Informed search for plans

Goal information

- The algorithms we discussed until now (depth first, breadth first, uniform cost search) assumed that the only information we have about the goal is a binary $G(\cdot)$ function
- We know that we found the goal, when we get there.

Goal information (cont'd)

- How do we go from Orlando to San Francisco?
  - It is to the west from us.
  - So we probably have to go mostly to west
  - But taking every action "to west" does not take you there
- In practice, we might have more information about the goal
  - But this information can be vague, incomplete, uncertain, probabilistic or wrong
- **Challenge:** how do we integrate additional information about the goal into our

Heuristics

- A function that provides an estimate for how far is a state from the goal $h(s)$
- It is a way to encapsulate knowledge about the goal
- Examples:
    - The "as-the-crow-flies" distance to San Francisco
    - The number of horcruxes remaining

Greedy search

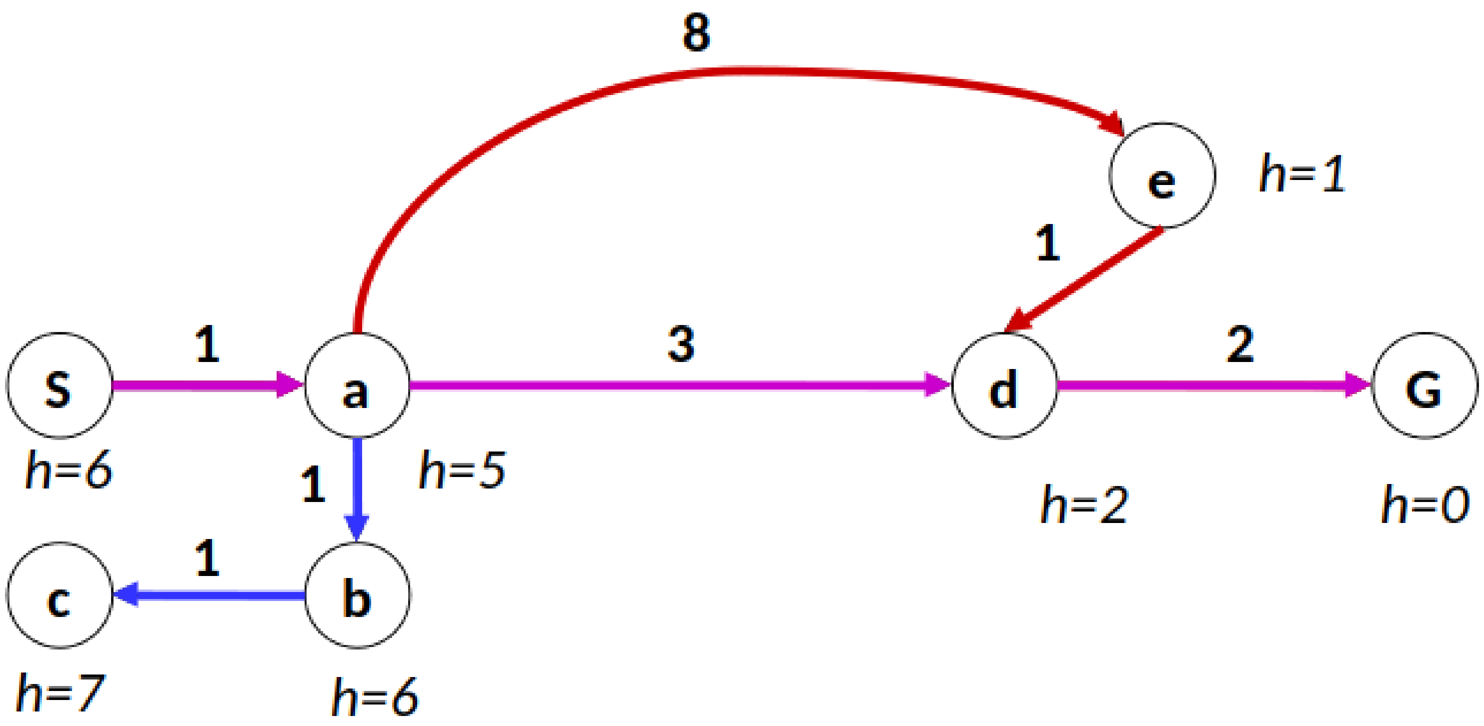
- Strategy: expand the node with the lowest heuristic value
  - Make the fringe a priority queue ordered by $h(s)$
  - Pick the smallest
- Sometimes also called as **best-first search**
- How good it is? Depends on the quality of the heuristics
  - If the heuristics gets the ordering right (not necessarily the values) - you go straight to the solution!
  - If the heuristic is wrong, you can end up like in DFS
- The quality of the heuristics reflects our understanding of the problem.

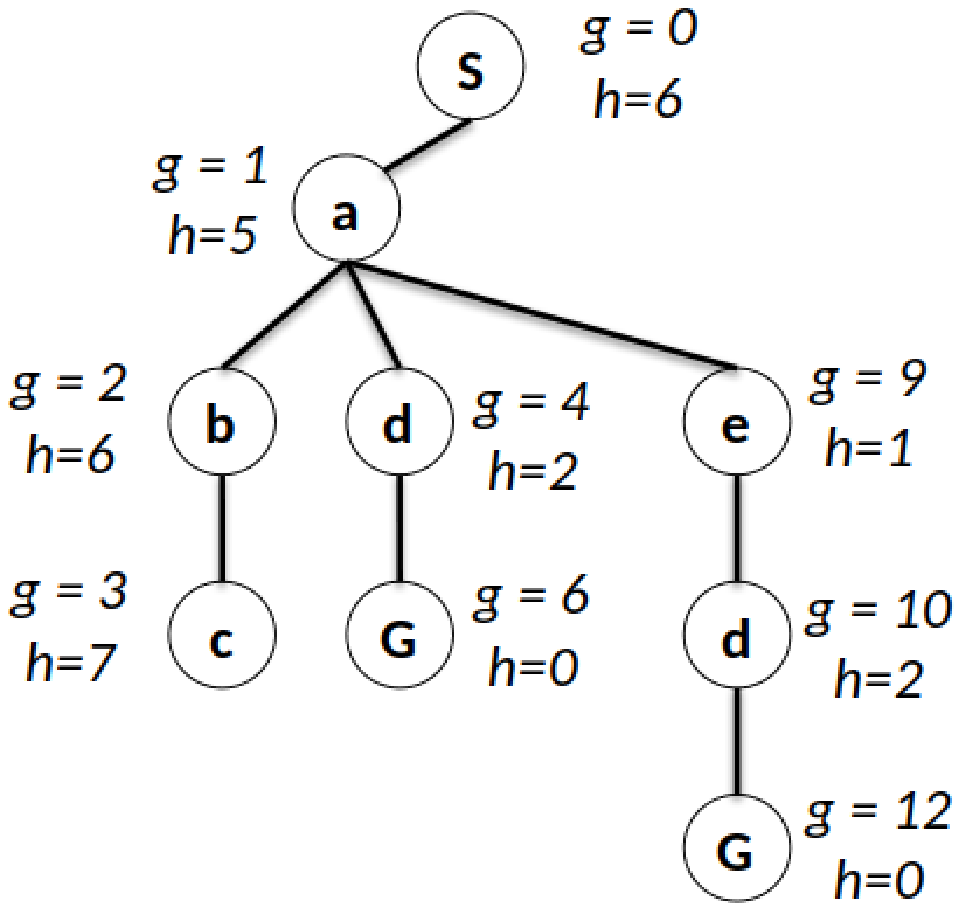
Greedy search (cont'd)

- Optimal: no, the heuristics might lead you on a non-optimal path or to the non-optimal goal.
- Space and time complexity: can range anything between BFS and DFS.
- Insight: DFS and BFS are heuristic search with a particular type of heuristic
- Can you get **stuck**?
  - No, if you are following the standard tree search algorithm - you will explore the other ones later.
  - But you can end up endlessly deep, like in DFS.

A* search

- Combines UCS and Greedy
  - Uniform cost orders by path cost $g(n)$ aka *backward cost*
  - Greedy orders by goal proximity $h(n)$ aka *forward cost*
  - A* orders by sum $f(n) = g(n) + h(n)$


8
e
h=1
1
S
1
a
3
d
2
G
h=6
1
h=5
h=2
h=0
c
1
b
h=7
h=6



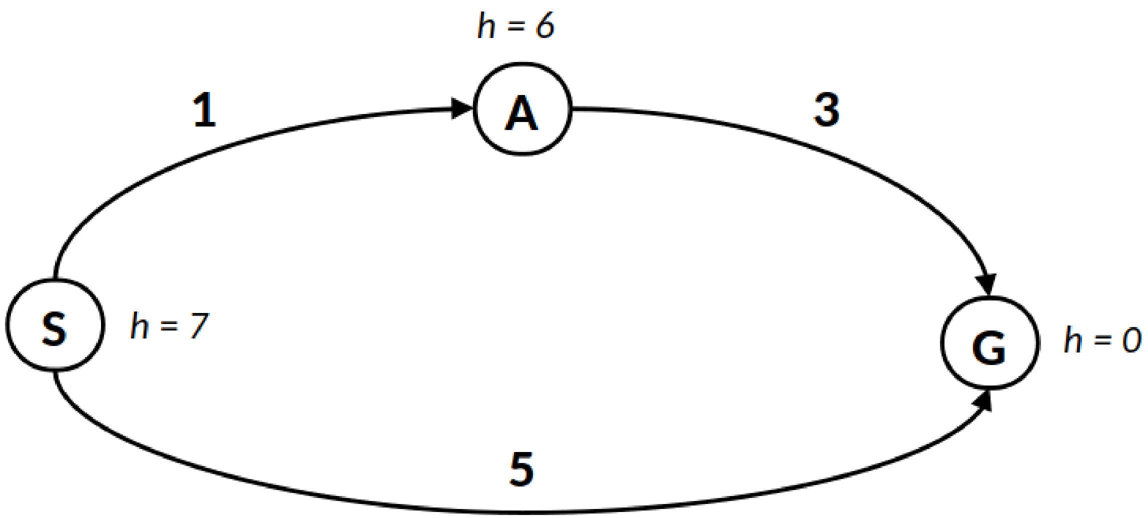
S
g = 0
h=6
a
g = 1
h=5
b
g = 2
h=6
d
g = 4
h=2
e
g = 9
h=1
c
g = 3
h=7
G
g = 6
h=0
d
g = 10
h=2
G
g = 12
h=0

When should A* terminate

- Don't stop when we add the goal to the fringe!
    - The fringe is not FIFO - it is possible that the goal we added is not the one that will come out first!
- Only stop when we take out a node labeled with a goal from the fringe

Is A* optimal?

- Not in this case!
- The heuristic misled us!
- But if we need a perfect heuristic, why do we bother with A*
- Turns out we don't need the heuristic to be perfect, we only need it to be **optimistic**



Admissible heuristics

- Inadmissible (pessimistic) heuristics break optimality by trapping good plans far down on the fringe
- Admissible (optimistic) heuristics never overweigh true costs:

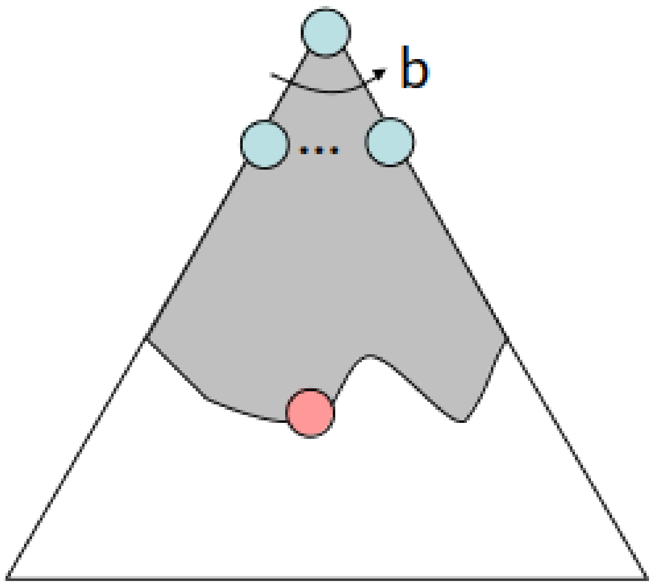
$$0 \leq h(n) \leq h^*(n)$$

- where $h^*(n)$ is the true cost to a nearest goal.

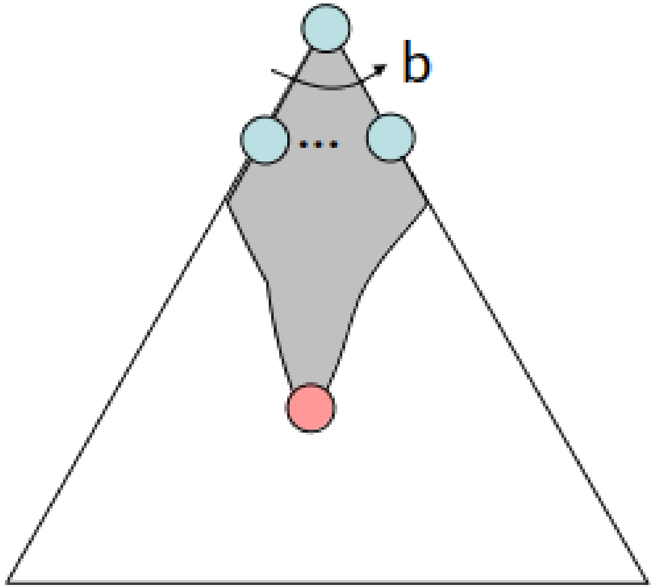
A* properties

- Uniform cost expands equally in all directions
- Greedy expands sharply towards what it thinks is the goal
- A* expands mainly towards the goal but also other directions

Uniform-Cost



A*



A* applications

Very extensive set of applications

- Pathing, routing problems
- Resource planning problems
- Video games
- Robot motion planning

Previously also used for

- Language analysis
- Machine translation
- Speech recognition

Creating admissible heuristics

- The critical challenge in making A* work for you is to come up with a good admissible heuristic
- Trivial admissible heuristic: $h(n) = 0$
  - Reverts A* to uniform cost search
- Perfect heuristic $h(n) = h^*(n)$
  - Go straight to the goal
- There is a partial ordering between admissible heuristics (*dominance*)
- The max of admissible heuristics is admissible

$$h(n) = max(h_a(n), h_b(n))$$

Relaxed problems

- How do we get good admissible heuristics?
- One way: try to solve a **relaxed problem**
  - A problem which is in some way easier than the original one
- One easy way to create a relaxed problem: add new actions
  - Imagine that the agent is a superhero!!!
  - Eg. ability to fly - Euclidean distance
  - Eg. ability to pass through walls - Manhattan distance
  - Eg. ability to destroy horcruxes from distance - horcrux count

Extra work in tree search

- Until now, all the algorithms were variations of **tree search**
    - You can have many plans in the tree labeled with the same node
    - Can lead to (exponentially more) extra work

Graph search

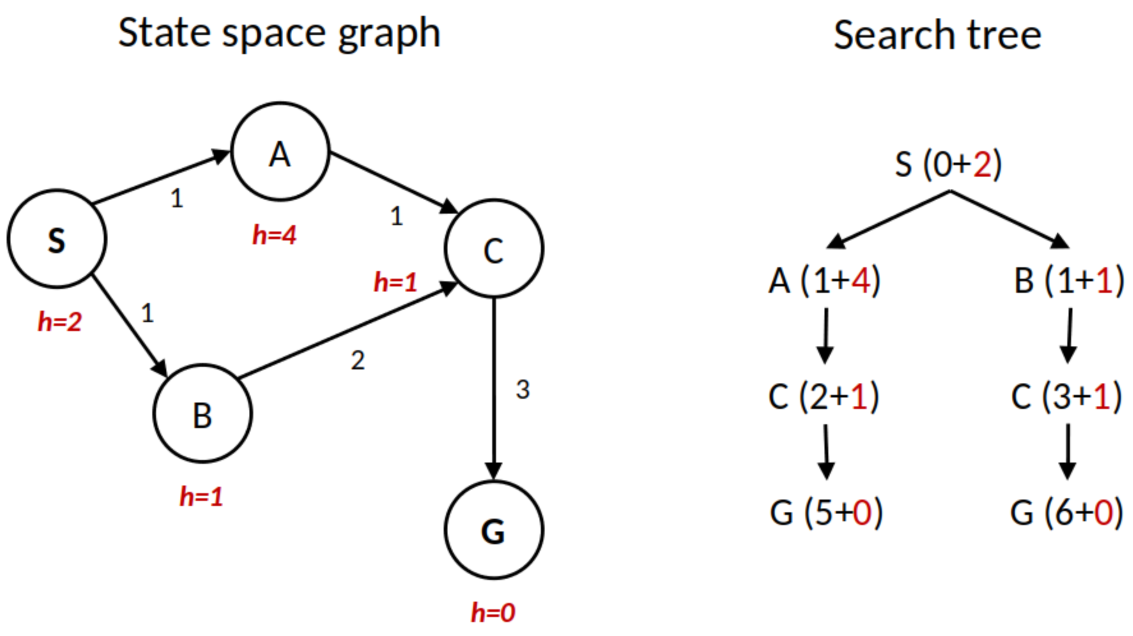
- Idea: never **expand** a state twice
- Augment the tree search algorithm with a **closed set** the set of expanded states
- Before expanding a node, check if the state was expanded before
  - Yes: skip it
  - No: expand it and add it to the closed set
- The closed set only used for membership check: implement as a hashset.

Graph search properties

- Any tree search algorithms can be converted to graph search
- Graph search obviously avoids some expansions
- Does it change the properties?
  - Space complexity: increased, due to the closed set
  - Completeness: whatever states had been expanded before, they will be expanded now as well, so the algorithm retains completeness
  - Optimality?

A* graph search optimality

- Admissible heuristic not sufficient
- Heuristics also needs to be **consistent**



Consistent heuristics

- Admissibility: heuristic cost $\leq$ actual cost to goal
  $h(A) \leq$ actual cost from A to G
- Consistence: heuristic "arc" cost $\leq$ actual cost for each arc
  $h(A) - h(C) \leq$ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases
  - A* graph search is optimal
- How do we find consistent heuristics?
  - Relaxed problems will be consistent