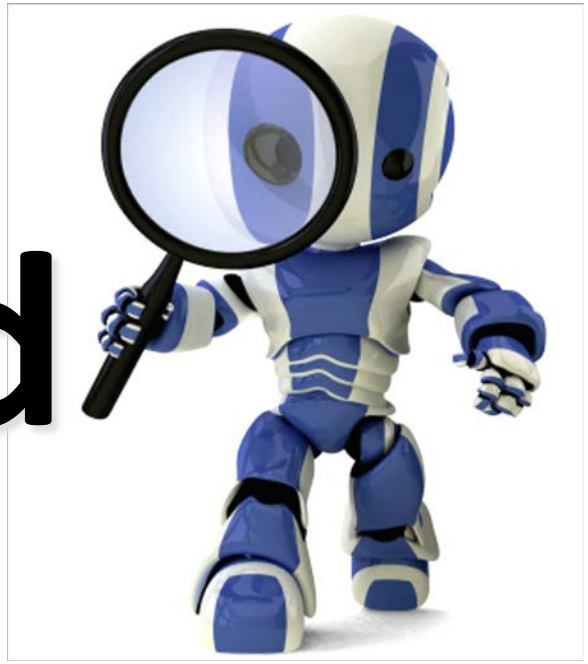
Informed Search

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. \hy*u*-ˊris-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)** = **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
  - h(n) = 0 only if n is a goal node
- **h(n)** estimates *goodness* of node n, i.e., the smaller it is the better

Heuristics

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

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

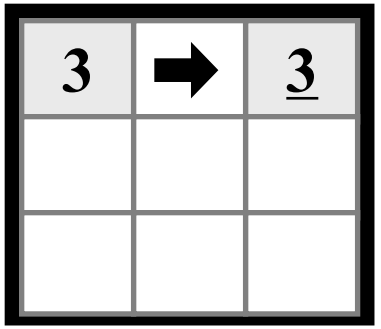
(not including the blank)

Current State

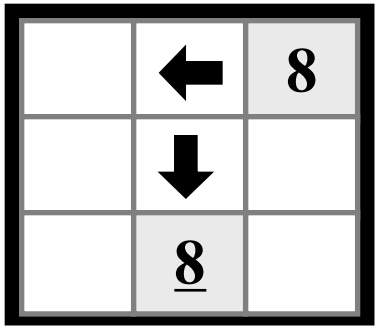
3	2	8
4	5	6
7	1	

Goal State

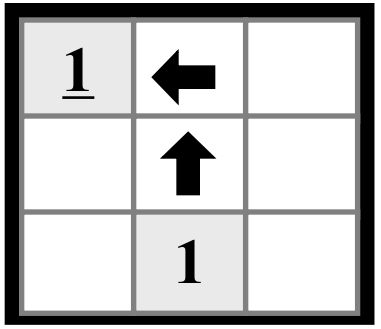
1	2	3
4	5	6
7	8	



2 steps



3 steps

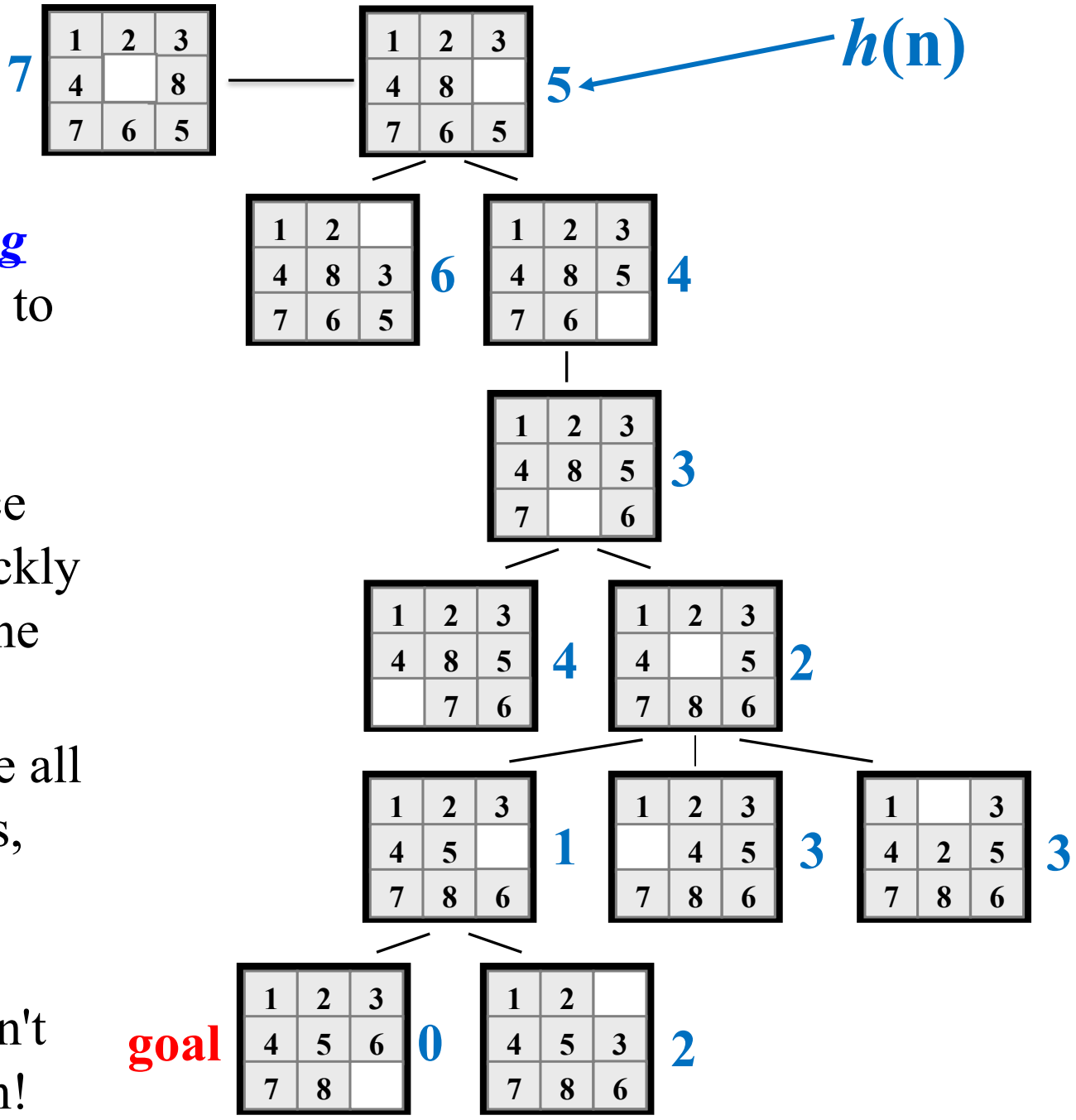


3 steps

- 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

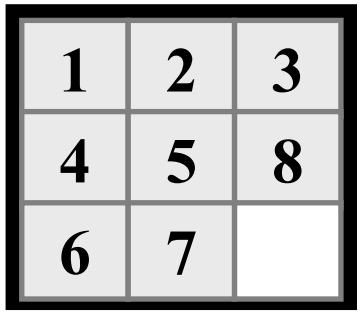
h = 8

- Use heuristics to guide search
- In this ***hill climbing*** example, we move to state with lowest **h** value
- 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
- Hill climbing doesn't need to build graph!

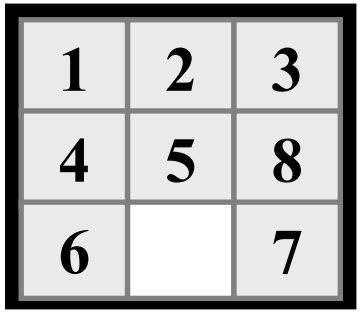


- In this example, **hill climbing** doesn't work!
- Gets stuck at a state where every move increases h(n)
- This puzzle *is* solvable in just 12 more steps
- Hill climbing is not good for most problems

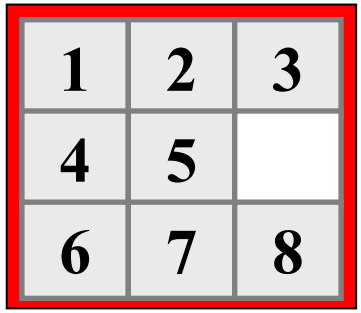
"Sometimes we need to take a step backwards to take two steps forward"



6 ← *h*(n)

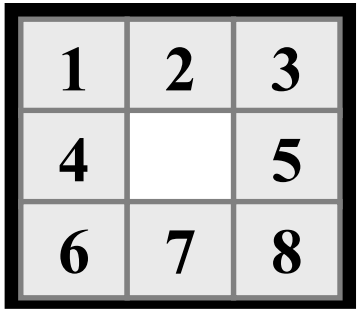


7

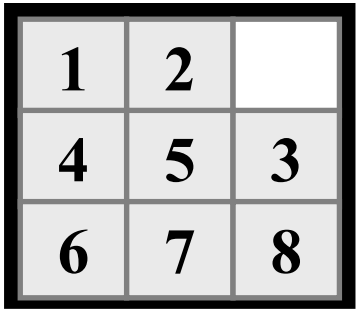


5

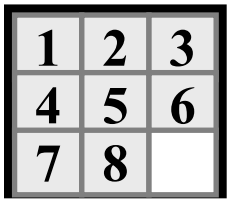
local minimum for heuristic; all moves make it higher (worse)



6



6



Goal

h(n): manhattan distance

Best-first search

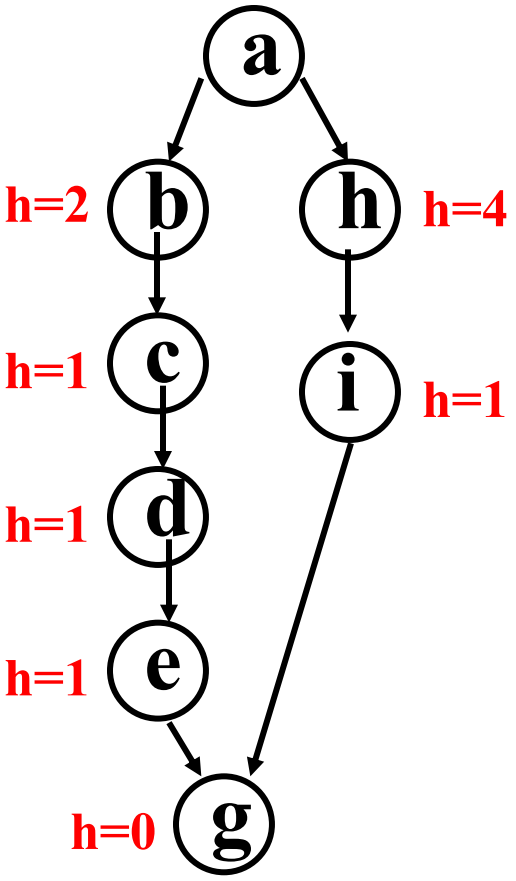
- Search algorithm that improves **depth-first search** by expanding **most promising node** chosen according to heuristic rule
- Order nodes on nodes list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information
- **f(n) = g(n) + h(n)** where
  - **g(n)** = 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

Will it find the best solution?

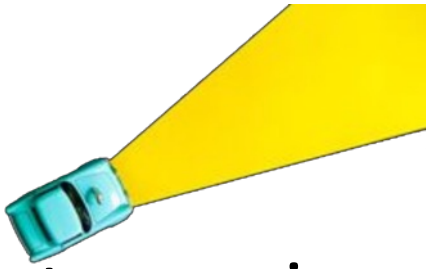
- We want search algorithms that find a solution if one exists (e.g., are **complete**)
- But do they find an **optimal** solution, i.e., the one with the lowest cost?
- This depends on both the
  - Search algorithm and
  - Heuristic used for the best state to expand
- Approaches guaranteed to find an optimal solution are said to be admissible

Greedy best first search 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



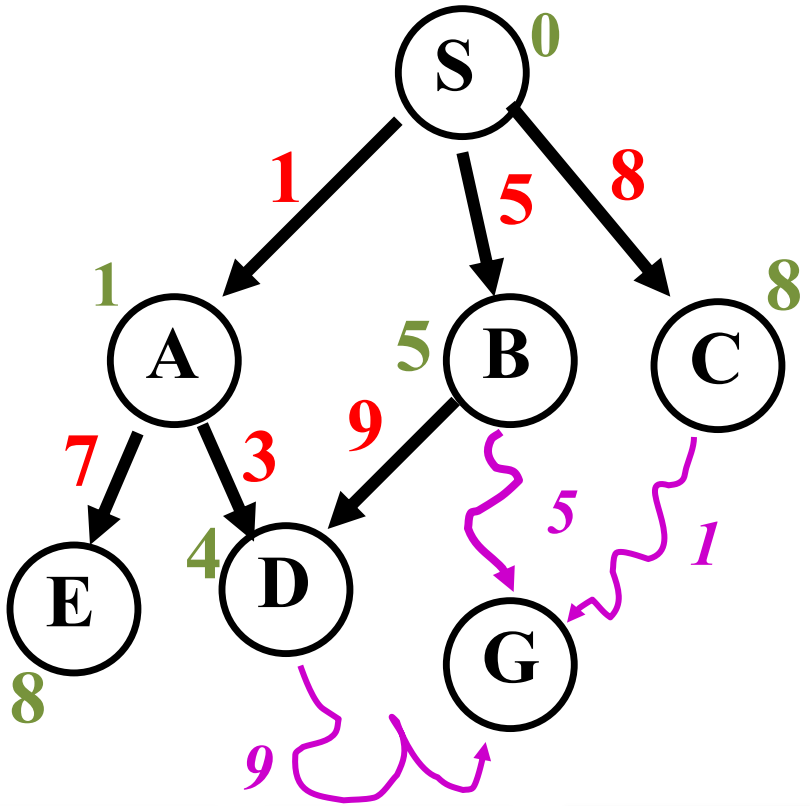
- Use evaluation function f(n)=h(n), but max size of the nodes list is k, a fixed constant
- Only keeps k best nodes as candidates for expansion, discard rest
- k is the ***beam width***
- More space efficient than greedy search, but may discard nodes on a solution path
- As k increases, approaches best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



g(d)=4
h(d)=9
f(d)=13

g(b)=5
h(b)=5
f(d)=10

g(c)=8
h(c)=1
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*

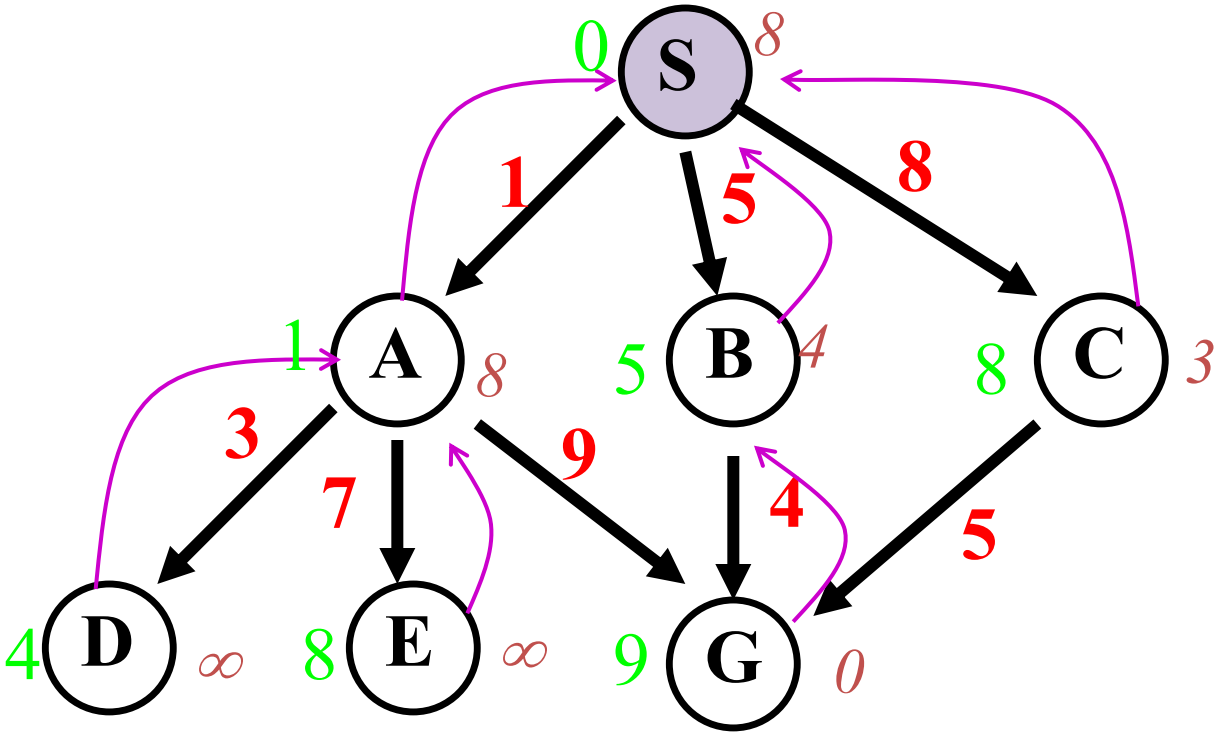
- Pronounced *"a star"*
- Algorithm A with constraint that **h(n) <= h*(n)**
  - **h*(n)** = *true cost* of *minimal cost path* from n to goal
  - So: h(n) ***never overestimates*** cost to get from n to goal
- h is **admissible** when h(n) <= h*(n) holds
- Using an admissible heuristic guarantees that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

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4* **4** (2): 100–107.

Observations on A

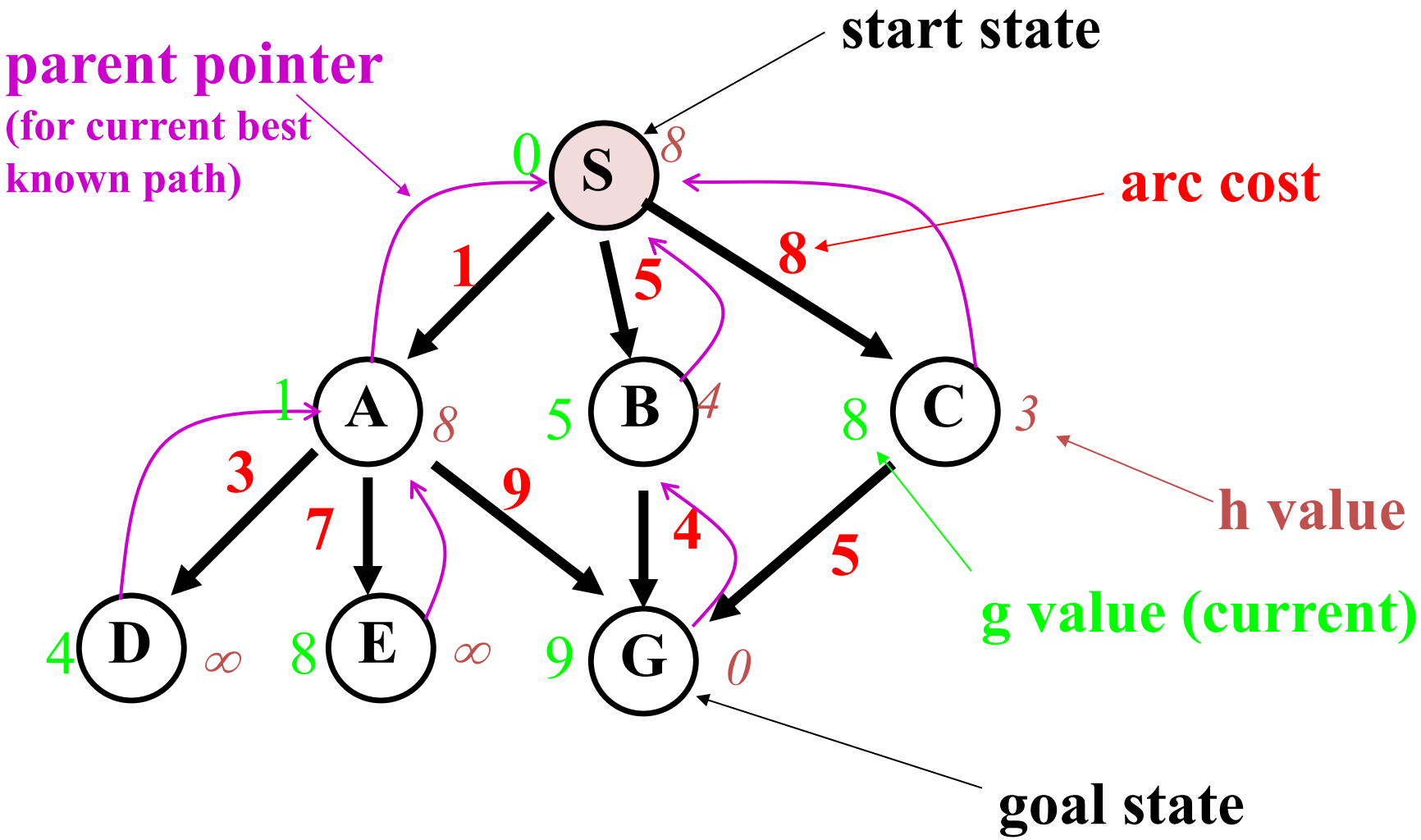
- **Perfect heuristic:** If h(n)=h*(n) for all n, only nodes on an optimal solution path expanded; no extra work done
- **Null heuristic:** If h(n) = 0 for all n, then it's an admissible heuristic; A* acts like uniform-cost search
- **Better heuristic:** If h1(n) **<** h2(n) **≤** h*(n) for all non-goal nodes, then h2 is a ***better*** heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*
  - i.e., A1 expands at least as many nodes as A2*
  - We say that A2* is *better informed* than A1*
- The closer h to h*, the fewer extra nodes expanded

Example search space

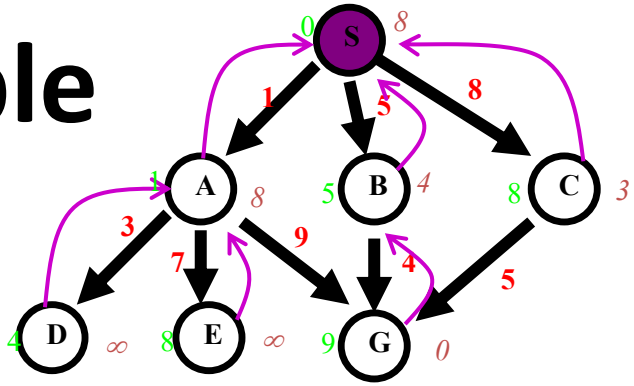


Next slide explains
the arcs and numbers

Example search space



Search Example



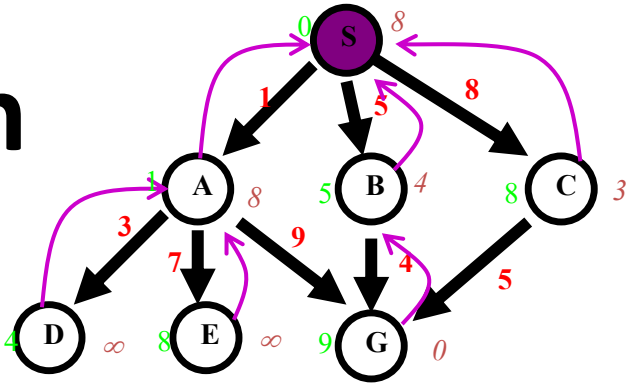
n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	∞	∞	∞
E	8	∞	∞	∞
G	9	0	9	0

The table and graph show values for the entire space, but we must discover or compute them during the search

- h*(n) is (hypothetical) perfect heuristic (an oracle*)
- Since h(n) <= h*(n) for all n, h is admissible (optimal)
- Optimal path = *S B G* with cost 9

* An **oracle** is a person or agency considered to provide wise and insightful counsel or prophetic predictions,

Greedy search



f(n) = h(n)

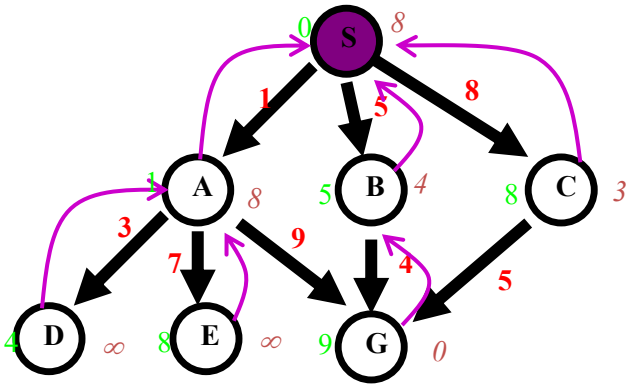
node expanded	nodes list
	{S(8)}
S	{C(3) B(4) A(8)}
C	{G(0) B(4) A(8)}
G	{B(4) A(8)}

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        {G(10)  C(11)  D(inf)  E(inf)}
```

current f(B) = 5 + 4

h(n)

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

- Stop when we've found a path to a goal and there's no other node with a f(n) value less that the path's cost
- Solution path found is **S B G**, 3 nodes expanded.
- Estimates total cost of path to try to find global optimum
- Still pretty fast and optimal, too [since h(n) ≤ h*(n)]

Dealing with hard problems

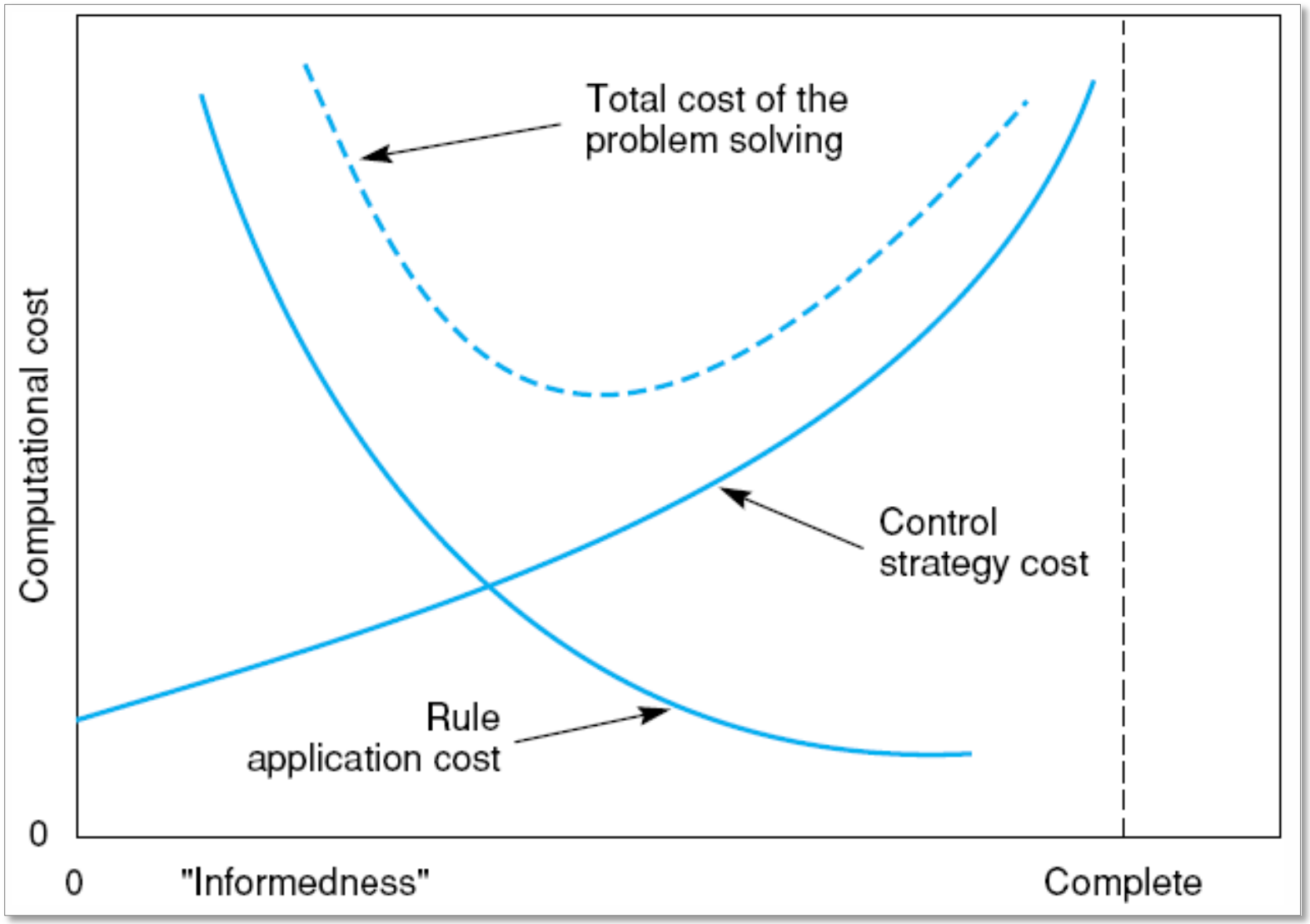
- A* is like a "smart" Best First Search (BFS)
- For large problems, A* may need too much space
- Variations conserve memory: IDA* and SMA*
  - May not guarantee an optimal solution
- IDA*, iterative deepening A*, uses successive iteration with growing limits on f, e.g.
  - A* but don’t consider a node 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, ...
- SMA* -- Simplified Memory-Bounded A*
  - Uses queue of restricted size to limit memory use

Finding good heuristics

- **Relaxing problem:** remove constraints for easier problem; use its solution cost as heuristic function
- **Ranking heuristics:** If h1(n) < h2(n) <= h*(n) for all n, h2 is a better heuristic than h1
  - We also say that h2 **dominates** h1
- 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 impactable 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** just chooses node with smallest g(n) to expand next (complete and optimal)
- **Greedy search** chooses node with smallest h(n) to expand next (neither complete nor optimal but may be fast)
- **Algorithm A search** combines uniform-cost search and greedy search: f(n) = g(n) + h(n).
- **Algorithm A*** requires that h(n) <= h*(n), i.e., never overestimates
  - Complete and optimal, but space complexity high
  - Time complexity depends on quality of heuristic function
  - IDA* and SMA* reduce the memory requirements of A*