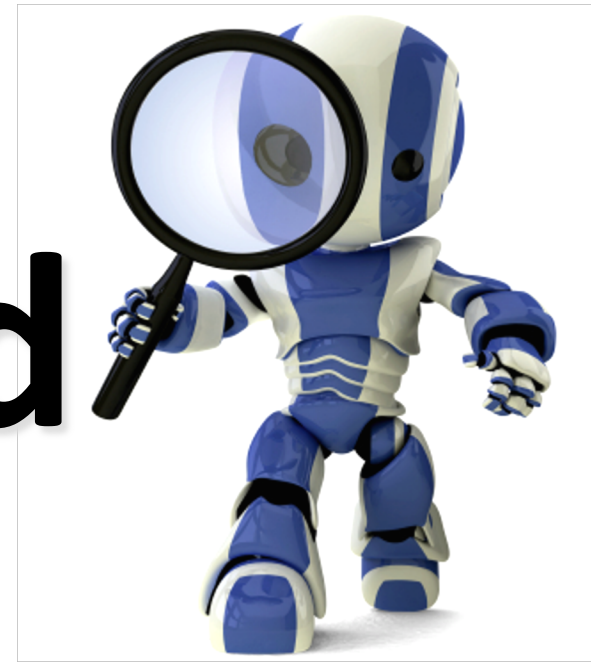


# Informed Search Chapter 4 (a)



Some material adopted from notes  
by Charles R. Dyer, University of  
Wisconsin-Madison

# Today's class

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - Algorithms  $A$  and  $A^*$
  - Examples
- Memory-conserving variations of  $A^*$
- Heuristic functions

# Big idea: heuristic



## Merriam-Webster's Online Dictionary

Heuristic (pron. \hyu- 'ris-tik\): adj. [from Greek *heuriskein* to discover] involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially **trial-and-error methods**

## Free On-line Dictionary of Computing

heuristic 1. <programming> A **rule of thumb**, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Unlike algorithms, heuristics do not guarantee feasible solutions and are often used with no theoretical guarantee. 2. <algorithm> **approximation algorithm**.

## WordNet

heuristic adj 1: (CS) relating to or using a heuristic rule 2: of or relating to a general formulation that serves to guide investigation [ant: algorithmic] n : a **commonsense rule** (or set of rules) intended to increase the probability of solving some problem [syn: heuristic rule, heuristic program]

# Informed methods add domain-specific information

- Select most promising path along which to continue searching
- $h(n)$ : estimates *goodness* of node  $n$
- $h(n)$  = **estimated cost** (or distance) of minimal cost path from  $n$  **to a goal state**.
- Estimates how close a state  $n$  is to a goal using domain-specific information

# Heuristics

- **All domain knowledge** used in search is encoded in the **heuristic function,  $h(\langle \text{node} \rangle)$**
- 8-puzzle example:
  - Number of tiles out of place
- Better 8-puzzle heuristic:
  - Sum of distances for each tile to its goal position
- In general
  - $h(n) \geq 0$  for all nodes  $n$
  - $h(n) = 0$  implies that  $n$  is a goal node
  - $h(n) = \infty$  implies  $n$  is a dead-end that can't lead to goal

# Weak vs. strong methods



- **Weak methods** are extremely general methods not tailored to a specific situation or domain, e.g.:
  - **Generate and test:** generate solution candidates and test until you find one
  - **Means-ends analysis:** represent current situation & goal, then seek ways to shrink differences between them
  - **Space splitting:** list possible solutions to a problem, then try to rule out classes of the possibilities
  - **Subgoaling:** split large problem into smaller ones that can be solved one at a time
- Called *weak* because they don't use more powerful, **domain-specific** heuristics; **strong methods** are specific to a particular problem
- **Weak methods** useful when we don't have a strong one

# Heuristics for 8-puzzle

## Misplaced Tiles Heuristic

(not including  
the blank)

Current  
State

3	2	8
4	5	6
7	1	

Goal  
State

1	2	3
4	5	6
7	8	

3	2	8
4	5	6
7	1	

3 tiles are not  
where they  
need to be

- Three tiles are misplaced (the 3, 8, and 1) so heuristic function evaluates to 3
- Heuristic says that it *thinks* a solution may be available in 3 or more moves
- Very rough estimate, but easy to calculate

**h = 3**

# Heuristics for 8-puzzle

## Manhattan Distance Heuristic

(not including the blank)

Current State

3	2	8
4	5	6
7	1	

Goal State

1	2	3
4	5	6
7	8	

3	→	<u>3</u>

2 steps

	←	8
	↓	
	<u>8</u>	

3 steps

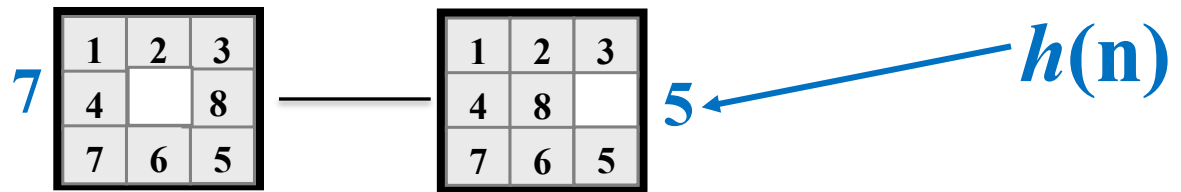
<u>1</u>	←	
	↑	
	1	

3 steps

- The 3, 8, and 1 tiles misplaced by 2, 3, and 3 steps, so heuristic function evaluates to 8
- Heuristic says that it *thinks* a solution may be available in 8 or more moves
- More accurate than the misplaced heuristic, but slightly more expensive to compute

**h = 8**

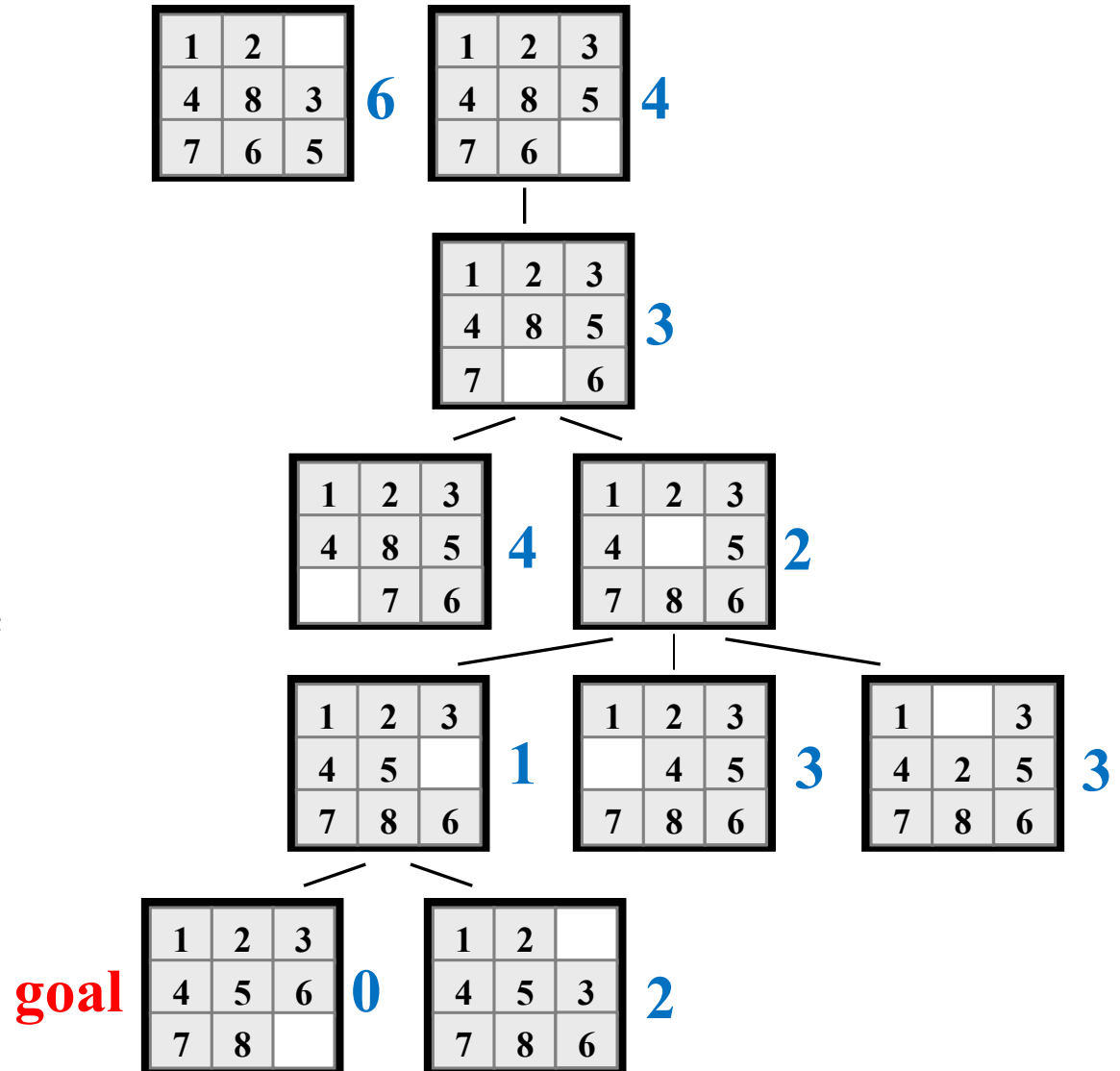




Use heuristics to guide search

In this hill climbing example, Manhattan Distance heuristic helps quickly find a solution to the puzzle

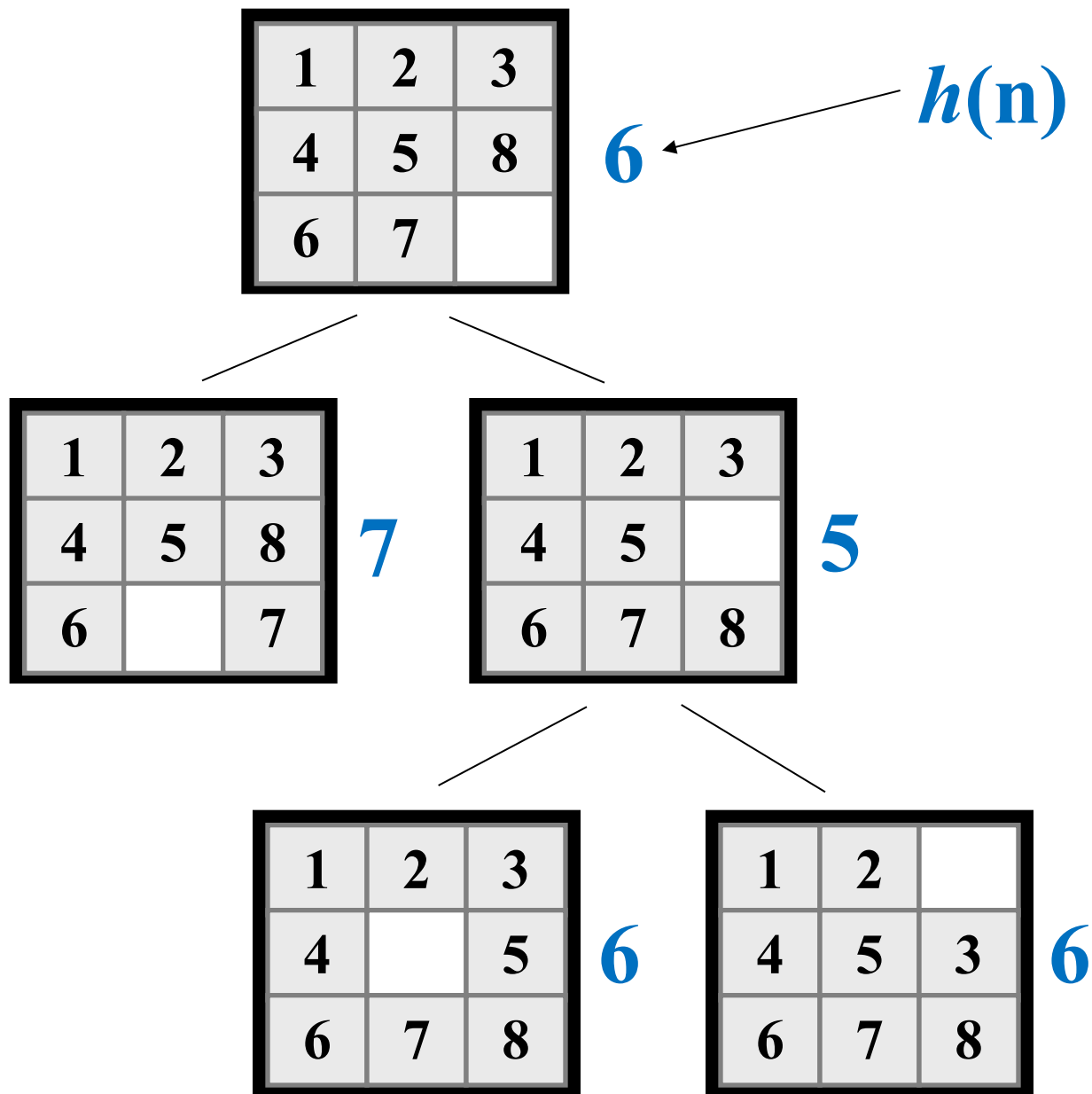
At a node, compute all possible next states, move to one with lowest value



In this example,  
**hill climbing**  
doesn't work!

All nodes on  
fringe are taking a  
step "backwards"  
(local minima)

This puzzle *is*  
solvable in just 12  
more steps

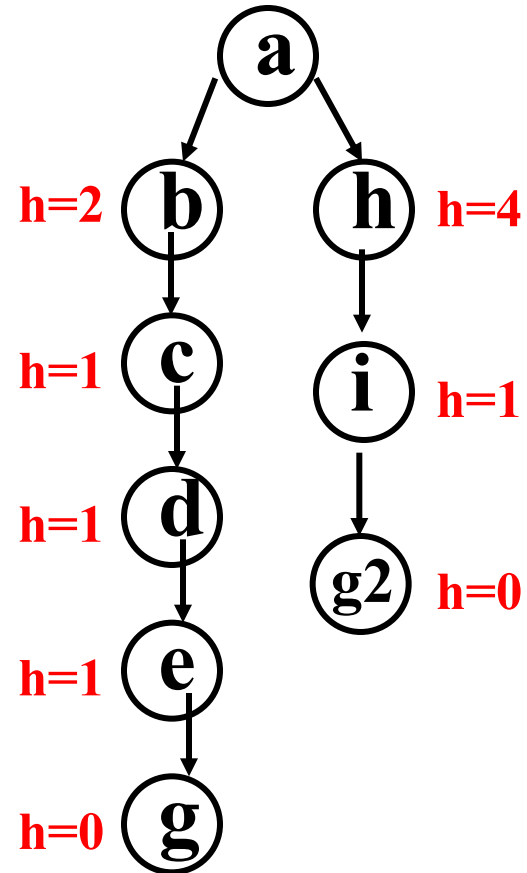


# Best-first search

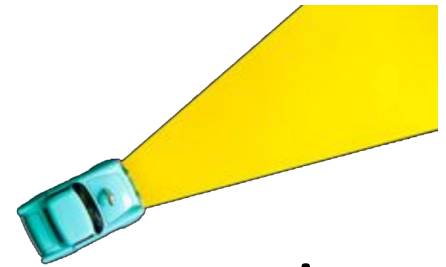
- Search algorithm that improves **depth-first search** by expanding most promising node chosen according to heuristic rule
- Order nodes on nodes list by increasing value of an evaluation function,  **$f(n)$** , incorporating domain-specific information
- $f(n) = g(n) + h(n)$  where
  - $g(n)$  = distance from start node to node  $n$
  - $h(n)$  = heuristic estimate of distance from  $n$  to a goal
- Using the  $f(n)$  concept is a generic framework for search methods

# Greedy best first search

- A [greedy algorithm](#) makes locally optimal choices in hope of finding a global optimum
- Uses evaluation function  $f(n) = h(n)$ , sorting nodes by increasing values of  $f$
- Selects node to expand appearing **closest** to goal, i.e., node with smallest  $f$  value
- Not complete (why?)
- Not [admissible](#), as in example
  - Assume arc costs = 1, greedy search finds goal  $g$ , with solution cost of 5
  - Optimal solution: path to goal with cost 3



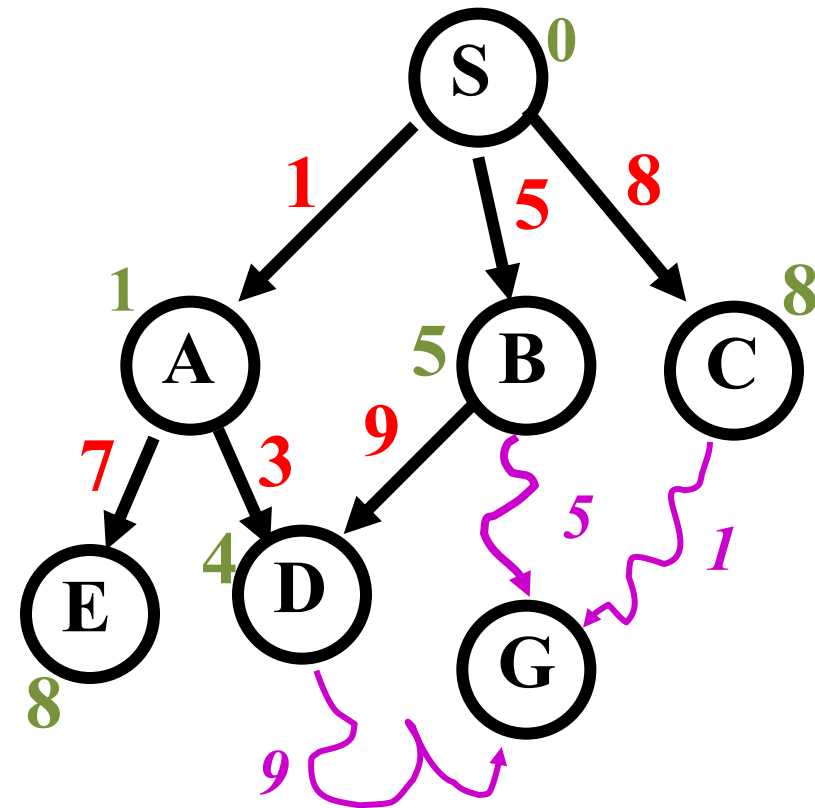
# Beam search



- Use evaluation function  $f(n)=h(n)$ , but max size of the nodes list is  $k$ , a fixed constant
- Only keeps  $k$  best nodes as candidates for expansion, discard rest
- $k$  is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As  $k$  increases, approaches best first search
- Not complete
- Not admissible (optimal)

# Algorithm A

- Use as an evaluation function
  - $f(n) = g(n) + h(n)$
- $g(n)$  = minimal-cost path from the start state to state  $n$
- $g(n)$  adds “breadth-first” term to evaluation function
- Ranks nodes on search frontier by estimated cost of solution from start node *via given node* to goal
- Not complete if  $h(n)$  can =  $\infty$
- Not admissible (optimal)



$g(d)=4$	$g(b)=5$	$g(c)=8$
$h(d)=9$	$h(b)=5$	$h(c)=1$
$f(d)=13$	$f(b)=10$	$f(c)=9$

*C is chosen next to expand*

# Algorithm A

- 1 Put start node  $S$  on the nodes list, called OPEN
- 2 If OPEN is empty, exit with failure
- 3 Select node in OPEN with minimal  $f(n)$  and place on CLOSED
- 4 If  $n$  is a goal node, collect path back to start and stop
- 5 Expand  $n$ , generating all its successors and attach to them pointers back to  $n$ . For each successor  $n'$  of  $n$ 
  - 1 If  $n'$  not already on OPEN or CLOSED
    - put  $n'$  on OPEN
    - compute  $h(n')$  then set  $g(n')=g(n)+ c(n,n')$ ;  $f(n')=g(n')+h(n')$
  - 2 If  $n'$  already on OPEN or CLOSED and if  $g(n')$  is lower for new version of  $n'$ , then:
    - Redirect pointers backward from  $n'$  on path with lower  $g(n')$
    - Put  $n'$  on OPEN

# Algorithm A\*

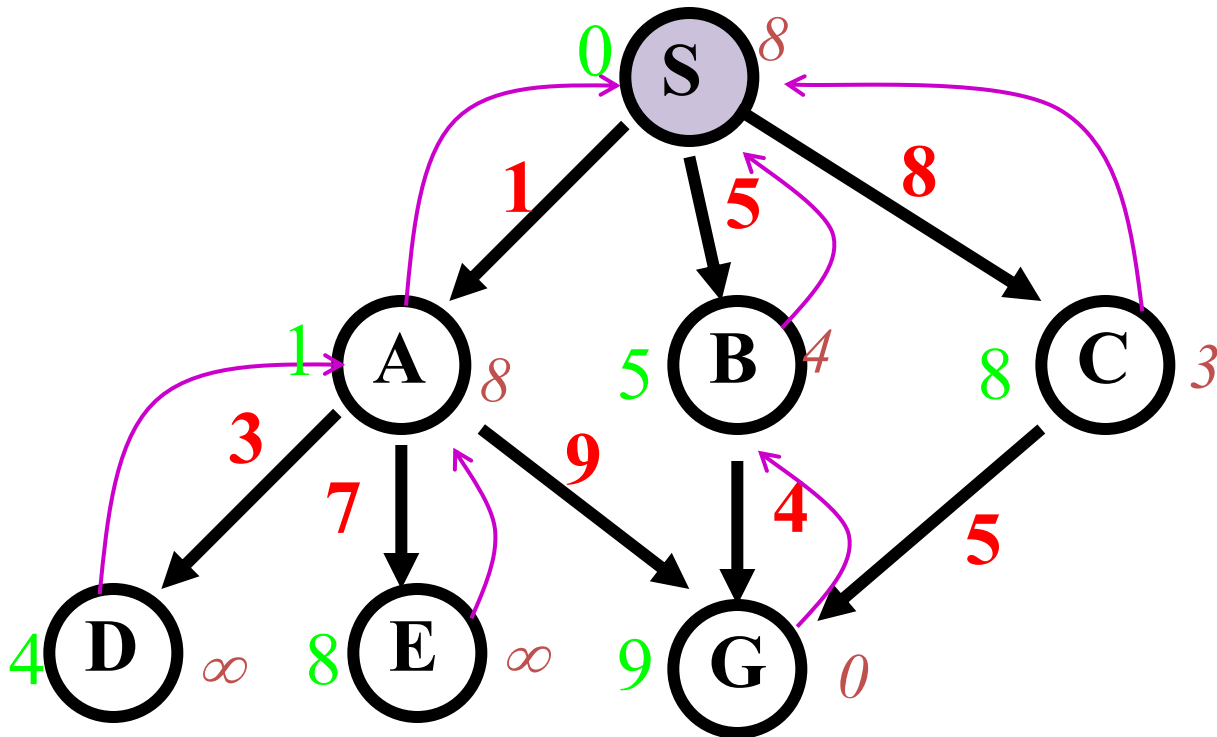
- Pronounced “*a star*”
- Algorithm A with constraint that  $h(n) \leq h^*(n)$ 
  - $h^*(n)$  = *true cost of minimal cost path* from  $n$  to goal
  - So:  $h(n)$  *never overestimates* cost to get from  $n$  to goal
- $h$  is **admissible** when  $h(n) \leq h^*(n)$  holds
- Using an admissible heuristic guarantees that 1st solution found will be an **optimal** one
- A\* is **complete** whenever branching factor is finite and every action has fixed, positive cost
- A\* is **admissible**



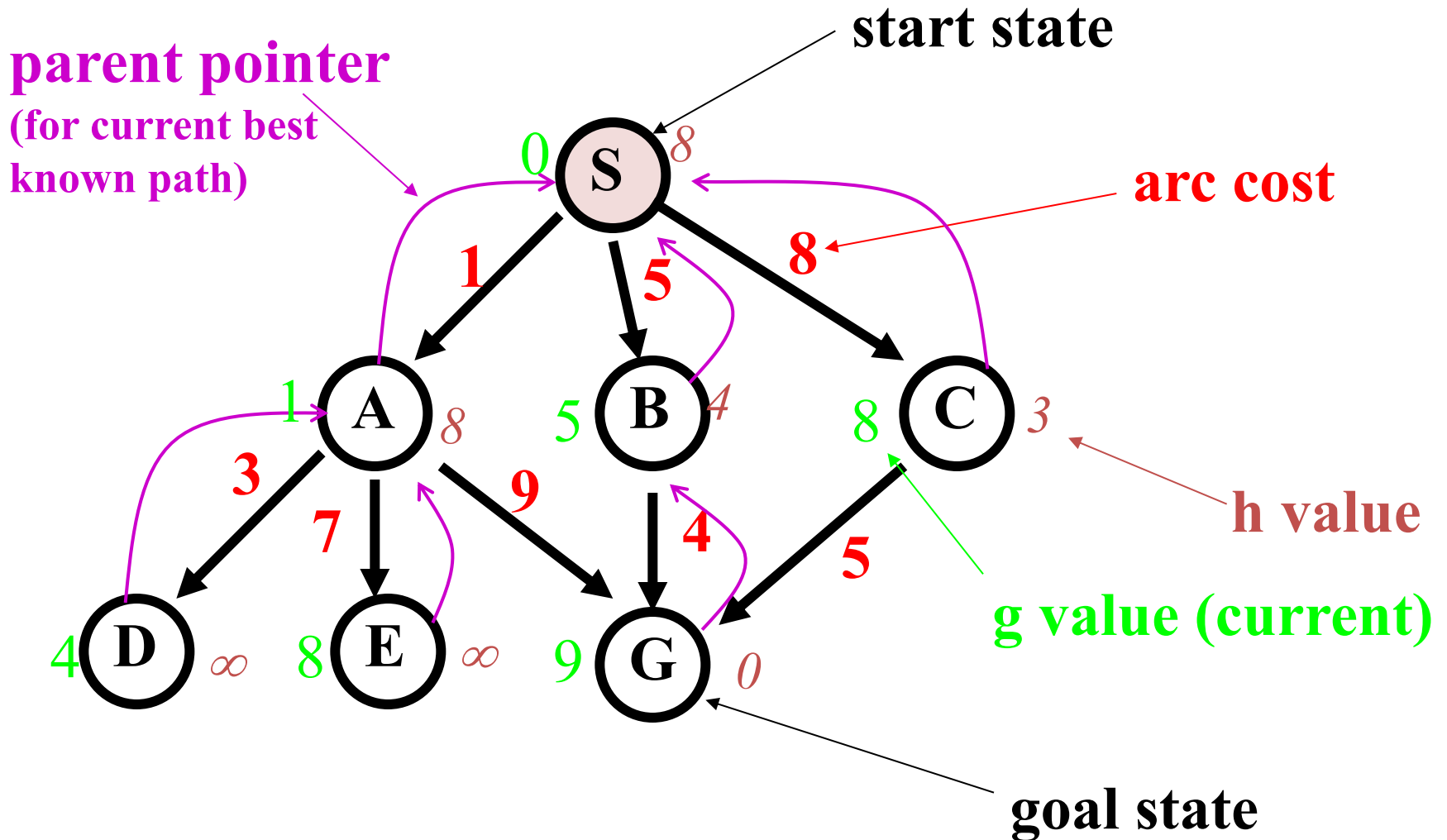
# Observations on A

- **Perfect heuristic:** If  $h(n)=h^*(n)$  for all  $n$ , only nodes on an optimal solution path expanded; no extra work done
- **Null heuristic:** If  $h(n) = 0$  for all  $n$ , then it's an admissible heuristic;  $A^*$  acts like uniform-cost search
- **Better heuristic:** If  $h_1(n) < h_2(n) \leq h^*(n)$  for all non-goal nodes, then  $h_2$  is a ***better*** heuristic than  $h_1$ 
  - If  $A_1^*$  uses  $h_1$ , and  $A_2^*$  uses  $h_2$ , then every node expanded by  $A_2^*$  is also expanded by  $A_1^*$
  - i.e.,  $A_1$  expands at least as many nodes as  $A_2^*$
  - We say that  $A_2^*$  is *better informed* than  $A_1^*$
- The closer  $h$  to  $h^*$ , the fewer extra nodes expanded

# Example search space

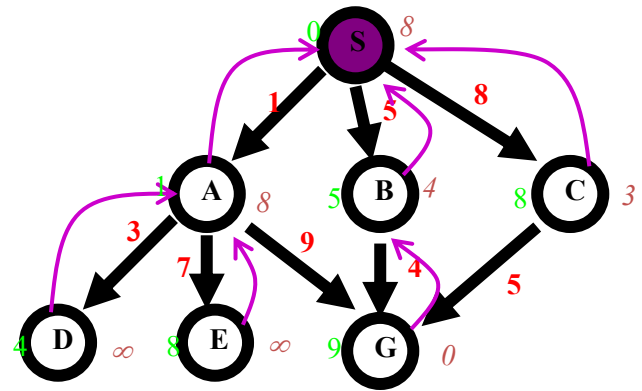


# Example search space



# Example

n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	$\infty$	$\infty$	$\infty$
E	8	$\infty$	$\infty$	$\infty$
G	9	0	9	0



- $h^*(n)$  is (hypothetical) perfect heuristic (an [oracle](#))
- Since  $h(n) \leq h^*(n)$  for all  $n$ ,  $h$  is admissible (optimal)
- Optimal path =  $S B G$  with cost 9

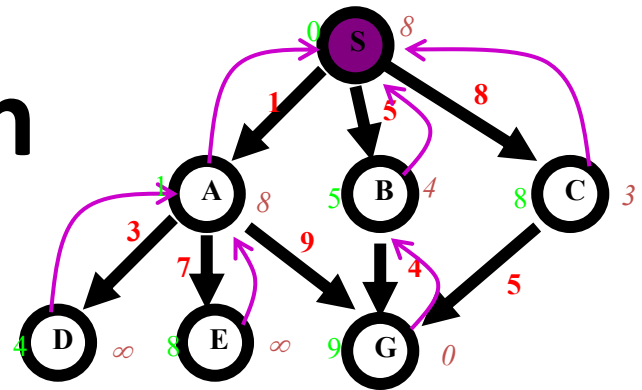
# Greedy search

$$f(n) = h(n)$$

node expanded

nodes list

node expanded	nodes list
	{S(8)}
S	{C(3) B(4) A(8)}
C	{G(0) B(4) A(8)}
G	{B(4) A(8)}



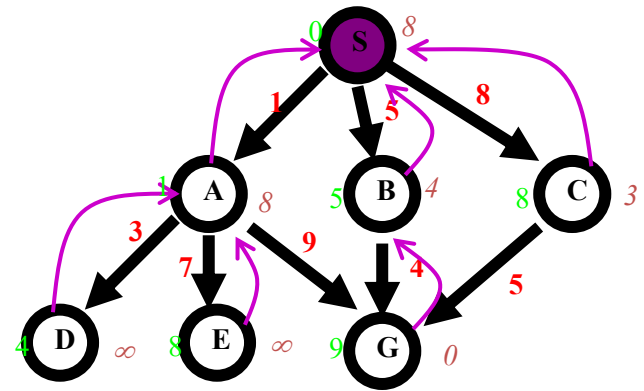
Queue of nodes on fringe ordered by  $f()$

- Pop leftmost node off queue
- If a goal, done
- Else compute its successor & update queue and graph

- Solution path found is S C G
- 3 nodes expanded
- Search was fast! But path is NOT optimal
- It didn't take into account high cost (8) to get to C
- Greedy algorithms make *locally optimal* choices at each step

# A\* search

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



expand      fringe

{S(8)}

S      {A(9) B(9) C(11)}

A      {B(9) G(10) C(11) D(inf) E(inf)}

B      {G(9) G(10) C(11) D(inf) E(inf)}

G      {C(11) D(inf) E(inf)}

h(S)=8  
h(A)=8  
h(B)=4  
h(C)=3  
h(D)=inf  
h(E)=inf  
h(G)=0

- Solution path found is S B G, 4 nodes expanded.
- Estimates total cost of path to try to find global optimum
- Still pretty fast. And optimal, too [assuming  $h(n) < h^*(n)$ ]

# Proof of the optimality of $A^*$

- Assume that  $A^*$  has selected  $G_2$ , a goal state with a suboptimal solution, i.e.,  $g(G_2) > f^*$
- Proof by contradiction shows it's impossible
  - Choose a node  $n$  on an optimal path to  $G$
  - Because  $h(n)$  is admissible,  $f^* \geq f(n)$
  - If we choose  $G_2$  instead of  $n$  for expansion, then  $f(n) \geq f(G_2)$
  - This implies  $f^* \geq f(G_2)$
  - $G_2$  is a goal state:  $h(G_2) = 0$ ,  $f(G_2) = g(G_2)$ .
  - Therefore  $f^* \geq g(G_2)$
  - Contradiction

# Dealing with hard problems

- For large problems,  $A^*$  may need too much space
- Variations conserve memory: IDA\* and SMA\*
- IDA\*, iterative deepening  $A^*$ , uses successive iteration with growing limits on  $f$ , e.g.
  - $A^*$  but don't consider a node  $n$  where  $f(n) > 10$
  - $A^*$  but don't consider a node  $n$  where  $f(n) > 20$
  - $A^*$  but don't consider a node  $n$  where  $f(n) > 30, \dots$
- SMA\* -- Simplified Memory-Bounded  $A^*$ 
  - Uses queue of restricted size to limit memory use

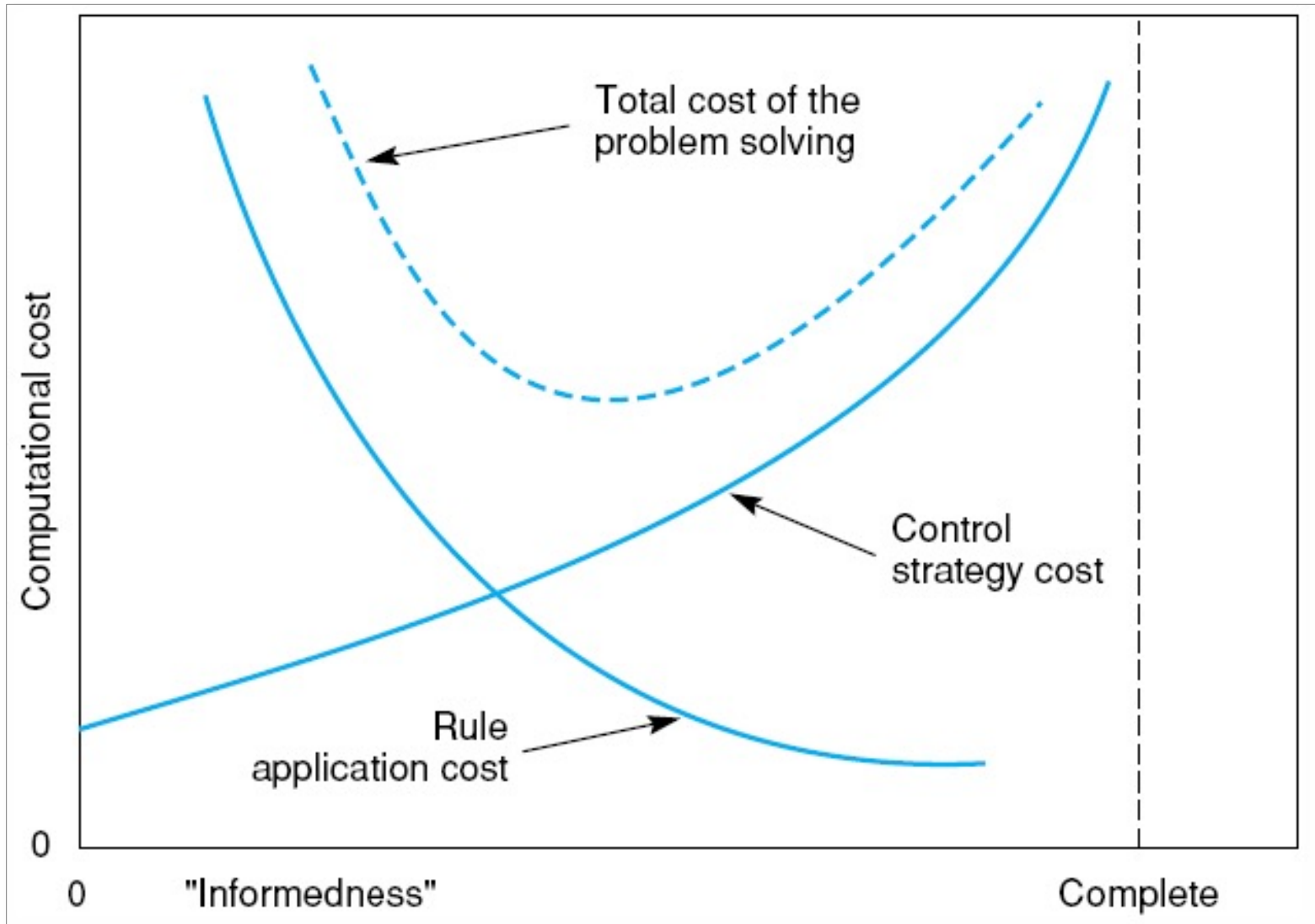


# Finding good heuristics

- If  $h_1(n) < h_2(n) \leq h^*(n)$  for all  $n$ ,  $h_2$  is better than (**dominates**)  $h_1$
- **Relaxing problem**: remove constraints for easier problem; use its solution cost as heuristic function
- Max of two admissible heuristics is a “**combining heuristic**”: admissible heuristic, and it’s better!
- Use statistical estimates to compute  $h$ ; may lose admissibility
- Identify good features, then use **machine learning** to find heuristic function; also may lose admissibility

# Use A or A\*?

- Finding a good heuristic that's always an underestimate can be hard
  - Some are intractable for real problems because they're expensive to compute or lead to large search spaces
- We may be happy with solutions that are at least close to an optimal one
- For many problems, using a fast heuristic that sometimes overestimates is a good choice
- Still, for some problems might be worth the effort to find an optimal solution



Informal plot of cost of searching and cost of computing heuristic evaluation against informedness of heuristic, Nils Nilsson, Principles of Artificial Intelligence (1980)

# What's in a Name?



- Why are these algorithms named A and A\*?
- To find out, read this short piece in CACM:

James W. Davis, Jeff Hachtel, [A\\* Search: What's in a Name?](#), Communications of the ACM, Jan. 2020, Vol. 63 No. 1, Pages 36-37

# Summary: Informed search

- **Best-first search** is general search where minimum-cost nodes (w.r.t. some measure) are expanded first
- **Greedy search** uses minimal estimated cost  $h(n)$  to goal state as measure; reduces search time, but is neither complete nor optimal
- **A\* search** combines uniform-cost search & greedy search:  $f(n) = g(n) + h(n)$ . Handles state repetitions &  $h(n)$  never overestimates
  - A\* is complete & optimal, but space complexity high
  - Time complexity depends on quality of heuristic function
  - IDA\* and SMA\* reduce the memory requirements of A\*