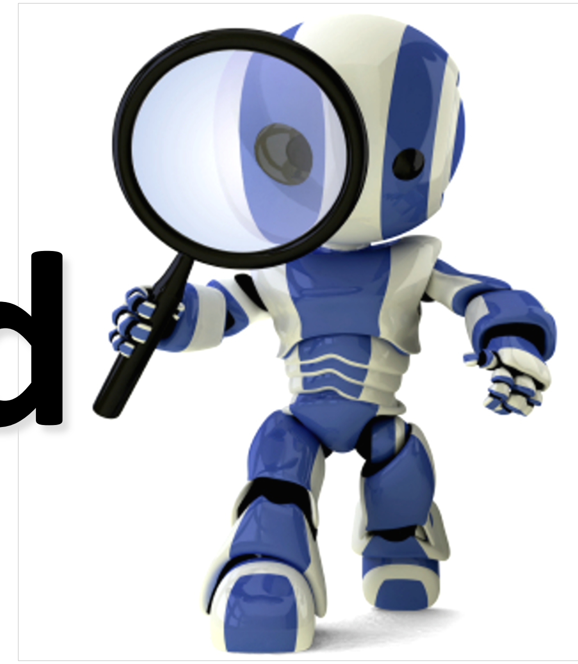
Informed Search

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*- ˈris-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**

The Free On-line Dictionary of Computing (15Feb98)

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**.

From WordNet (r) 1.6

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 best 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**.
- Based on domain-specific information and computable from current state description that estimates how close we are to a goal

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 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

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

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 just 3 more moves
- Very rough estimate, but easy to calculate

h = 3

Heuristics for 8-puzzle

Manhattan Distance Heuristic

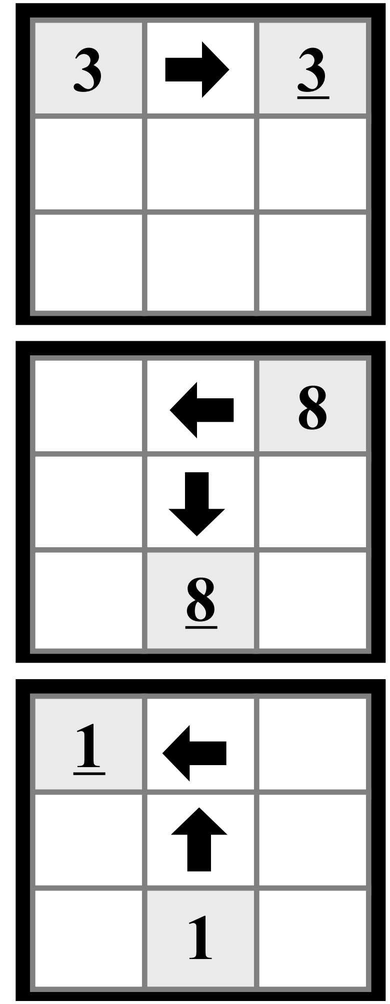
(not including the blank)

Current State

3	2	8
4	5	6
7	1	

Goal State

1	2	3
4	5	6
7	8	



2 steps

3 steps

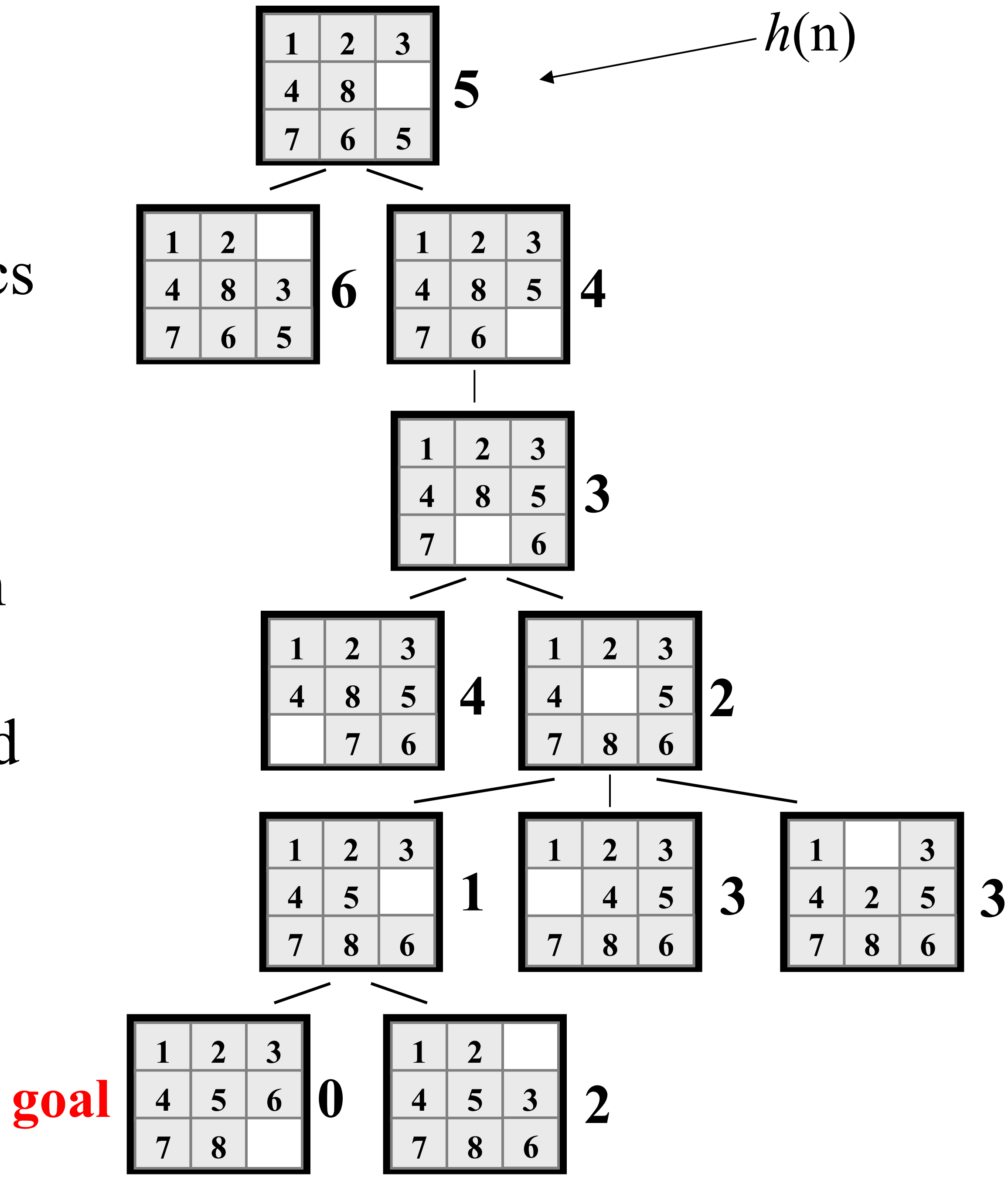
3 steps

- 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 just 8 more moves
- More accurate than the misplaced heuristic, but slightly more expensive to compute

h = 8

We can use heuristics to guide search

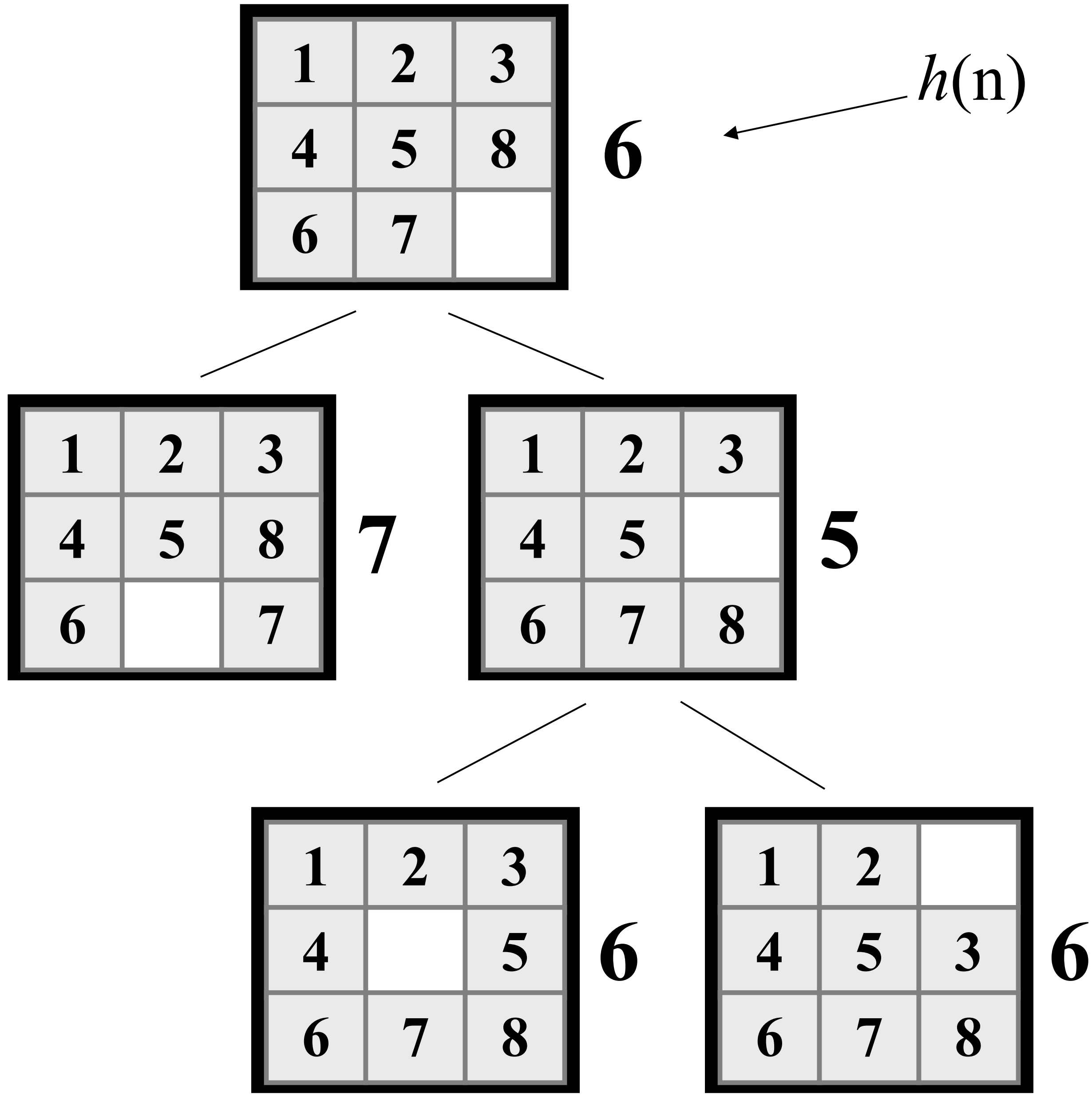
In this ***hill climbing*** example, Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle



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

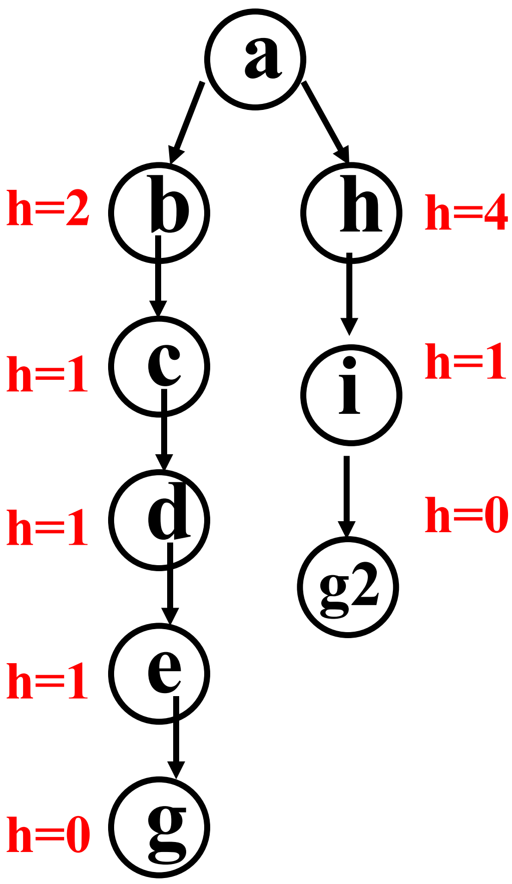


Best-first search

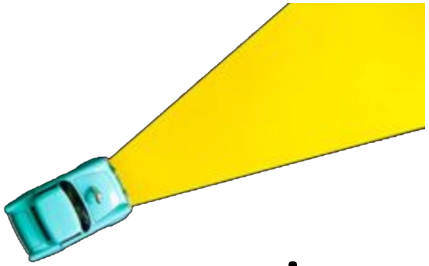
- Search algorithm that improves **depth-first search** by expanding most promising node chosen according to heuristic rule
- Order nodes on nodes list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information
- $f(n) = g(n) + h(n)$ where
  - $g(n)$ = distance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

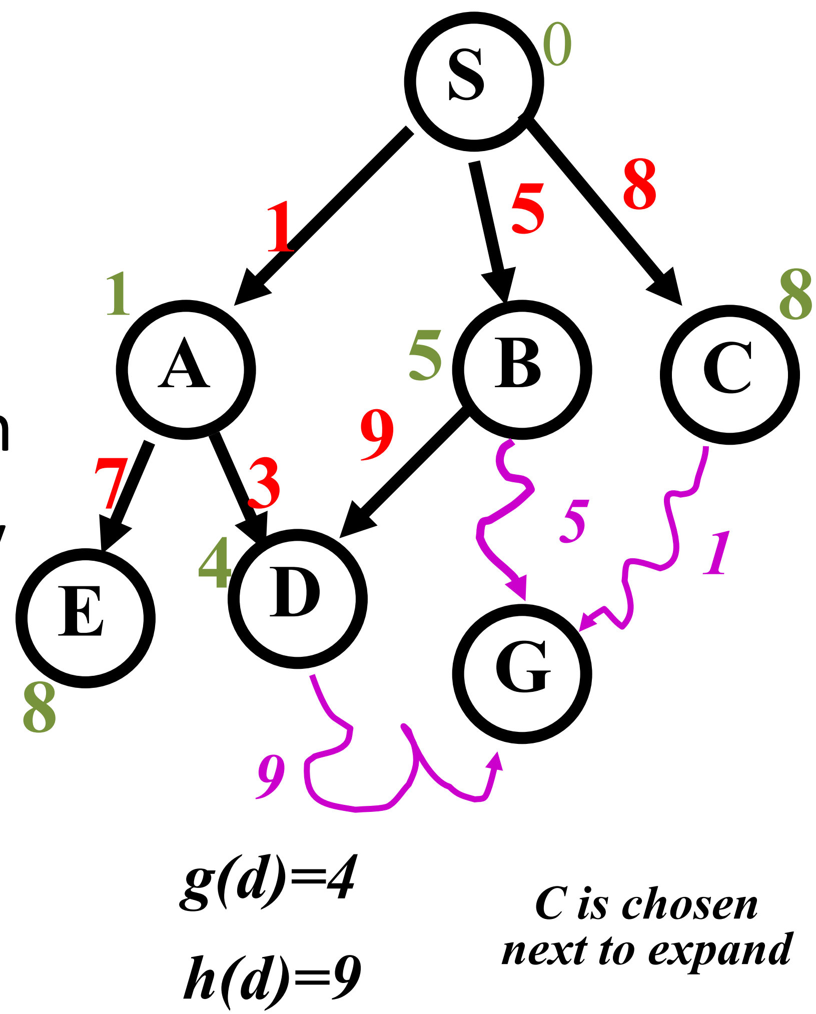


- Use evaluation function $f(n)=h(n)$, but max size of the nodes list is k, a fixed constant
- Only keeps k best nodes as candidates for expansion, discard rest
- k is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As k increases, approaches best 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
- $g(n)$ term adds "breadth-first" component 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)



Algorithm A

1 Put the 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'), 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*

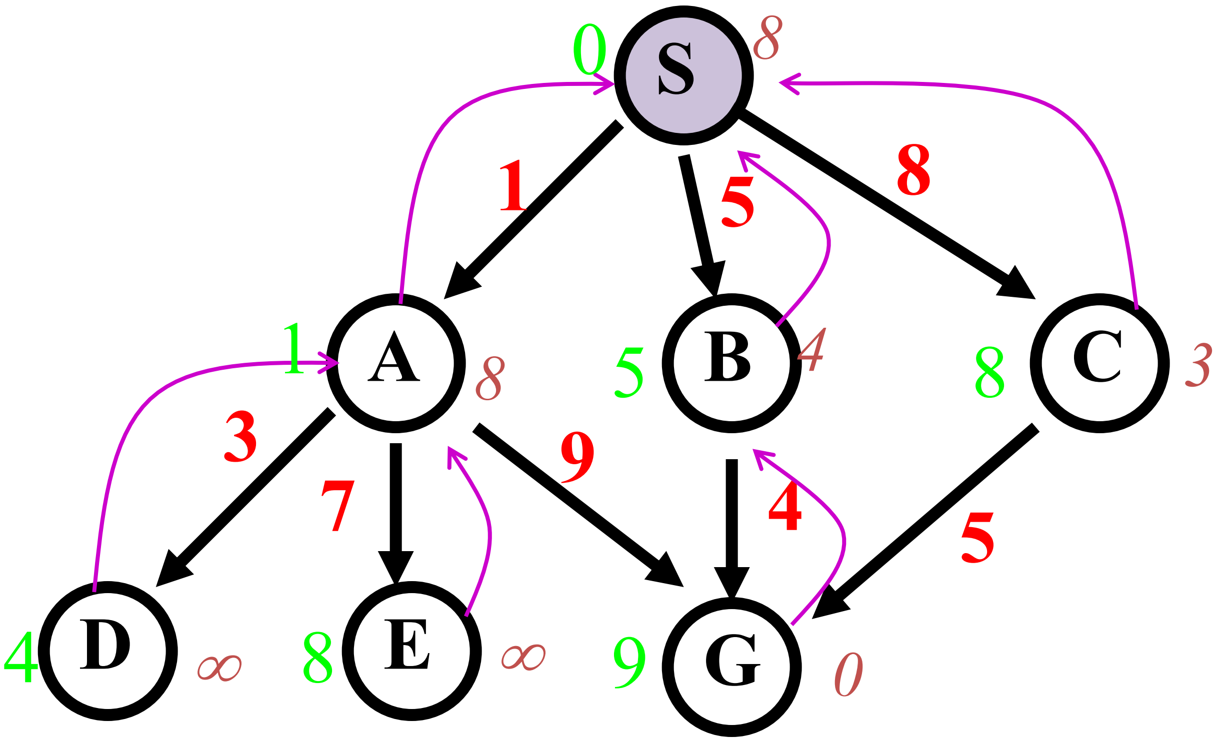
- Pronounced *"a star"*
- Algorithm A with constraint that **h(n) <= h*(n)**
  - **h*(n)** = *true cost* of *minimal cost path* from n to goal
  - So: h(n) *never overestimates* cost to get from n to goal
- h is **admissible** when h(n) <= h*(n) holds
- Using an admissible heuristic guarantees that 1st solution found will be an **optimal** 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.

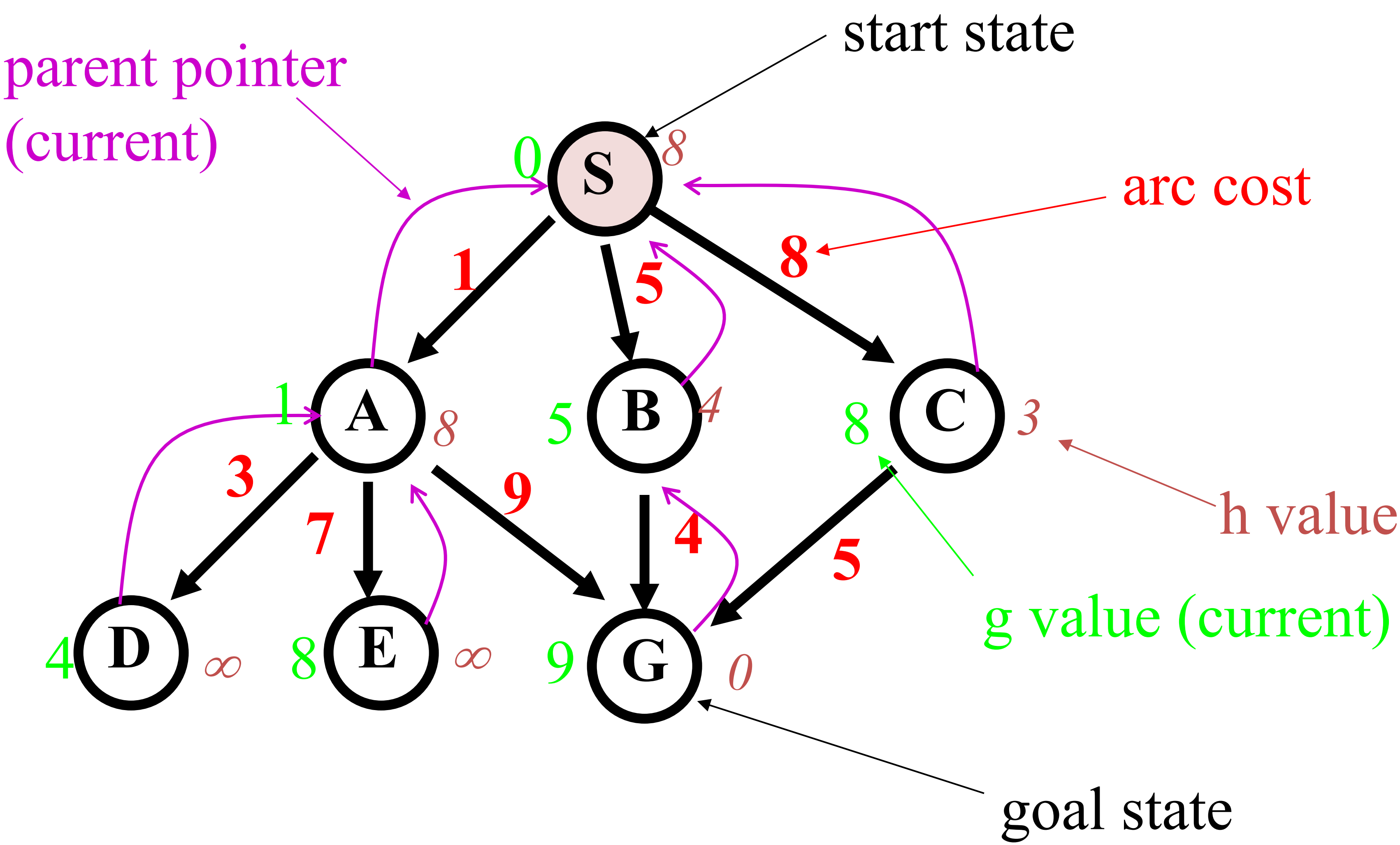
Observations on A

- **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

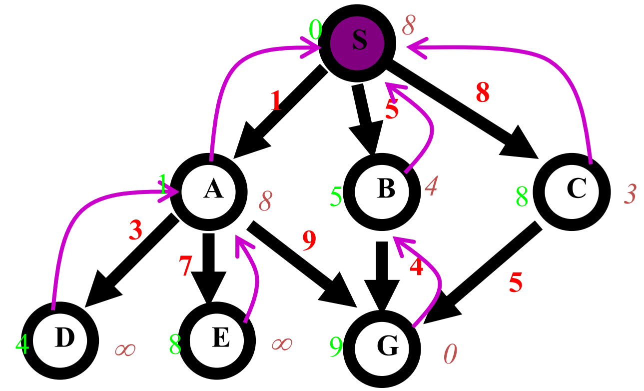
Example search space



Example search space



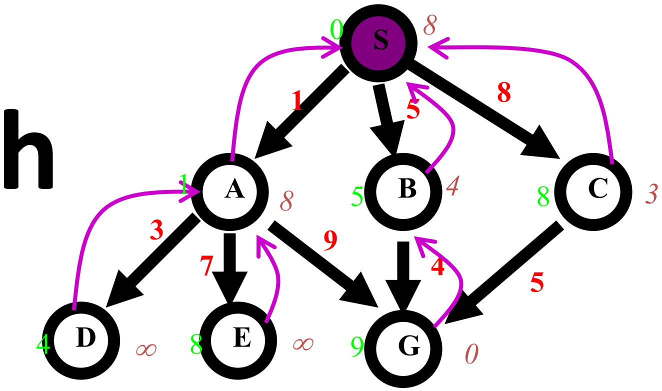
Example



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	∞	∞	∞
E	8	∞	∞	∞
G	9	0	9	0

- 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

Greedy search

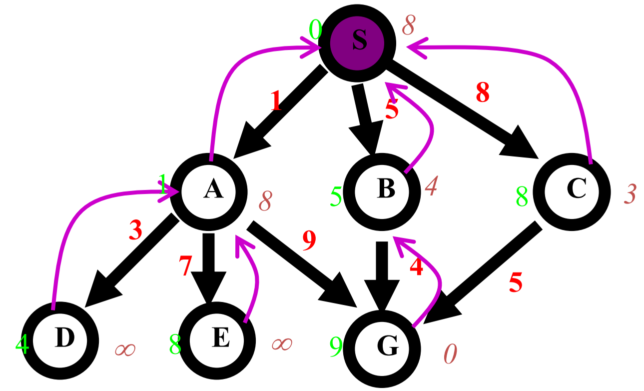


f(n) = h(n)

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) }

- Solution path found is S C G, 3 nodes expanded.
- See how fast the search is!! But it is NOT optimal.

A* search



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

```
node exp.        nodes list
              { 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) }
```

- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast. And optimal, too [assuming h(n)<h*(n)]

Proof of the optimality of A*

- Assume that A* has selected G2, a goal state with a suboptimal solution, i.e., g(G2) > f*
- Proof by contradiction shows it's impossible
  - Choose a node n on an optimal path to G
  - Because h(n) is admissible, f* >= f(n)
  - If we choose G2 instead of n for expansion, then f(n) >= f(G2)
  - This implies f* >= f(G2)
  - G2 is a goal state: h(G2) = 0, f(G2) = g(G2).
  - Therefore f* >= g(G2)
  - 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, ...
- SMA* -- Simplified Memory-Bounded A*
  - Uses queue of restricted size to limit memory use

Finding good heuristics

- If h1(n) < h2(n) <= h*(n) for all n, h2 is better than (**dominates**) h1
- **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

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*