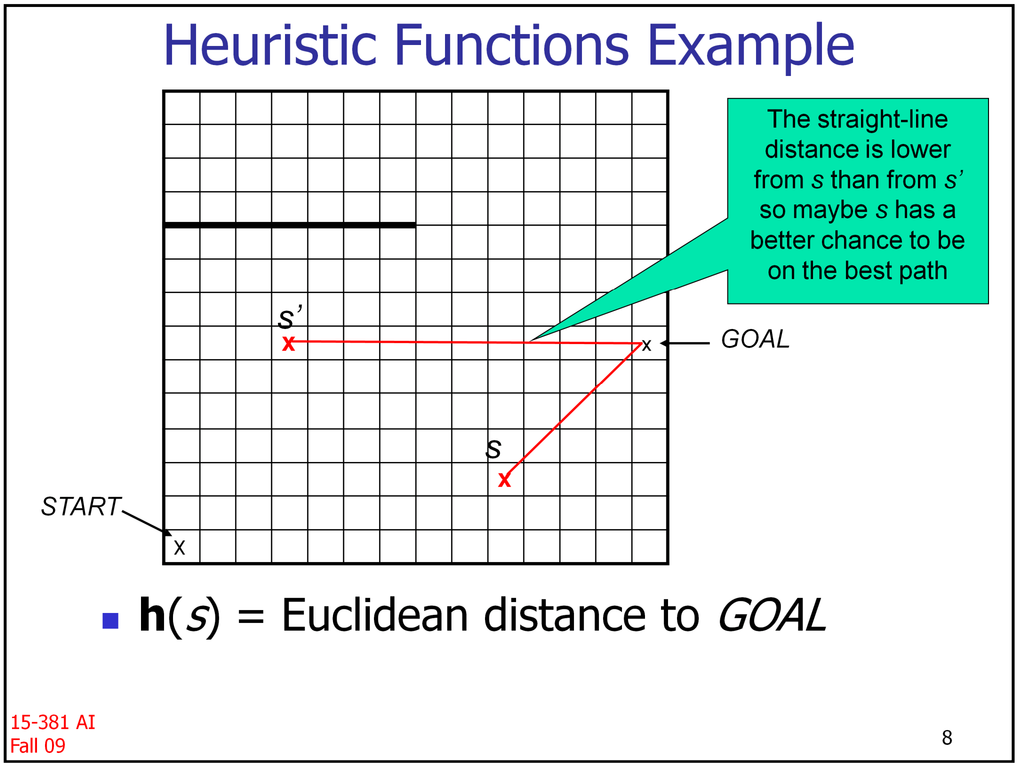
- **h** is a ***heuristic*** function for the search problem
- **h**(*s*) = estimate of the cost of the shortest path from *s* to GOAL
- **h** cannot be computed solely from the states and transitions in the current problem → If we could, we would already know the optimal path!
- **h**(.) is based on external knowledge about the problem → *informed* search
- Questions:
  1. Typical examples of **h**?
  2. How to use **h**?
  3. What are desirable/necessary properties of **h**?

15-381 AI
Fall 09

7

Informed: Concept of "external knowledge". Use more than just the state and the actions.

For those of you who have taken 15-211: You wrote a heuristic for your chess playing AI. Your heuristic would take the state (a state of the chess board) and output how good or bad the board is, which is an estimate of how close you are to winning.



You can only move North, South, East, or West. Manhattan distance is the number of moves you must make when moving this way on a grid. In this example, the Manhattan distance from S to the goal is 8 (4 North, 4 East).

Is Euclidean distance an accurate estimate?

No- because you move based on Manhattan distance.

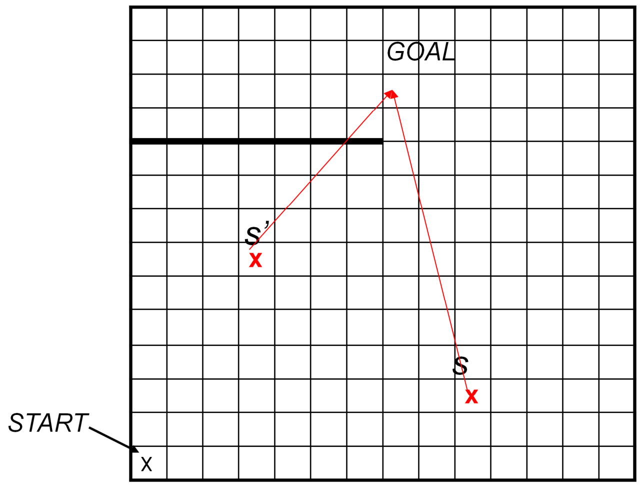
Is the Euclidean distance ever greater than the Manhattan distance? No.

Euclidean is a *lower bound*. It always underestimates the distance you must travel, since you can't go in a straight line or pass through walls.

Simon got the Nobel for introducing this concept of heuristic.

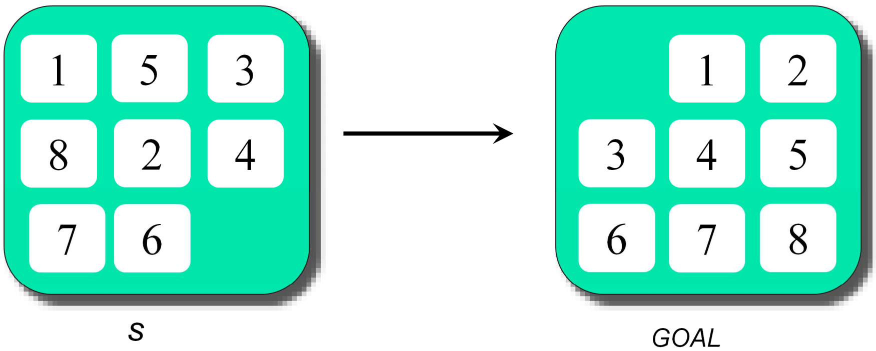
A guess of what's best, but not a proven best.

Heuristic Functions Example

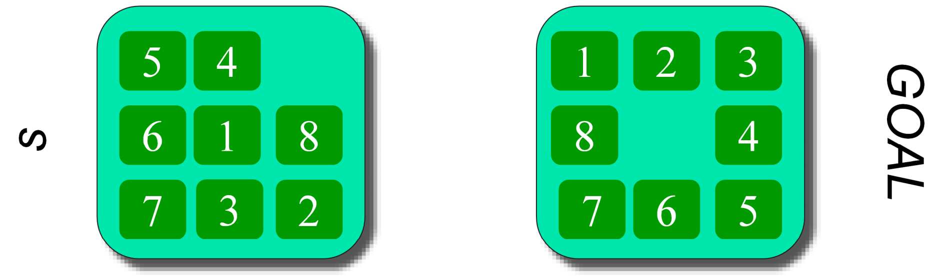


- **h**(*s*) = Euclidean distance to *GOAL*
- *Euclidean distance is an heuristic.*

Heuristic Functions Example



- How could we define **h**(*s*)?



Misplaced titles:

$\mathbf{h}_1(s) = 7$

Manhattan distance:

$\mathbf{h}_2(s) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$

Why is h2 better?

See Pearl- even more sophisticated heuristics.

First Attempt: Greedy Best First Search

- Simplest use of heuristic function: Alway node with smallest **h**(.) for expansion (i.e

> The heuristic value of a state is independent of the path to the state.

Initialize *PQ*
Insert *START* with value **h**(*START*) in *PQ*
While (*PQ* not empty and no goal state is in *PQ*)
  Pop the state *s* with the minimum value of **h** from *PQ*
  For all *s′* in **succs**(*s*)
    If *s′* is not already in *PQ* and has not already been visited
      Insert *s′* in *PQ* with value **h**(*s′*)

15-381 AI
Fall 09

12

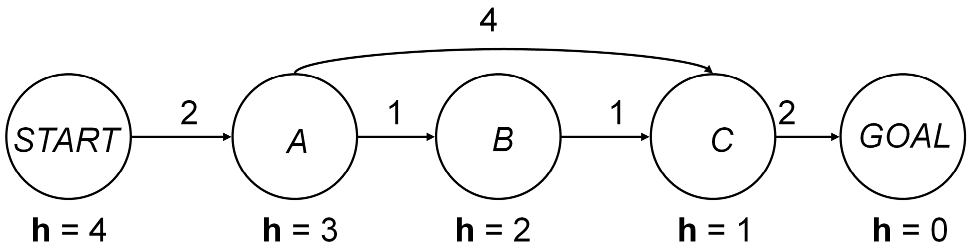
How is this different from UCS?

Heuristic value of state is a property of state, not the path to get to the state. (All step costs are the same).

h(s) is independent of path to reach s

BFS is generally good, but there can be some problems…

Problem



- What solution do we find in this case?
  - (START,4)
  - (A,3)
  - (C,1), (B,2)
  - (Goal,0)   START-A-C-Goal
- Greedy search clearly not optimal, even though the heuristic function is "good."

15-381 AI
Fall 09

13

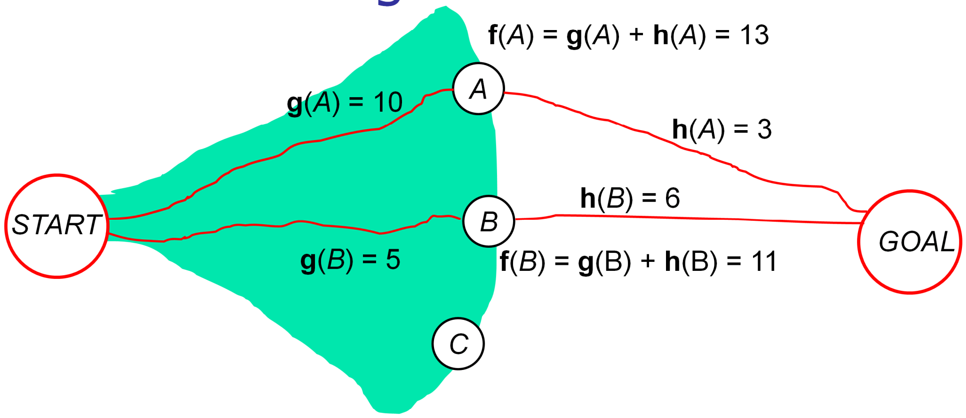
Queue for Greedy BFS is shown, and we always take the lowest h value. Path is first node in each step if sorted by best heuristic.

But this isn't the shortest path in terms of cost. So GBFS not optimal, even though heuristic function is "good".

H always *underestimates* shortest path to goal.

But GBFS is very easy to implement, and there are problems for which it is great.

Fixing the Problem



- **g**(*s*) **is** the (shortest cost so far) from *START* to *s* only
- **h**(*s*) **estimates** the cost from *s* to *GOAL*
- Key insight: **g**(*s*) + **h**(*s*) estimates the ***total*** cost of the cheapest path from START to GOAL going through *s*
- → A* algorithm

Can also do a modification– put weights:

F(n) = alpha * g(n) + (1-alpha) h(n). You can play around with the alphas.

A* Algorithm

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

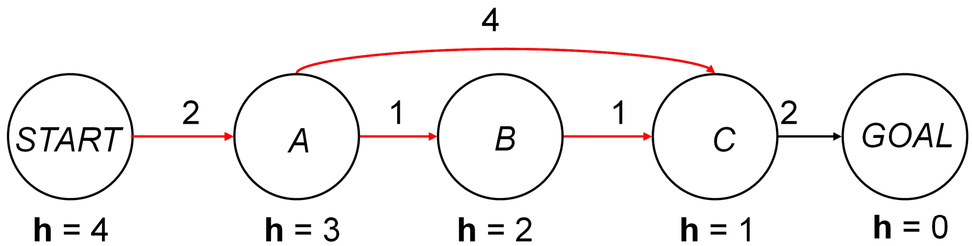
- heuristics
  - good, less good…, alternative
  - “easy” to define…

- efficiency

Heuristic should be easy to define– we don’t want heuristics to take longer than the search itself.

Defining good heuristics is very important and can be very complicated. For a search problem, choosing the heuristic is a big decision.

Can A* Fix the Problem?



{(*START*,4)}

{(*A*,5)}

(**f**(*A*)=**g**(*A*)+**h**(*A*) = **g**(*START*) + **cost**(*START*,*A*) + 3 = 0 + 2 + 3)

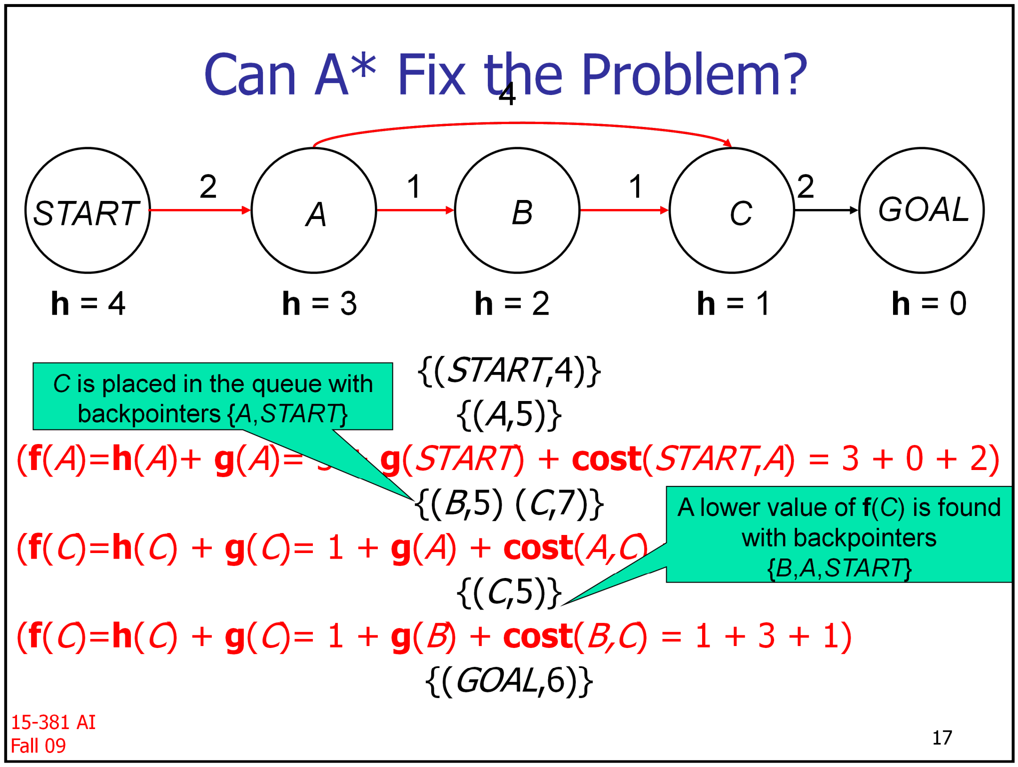
{(*B*,5) (*C*,7)}

(**f**(*C*)=**g**(*C*)+**h**(*C*) = **g**(*A*) + **cost**(*A*,*C*) + 1 = 2 + 4 + 1)

{(*C*,5)}

(**f**(*C*)= **g**(*C*)+**h**(*C*) = **g**(*B*) + **cost**(*B*,*C*) + 1 = 3 + 1 + 1)

{(*GOAL*,6)}



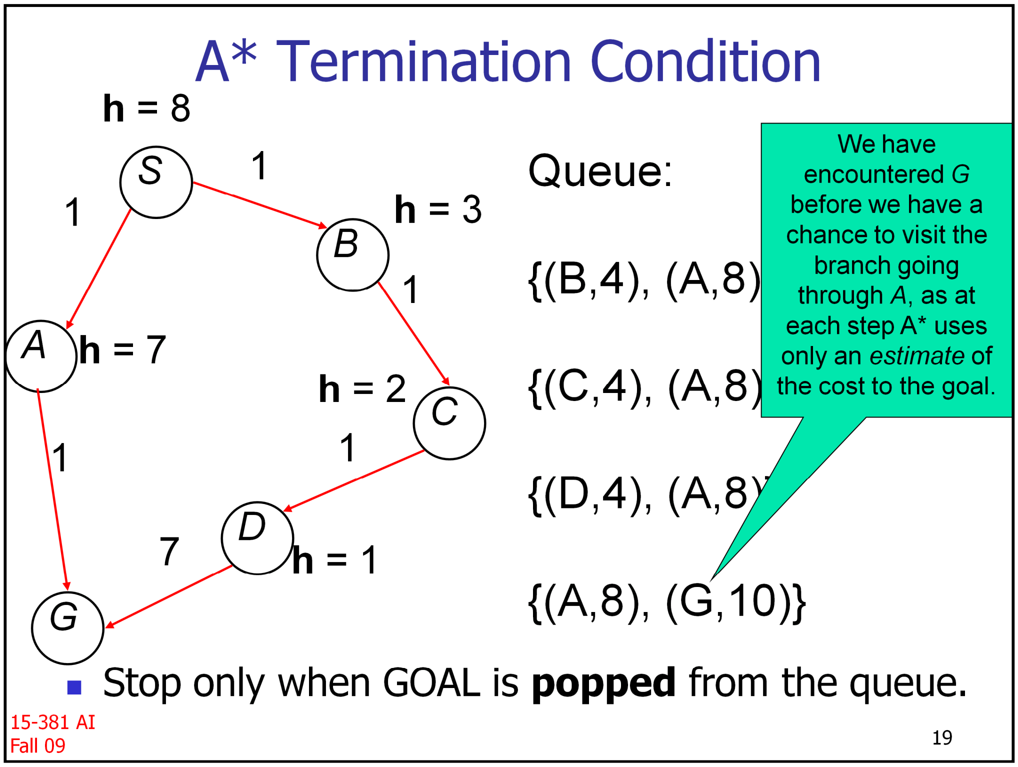
In implementation you have to keep track of both g and h throughout

Be disciplined running A* and don't take intuitive shortcuts. Manuela is "a master" at coming up with graphs designed to trick you. And she's been teaching this since 1992. That's, like, a really long time!

A* Core Issues

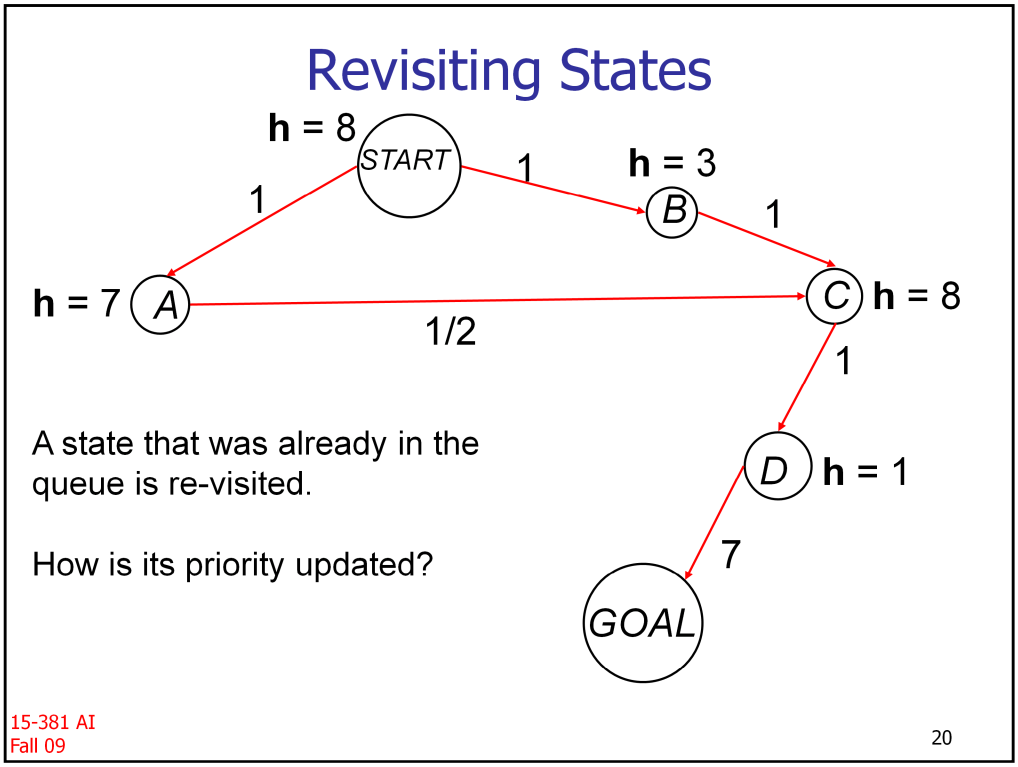
- Termination condition
- Revisiting states
- Algorithm
- Optimality
- Avoiding revisiting states
- Choosing good heuristics
- Reducing memory usage

These are what's really important.



If we stopped as soon as we see G, we'd lose. Would we stop if we had (G,8)? Yes. And you wouldn't find the optimal path because your heuristic sucks. Same if h(A)=20. You'd pop (G,10) and you'd stop and lose. However, Manuela will blame the bad heuristic for overestimating the cost to the goal from A.

Bad is well-defined too, not just slang.



Unlike in GBFS, you have to revisit states.

(Start, 8)

(B, 4), (A, 8)

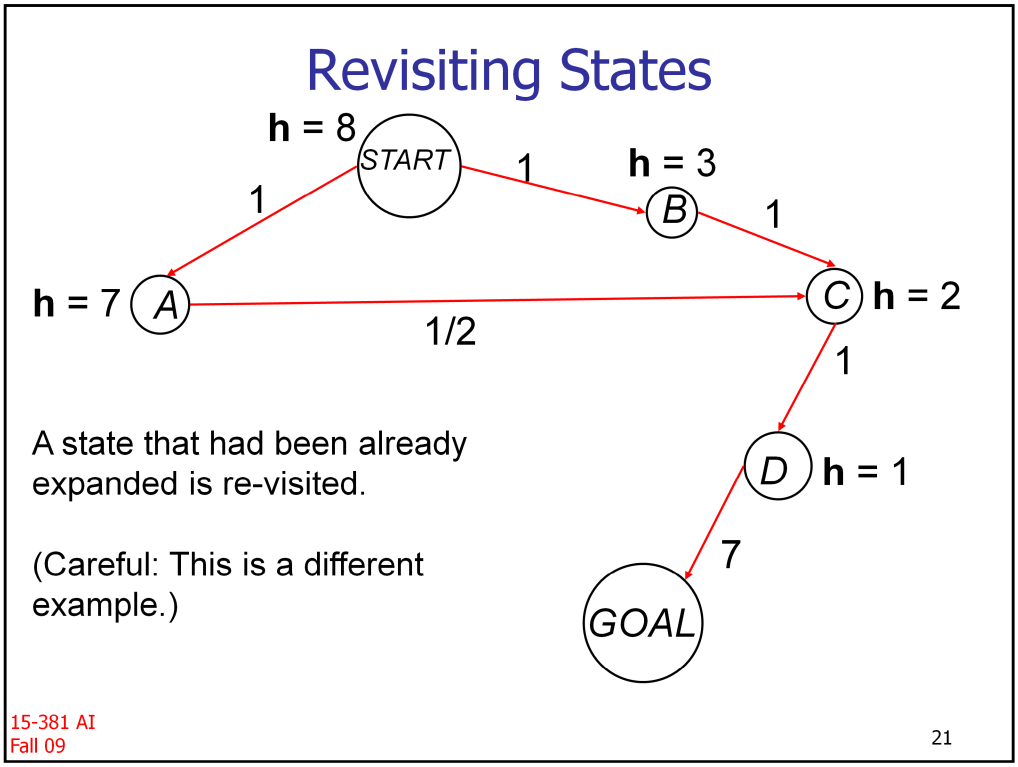
(A, 8), (C, 10)

(C, 9.5)   (C has been updated!)

(D, 3.5)

(Goal, 9.5)

Done! (Popped the goal)



Two cases:

1) Revisited state s is still in PQ. If new g(s) is smaller than old, update it.

2) Revisited state s has already been expanded. If new g(s) is smaller than old, re-insert it

(Start, 8)

(B, 4), (A, 8)

(C, 4), (A, 8)

(D, 4), (A, 8)

(A, 8), (Goal, 10)

(C, 3.5), (Goal, 10) (Re-inserted C into PQ!)

(D, 3.5), (Goal, 10) (Re-inserted D into PQ!)

(Goal, 9.5) (Updated Goal)

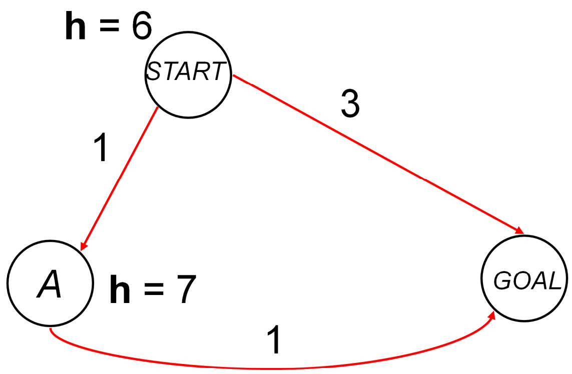
Done! (popped goal)

Pay attention to these two slides, walk through it yourself.

Pop state s with lowest **f**(s) in queue
If s = *GOAL*
  return *SUCCESS*
Else expand s:
  For all s' in **succs** (s):
    f' = **g**(s') + **h**(s') = **g**(s) + **cost**(s,s') + **h**(s')
    If (s′ not seen before OR
      s′ previously expanded with **f**(s') > f' OR
      s' in PQ with with **f**(s') > f')
        Promote/Insert s' with new value f' in *PQ*
        previous(s') ← s
    Else
        Ignore s' (because it has been visited and its current path cost **f**(s') is still the lowest path cost from *START* to s')

A* Algorithm (inside loop)

Under what Conditions is A* Optimal?



{(*START*,6)}
{(*GOAL*,3) (A,8)}

Final path:
{*START*, *GOAL*}
with cost = 3

- Problem: **h**(.) is a *poor* estimate of path cost to the goal state

Why doesn't it find the optimal? H is *overestimating*.

A good heuristic does not overestimate.

Let h* be actual cost from n to goal.

h(n) <= h*(n)

We've done one uninformed search that was always optimal (BFS).

What was the heuristic for BFS? h(s)=c, where c is some constant. Therefore f(s)=g(s)+h(s)=g(s)+c.

Admissible Heuristics

- Define **h***(s) = the true minimal cost to the goal from s
- **h** is admissible if

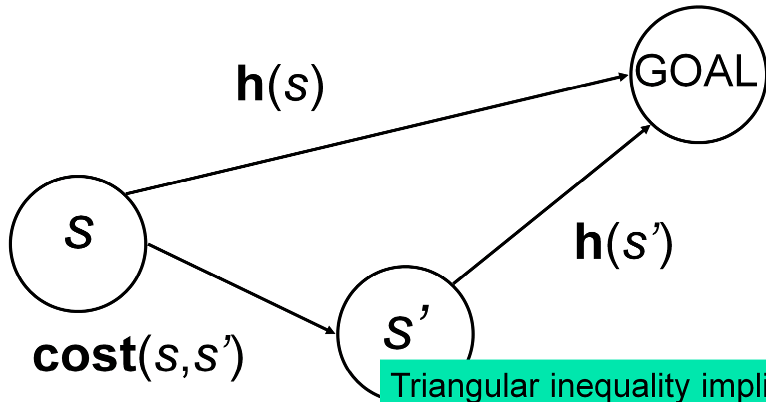
h(s) <= **h***(s) for **all** states s

- I.e., an admissible heuristic never overestimates the cost to the goal. "Optimistic" estimate of cost to goal.

A* is guaranteed to find the optimal path if **h** is admissible.

To be continued next lecture! Stay tuned, true believers!

Consistent (Monotonic) Heuristics



Triangular inequality implies that path cost always increases + need to expand node only once

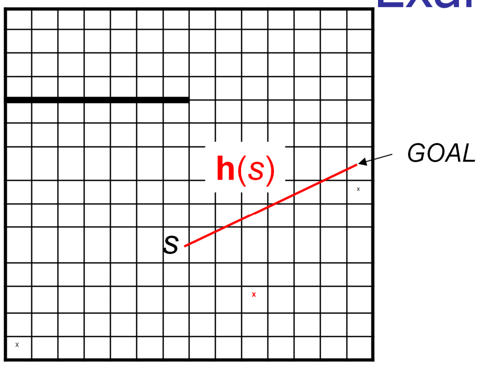
$\mathbf{h}(s) <= \mathbf{h}(s') + \mathbf{cost}(s,s'), h(G)=0$

f values are monotonically nondecreasing, f(s')>=f(s)

Pop state s with lowest **f**(s) in queue
If s = *GOAL*
  return *SUCCESS*
Else expand s:
  For all s' in **succs** (s):
    f' = **g**(s') + **h**(s') = **g**(s) + **cost**(s,s') + **h**(s')
    If (s′ not seen before OR
      ~~s′ previously expanded with **f**(s') > f' OR~~
      s' in PQ with with **f**(s') > f')
        Promote/Insert s' with new value f' in *PQ*
        previous(s') ← s
    Else
        Ignore s' (because it has been visited and its current path cost **f**(s') is still the lowest path cost from *START* to s')

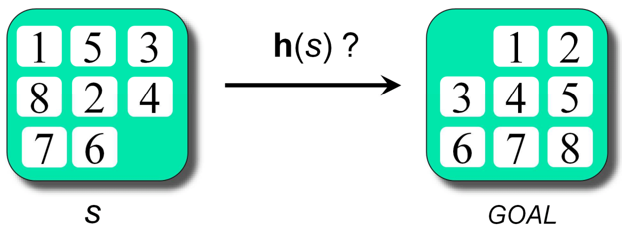
If **h** is consistent

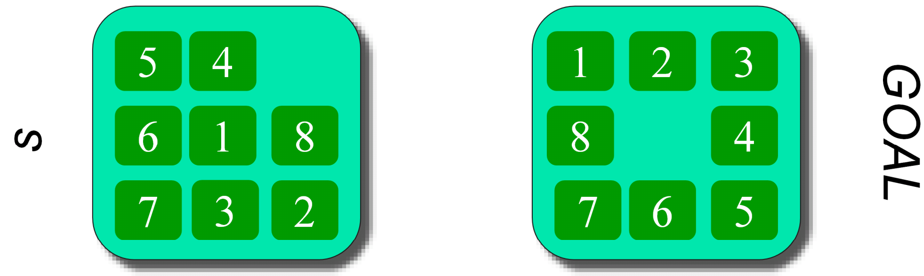
Examples



For the navigation problem: The length of the shortest path is at least the distance between *s* and *GOAL* → Euclidean distance is an admissible heuristic

What about the puzzle?





Misplaced titles:

h$_1$*(s)* = 7

Manhattan distance:

h$_2$*(s)* = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18

Are these heuristics admissible?

They both underestimate the number of moves you need to solve!

Comparing Heuristics – States expanded

h_1 = misplaced tiles

h_2 = Manhattan distance

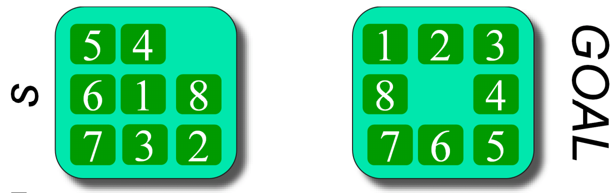
	L = 4 steps	L = 8 steps	L = 12 steps
Iterative Deepening	112	6,384	364,404
A* with heuristic h_1	13	39	227
A* with heuristic h_2	12	25	73

- Data is averaged over 100 instances of the 8-puzzle for various solution lenghts.

Example from Russell&Norvig

Using a heuristic you only have to look at far, far fewer nodes!

Comparing Heuristics



$\mathbf{h}_1(s) = 7$
$\mathbf{h}_2(s) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$
$\mathbf{h}_2$ is larger than $\mathbf{h}_1$ and, at same time, A* seems to be more efficient with $\mathbf{h}_2$.

$\mathbf{h}_2$ *dominates* $\mathbf{h}_1$, if $\mathbf{h}_2(s) >= \mathbf{h}_1(s)$ for all *s*

For any two heuristics $\mathbf{h}_2$ and $\mathbf{h}_1$:
If $\mathbf{h}_2$ dominates $\mathbf{h}_1$ then A* is more efficient (expands fewer states) with $\mathbf{h}_2$

Intuition: since **h** <= **h***, a larger **h** is a better approximation of the true path cost

Manhattan distance h_2 (s) is a tighter lower bound, so it works better.

Domination is always good if the dominating heuristic is admissible, however domination does NOT imply admissibility.

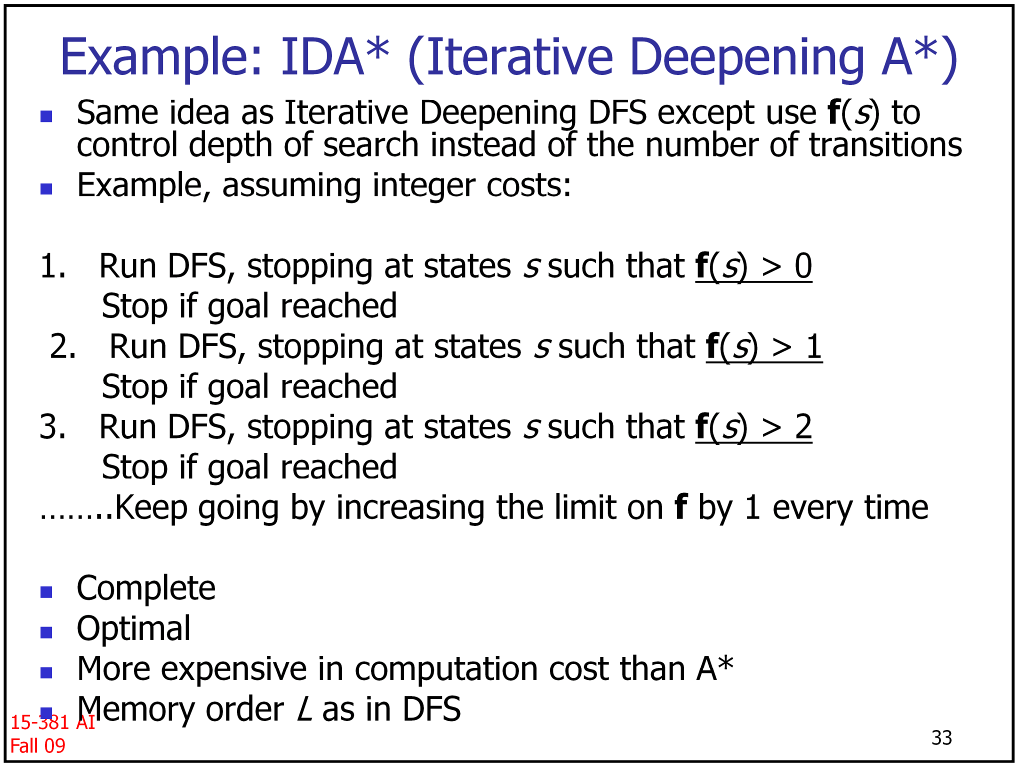
Limitations

- Computation: In the worst case, we may have to explore all the states
- The good news: A* is optimally efficient → For a given **h**(.), no other optimal algorithm will expand fewer nodes

IDS (Iterative Deepening Search)

- Need to make DFS optimal

- IDS (Iterative Deepening Search):
  - Run DFS by searching only path of length 1 (DFS stops if length of path is greater than 1)
  - If that doesn't find a solution, try again by running DFS on paths of length 2 or less
  - If that doesn't find a solution, try again by running DFS on paths of length 3 or less
  - ………..
  - Continue until a solution is found



For IDS you iterate depth.

For IDA* you iterate f(s)

Summary

- Informed search and heuristics
- Best-First Greedy search
- A* algorithm
  - Admissible heuristics, optimality
  - Condition on heuristic functions
  - Completeness, efficiency
- IDA*

Nils Nilsson. Problem Solving Methods in Artificial Intelligence. McGraw Hill (1971)
Judea Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving (1984)
Chapters 3&4 Russell & Norvig

Proof of A* Optimality with Admissible h

- By contradiction – assume that a suboptimal goal state, G' is returned.
- Let G be a goal state with optimal path cost f* and let n be a node in the path to G. h admissible, therefore f* >= f(n)
- If n is missed for expansion and instead G' is chosen, then f(n) >= f(G')
- So f* >= f(G'), and f* >= g(G') + h(G'), and f* >= g(G'), which contradicts the assumption that G' is suboptimal.

You don't need to know this proof…just here in case you're interested.