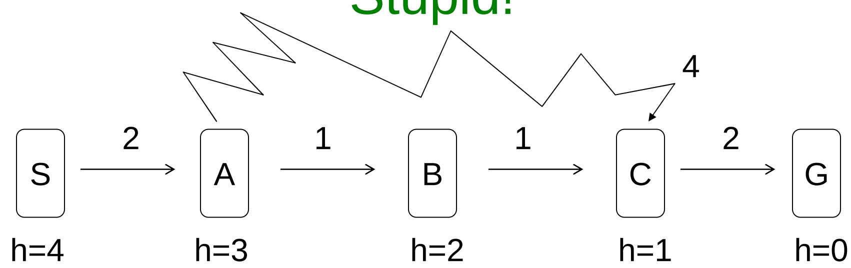
A* Heuristic Search

Andrew W. Moore
Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

Overview

- The inadequacies of "Best First Greedy" heuristic search.
- Good trick: take account of your cost of getting to the current state.
- When should the search stop?
- Admissible heuristics
- A* search is complete
- A* search will always terminate
- A*'s dark secret
- Saving masses of memory with IDA* (Iterative Deepening A*)

Let’s Make “Best first Greedy” Look Stupid!



- Best –first greedy is clearly not guaranteed to find optimal
- Obvious question: What can we do to avoid the stupid mistake?

Slide 3

A* - The Basic Idea

- Best-first greedy: When you expand a node *n*, take each successor *n'* and place it on PriQueue with priority $h(n')$
- A*: When you expand a node *n*, take each successor *n'* and place it on PriQueue with priority

$$\text{(Cost of getting to } n') + h(n') \qquad (1)$$

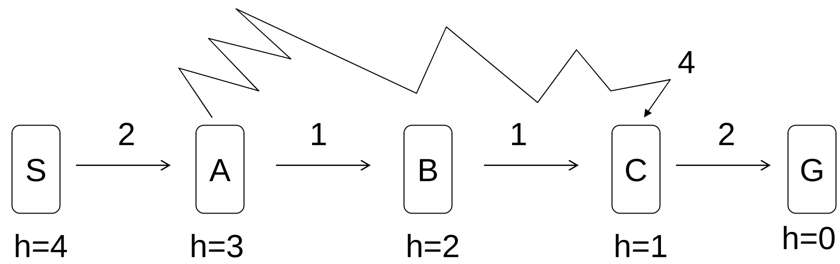
$$\text{Let } g(n) = \text{Cost of getting to } n \qquad (2)$$

and then define…

$$f(n) = g(n) + h(n) \qquad (3)$$

Slide 4

A* Looking Non-Stupid

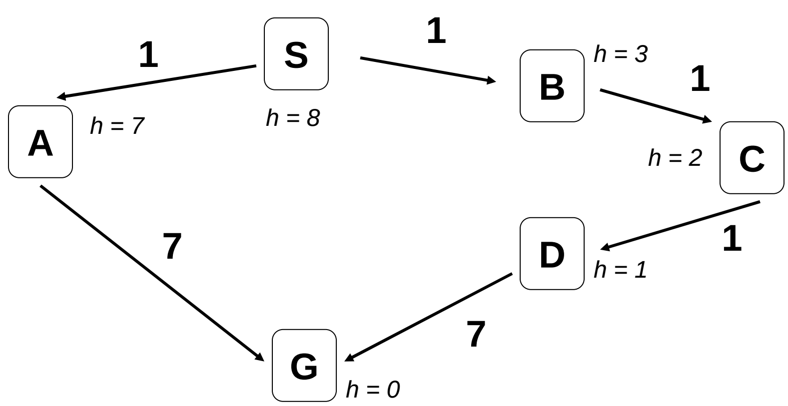


Slide 5

When should A* terminate?

Idea: As soon as it generates a goal state?

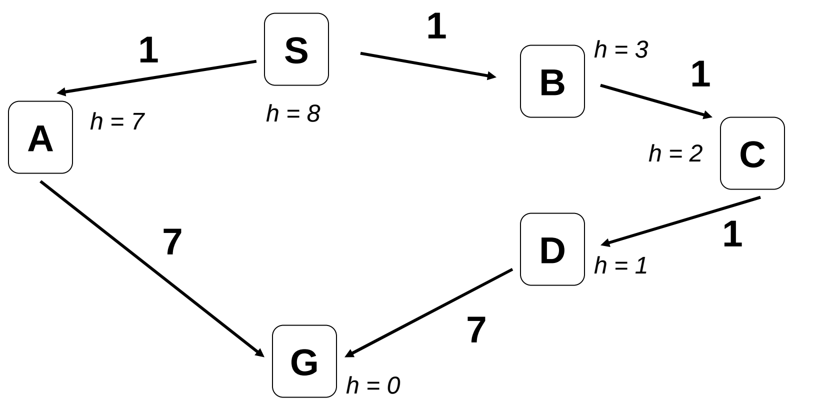
Look at this example:



Slide 6

Correct A* termination rule:

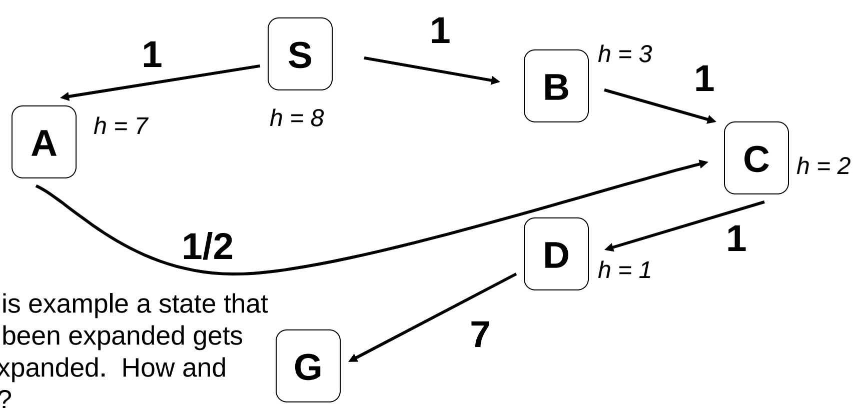
A* Terminates Only When a Goal State Is Popped from the Priority Queue



Slide 7

A* revisiting states

Another question: What if A* revisits a state that was already expanded, and discovers a shorter path?

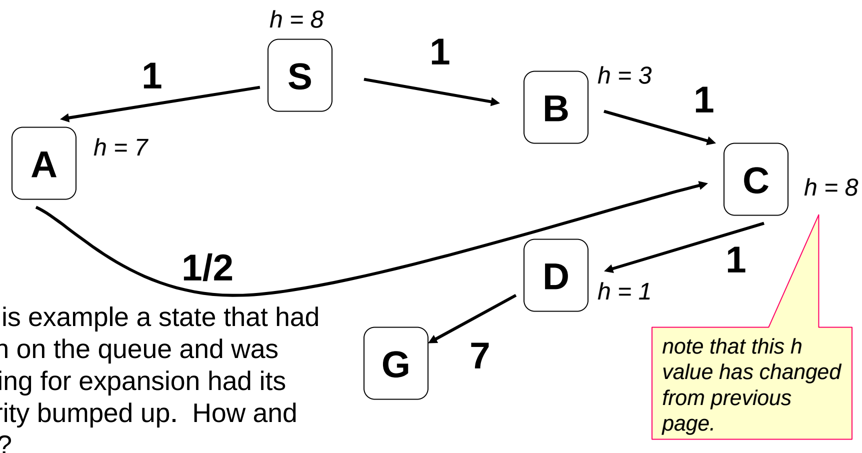


In this example a state that had been expanded gets re-expanded. How and why?

Slide 8

A* revisiting states

What if A* visits a state that is already on the queue?



In this example a state that had been on the queue and was waiting for expansion had its priority bumped up. How and why?

Slide 9

The A* Algorithm

Reminder: $g(n)$ is cost of shortest known path to n

Reminder: $h(n)$ is a heuristic estimate of cost to a goal from n

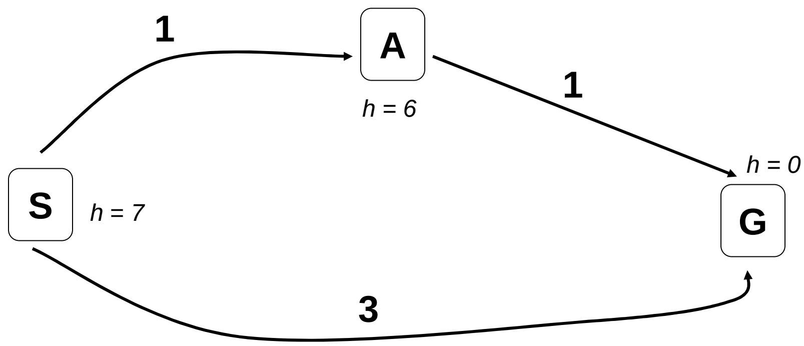
- Priority queue *PQ* begins empty.
- *V* (= set of previously visited (*state*,*f*,*backpointer*)-triples) begins empty.
- Put *S* into *PQ* and *V* with priority $f(s) = g(s) + h(s)$

  = $h(s)$ because $g(\text{start}) = 0$
- Is *PQ* empty?
  - ➢ Yes? Sadly admit there's no solution
  - ➢ No? Remove node with lowest $f(n)$ from queue. Call it *n*.
  - ➢ If *n* is a goal, stop and report success.
  - ➢ "expand" *n* : For each n' in **successors**(n)….

    use sneaky trick to compute $g(n)$
    - Let $f' = g(n') + h(n') = g(n) + cost(n,n') + h(n')$
    - **If** *n'* not seen before, or *n'* previously expanded with $f(n')>f'$, or *n'* currently in *PQ* with $f(n')>f'$
    - **Then** Place/promote *n'* on priority queue with priority *f'* and update *V* to include (*state=n'*, *f '*, *BackPtr=n*).
    - **Else** Ignore *n'*

Slide 10

Is A* Guaranteed to Find the Optimal Path?



Nope. And this example shows why not.

Slide 11

Admissible Heuristics

- Write $h^*(n)$ = the true minimal cost to goal from n.
- A heuristic h is admissible if
  $h(n) <= h^*(n)$ for all states n.
- An admissible heuristic is guaranteed never to overestimate cost to goal.
- An admissible heuristic is optimistic.

Slide 12

8-Puzzle Example

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

Which of the following are admissible heuristics?

- $h(n)$ = Number of tiles in wrong position in state n
- $h(n) = 0$
- $h(n)$ = Sum of Manhattan distances between each tile and its goal location
- $h(n) = 1$

- $h(n) = \min(2, h^*[n])$
- $h(n) = h^*(n)$
- $h(n) = \max(2, h^*[n])$

Slide 13

A* with Admissible Heuristic Guarantees Optimal Path

- Simple proof
- Your lecturer will attempt to give it from memory.
- He might even get it right. But don't hold your breath.

Slide 14

Is A* Guaranteed to Terminate?

i.e. is it complete?

- There are finitely many acyclic paths in the search tree.
- A* only ever considers acyclic paths.
- On each iteration of A* a new acyclic path is generated because:
  - When a node is added the first time, a new path exists.
  - When a node is "promoted", a new path to that node exists. It must be new because it's shorter.
- So the very most work it could do is to look at every acyclic path in the graph.
- So, it terminates.

Slide 15

Comparing Iterative Deepening with A*

From Russell and Norvig, Page 107, Fig 4.8

	For 8-puzzle, average number of states expanded over 100 randomly chosen problems in which optimal path is length…		
	…4 steps	…8 steps	…12 steps
Iterative Deepening (see previous slides)	112	6,300	3.6×10^6
A* search using "number of misplaced tiles" as the heuristic	13	39	227
A* using "Sum of Manhattan distances" as the heuristic	12	25	73

Andrew's editorial comments

1. At first sight might look like even "number of misplaced tiles" is a great heuristic. But probably h(state)=0 would also do much much better than ID, so the difference is mainly to do with ID's big problem of expanding the same state many times, not the use of a heuristic.
2. Judging solely by "number of states expanded" does not account for overhead of maintaining hash tables and priority queue for A*, though it's pretty clear here that this won't dramatically change the results.

Indeed there are only a couple hundred thousand states for the entire eight puzzle

…ig 4.8

	…ates …ndomly …ich optimal		
	…	…8 steps	…12 steps
Iterative Deepening (see previous slides)	112	6,300	3.6×10^6
A* search using "number of misplaced tiles" as the heuristic	13	39	227
A* using "Sum of Manhattan distances" as the heuristic	12	25	73

A* : The Dark Side

- A* can use lots of memory. In principle:

  O(number of states)

- For really big search spaces, A* will run out of memory.

Slide 18

IDA* : Memory Bounded Search

- Iterative deepening A*. Actually, pretty different from A*. Assume costs integer.
  1. Do loop-avoiding DFS, not expanding any node with $f(n) > 0$. Did we find a goal? If so, stop.
  2. Do loop-avoiding DFS, not expanding any node with $f(n) > 1$. Did we find a goal? If so, stop.
  3. Do loop-avoiding DFS, not expanding any node with $f(n) > 2$. Did we find a goal? If so, stop.
  4. Do loop-avoiding DFS, not expanding any node with $f(n) > 3$. Did we find a goal? If so, stop.

  …keep doing this, increasing the $f(n)$ threshold by 1 each time, until we stop.
- This is
  - Complete
  - Guaranteed to find optimal
  - More costly than A* in general.

Slide 19

What You Should Know

- Thoroughly understand A*.
- Be able to trace simple examples of A* execution.
- Understand “admissibility” of heuristics. Proof of completeness, guaranteed optimality of path.
- Be able to criticize best first search.

References:

Nils Nilsson. *Problem Solving Methods in Artificial Intelligence*. McGraw Hill (1971) E&S-BK 501-5353 N71p.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley (1984) E&S-BK 501-535 P35h.

Chapters 3 & 4 of Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach.

Slide 20

Proof: A* with Admissible Heuristic Guarantees Optimal Path

- Suppose it finds a suboptimal path, ending in goal state G_1 where $f(G_1) > f^*$ where $f^* = h^*(start)$ = cost of optimal path.
- There must exist a node n which is
  - Unexpanded
  - The path from start to n (stored in the BackPointers(n) values) is the start of a true optimal path

Why must such a node exist? Consider any optimal path *s*,*n1*,*n2*…goal. If all along it were expanded, the goal would've been reached along the shortest path.

- $f(n) >= f(G_1)$ (else search wouldn't have ended)
- Also $f(n) = g(n) + h(n)$

$= g^*(n) + h(n)$ because it's on optimal path

$<= g^*(n) + h^*(n)$ By the admissibility assumption

$= f^*$ Because n is on the optimal path

So $f^* >= f(n) >= f(G_1)$ contradicting top of slide

Exercise Part 1

In the following maze the successors of a cell include any cell directly to the east, south, west or north of the current cell except that no transition may pass through the central barrier. for example *successors*(*m*) = { *d* , *n* , *g* }.

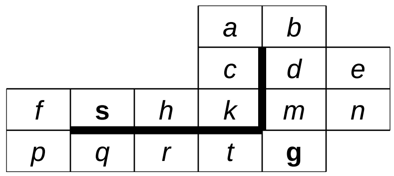
			a	*b*	
			c	*d*	*e*
f	**s**	*h*	*k*	*m*	*n*
p	*q*	*r*	*t*	**g**	

The search problem is to find a path from **s** to **g**. We are going to examine the order in which cells are expanded by various search algorithms. for example, one possible expansion order that breadth first search might use is:

s *h f k p c q a r b t d* **g**

There are other possible orders depending on which of two equal-distance-from-start states happen to be expanded first. For example **s** *f h p k c q r a t b* **g** is another possible answer. *continued->*

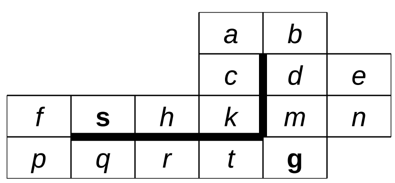
Exercise Part 1 continued



Assume you run **depth-first-search** until it expands the goal node. Assume that you always try to expand East first, then South, then West, then North. Assume your version of depth first search avoids loops: it never expands a state on the current path. What is the order of state expansion?

Slide 23

Exercise Part 2



Next, you decide to use a Manhattan Distance Metric heuristic function

h(*state*) = shortest number of steps from *state* to **g** if there were no barriers

So, for example, *h*(*k*) = 2, *h*(**s**) = 4, *h*(**g**) = 0

Assume you now use best-first greedy search using heuristic *h* (a version that never re-explores the same state twice). Again, give all the states expanded, in the order they are expanded, until the algorithm expands the goal node.

Finally, assume you use A* search with heuristic *h*, and run it until it terminates using the conventional A* termination rule. Again, give all the states expanded, in the order they are expanded. (Note that depending on the method that A* uses to break ties, more than one correct answer is possible).

Slide 24

Another Example Question

Consider the use of the A* algorithm on a search graph with cycles, and assume that this graph does not have negative-length edges. Suppose you are explaining this algorithm to Pat, who is not familiar with AI. After your elaborated explanation of how A* handles cycles, Pat is convinced that A* does a lot of unnecessary work to guarantee that it works properly (i.e. finds the optimal solution) in graphs containing cycles. Pat suggests the following modification to improve the efficiency of the algorithm:

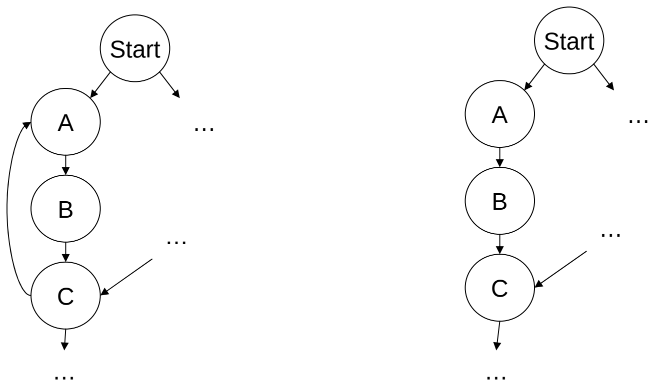
> Since the graph has cycles, you may detect new cycles from time to time when expanding a node. For example, if you expand nodes A, B, and C shown on figure (a) on the next slide, then after expanding C and noticing that A is also a successor of C, you will detect the cycle A-B-C-A. Every time you notice a cycle, you may remove the last edge of this cycle from the search graph. For example, after expanding C, you can remove the edge C-A (see figure (b) on next slide). Then, if A* visits node C again in the process of further search, it will not need to traverse this useless edge the second time.

continued next slide

Slide 25

more Another Example Question

Does this modified version of A* always find the optimal path to a solution? Why or why not?



(a) Detecting a Cycle

(b) Removing the detected cycle

Slide 26