EECS 391: Introduction to AI

Soumya Ray

Website: http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tue 2:30-4:00 or by appointment

Announcements

- HW1 is due at 5pm
  - Can submit electronically through blackboard
  - Solutions posted Monday noon
- Quiz 1 next week (Tuesday)
  - Introduction, Agent architecture, Uninformed search and informed search

Today

- Informed Search
- Intro to Search for Optimization

Review

- What do we need to set up a search problem?
- What are the basic steps of all search algorithms?
- How do search algorithms differ?

Basic Steps of Search Algorithms

- Add initial state to open list (list of unvisited states)
- While open list is not empty
  - Remove node to be expanded from the open list
  - If this is the goal, solution found, return path
  - Else
    - Find the successors of this node ("expanding a state")
    - Add the current state to the list of visited (expanded) states (closed list)
    - Add the new states to the open list (if they do not appear in the closed list)
      - State could already be in open list, set parent pointer correctly (based on minimum path cost)

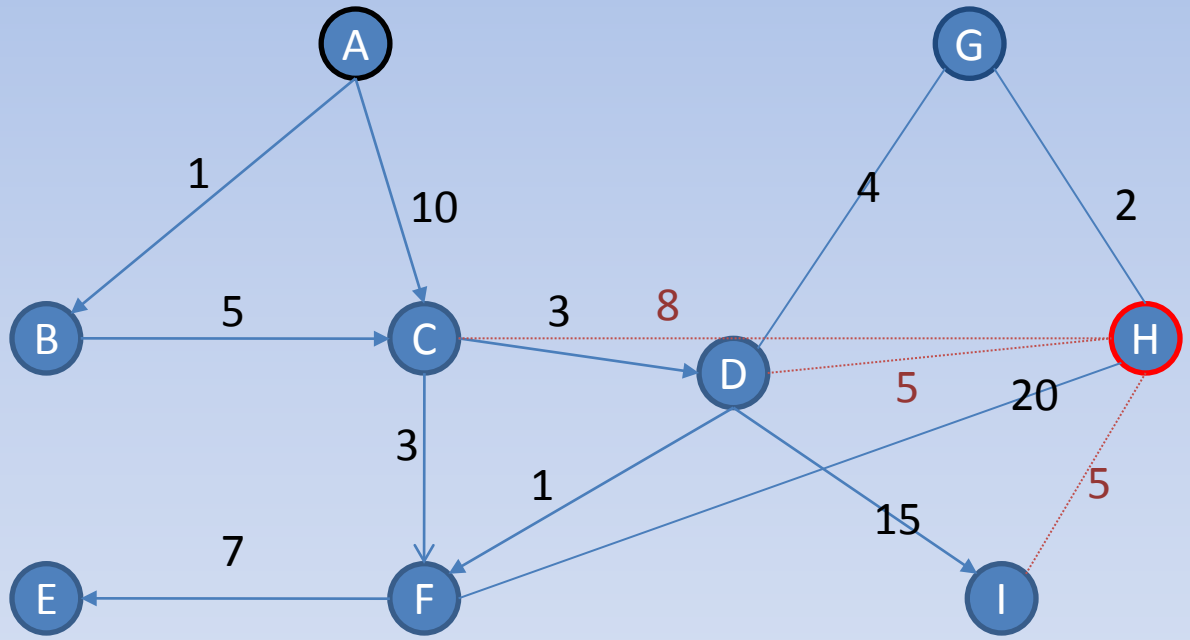
Search algorithms differ here

Search Heuristic

- Suppose we had a function that *estimated* the distance to the goal for a node n
  - Call it $h(n)$ (remember path cost is denoted $g(n)$)

- This is called a search heuristic
  - This function depends on the specific problem and is not externally specified; has to be designed by us or possibly learned by the agent
  - Algorithms using $h(n)$ are also sometimes called "best-first" algorithms

Example

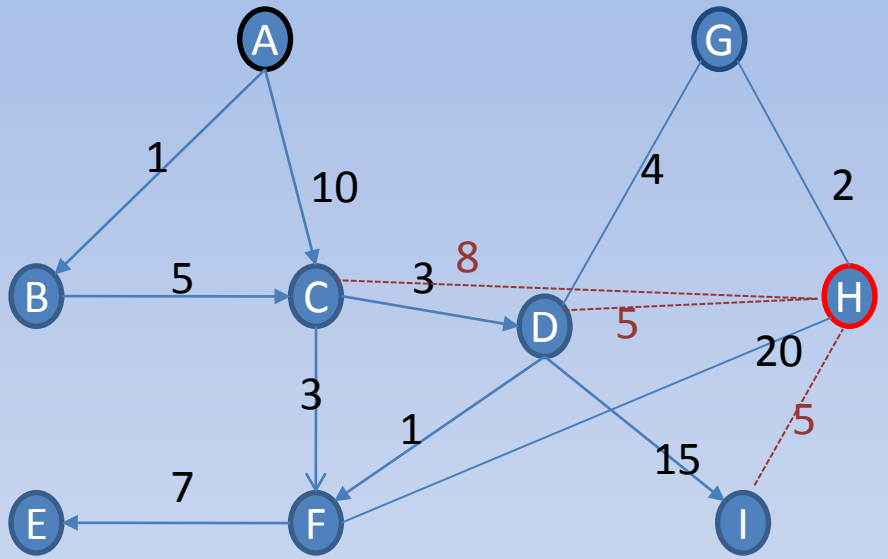


Heuristic: straight line distance (imagine straight lines joining each node to H)

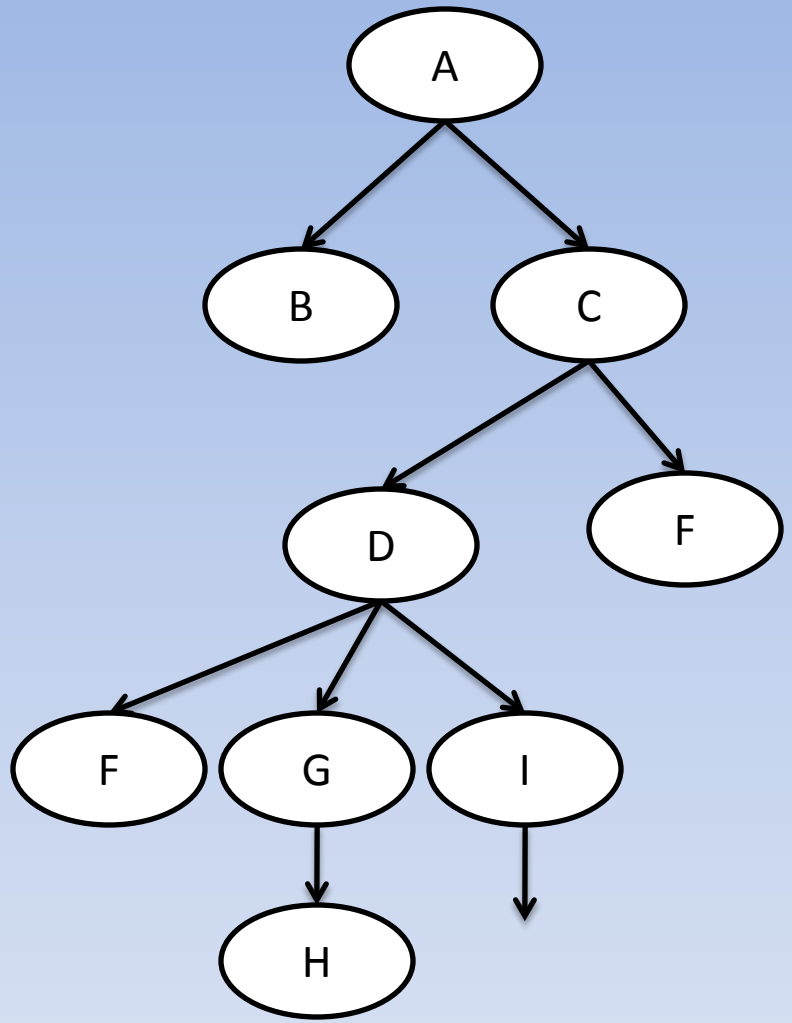
Greedy Best-first Search

- Nodes on open list with smallest $h(n)$ are expanded first
- Expand successors of initial state, choose the one with lowest $h(n)$ and expand it, etc
  - Like DFS
- The open list will be implemented with?

Example



Heuristic: straight line distance (imagine straight lines joining each node to H)



Greedy Search Performance

- Complete?
  - Yes, assuming no infinite paths
- Optimal?
  - Sometimes
- Time Complexity?
  - $O(b^m)$
- Space Complexity?
  - $O(b^m)$

Fixing Greedy Search

- We saw one problem with greedy search: because it ignored $g(n)$, it could be suboptimal
- What if, instead of $h(n)$, we used the function $f(n)=g(n)+h(n)$?
- The resulting algorithm is called A* (A-star) and turns out to have some great properties

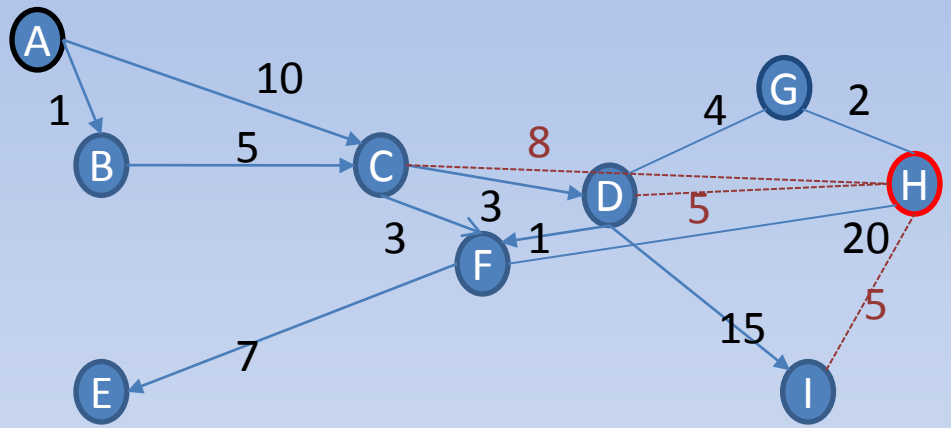
A* Search

- Nodes on open list with smallest $f(n)=g(n)+h(n)$ are expanded first

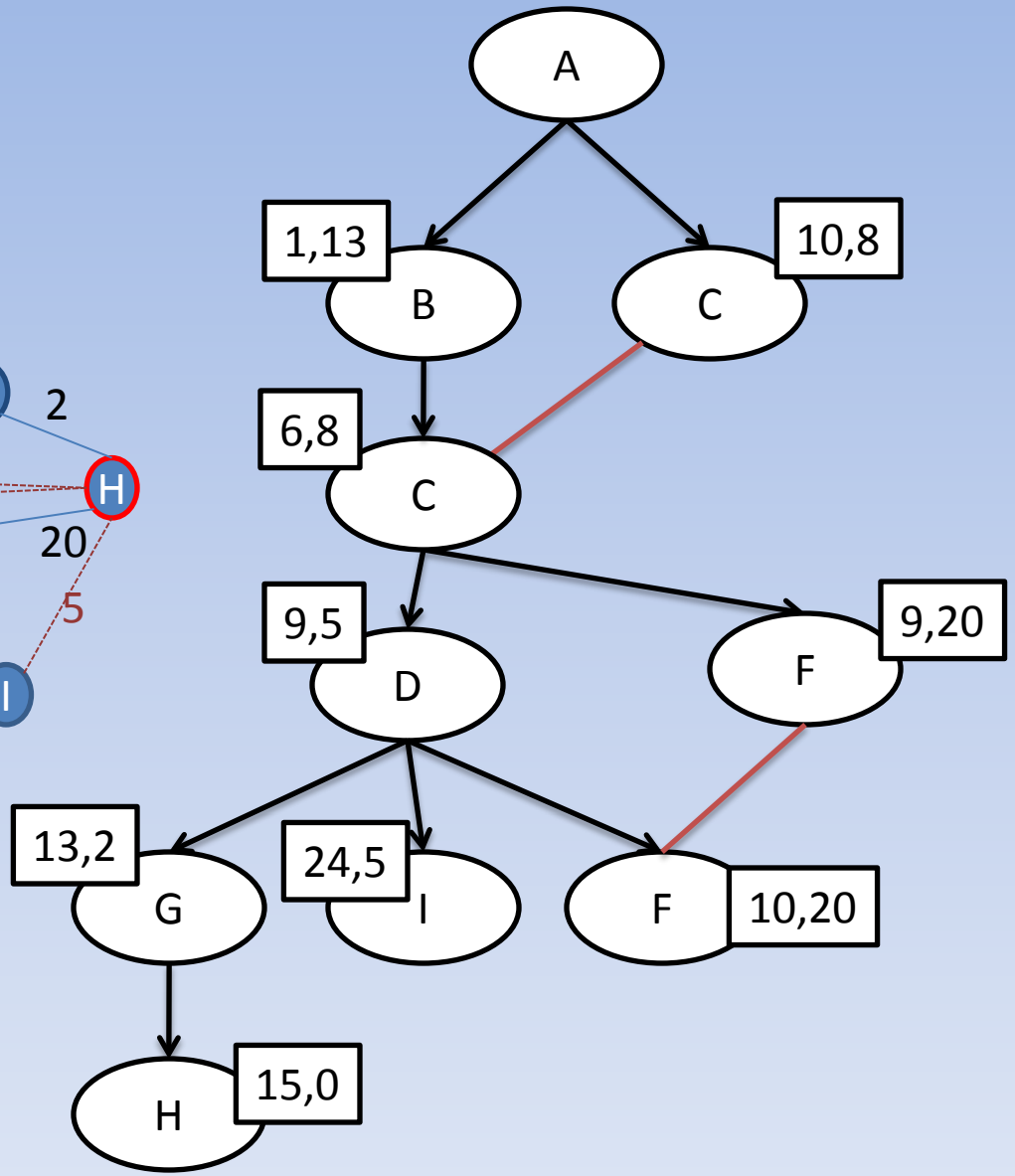
- Expand successors of initial state, choose the one with lowest $f(n)$ and expand it, etc

- The open list will be implemented with?

Example



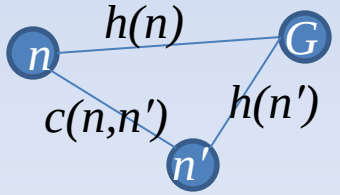
Heuristic: straight line distance (imagine straight lines joining each node to H)



Optimality of A*

- If the heuristic function $h(n)$ satisfies certain properties, A* is both complete and optimal

- Properties of $h(n)$
  - Admissibility: $h(n)$ never overestimates the true cost to the goal (every goal) from n
  - Consistency: for every node and successor pair (n,n'), $h(n) < c(n,n') + h(n')$



Effect of admissibility

- Suppose a suboptimal goal node SG is on the open list, and the "global minimum" goal is TG. Then

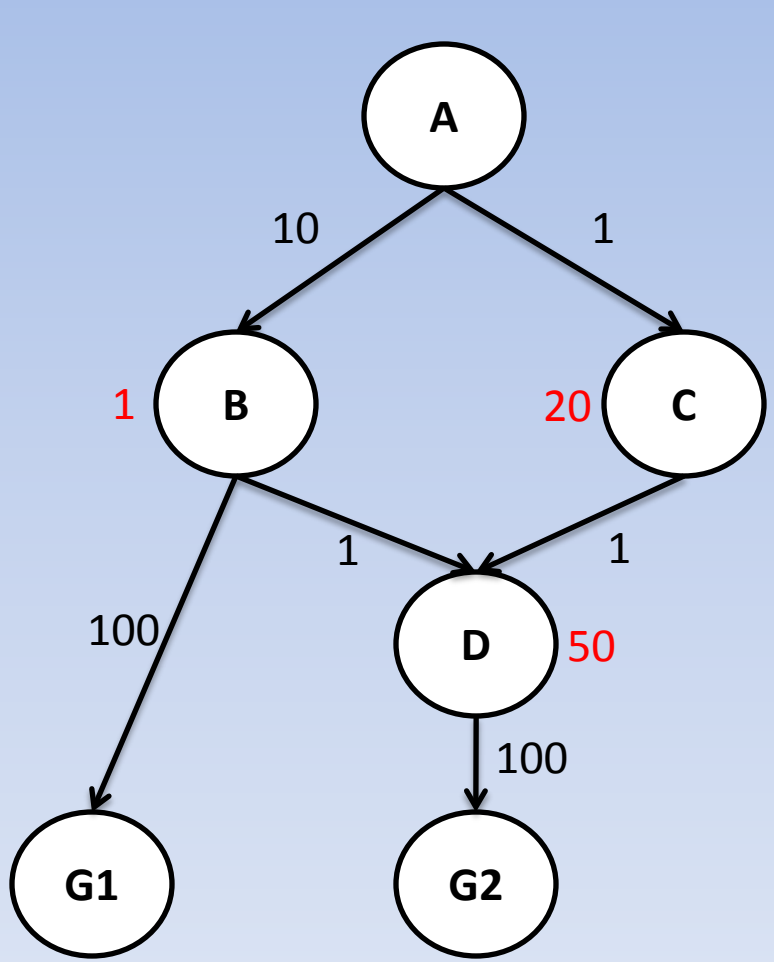
$$f(SG) = g(SG) + h(SG) = g(SG) > g(TG)$$

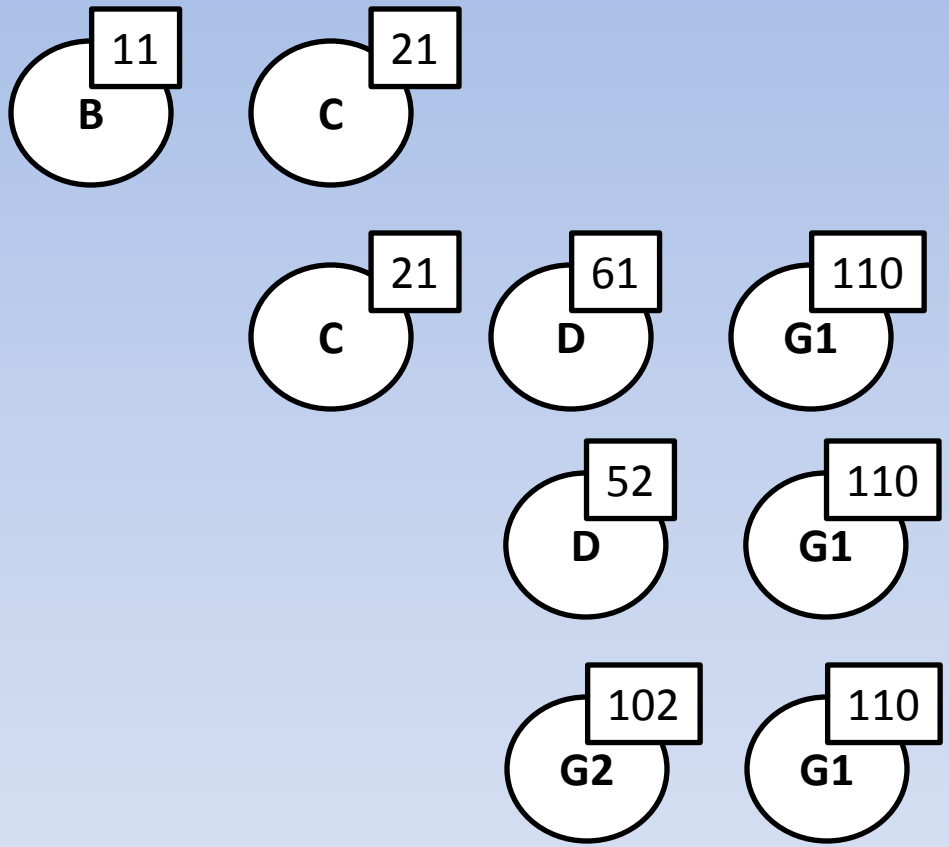
- Consider another node n on the open list that is on the path to TG (must exist)

$$f(n) = g(n) + h(n) \leq g(TG) \quad \text{(admissibility)}$$

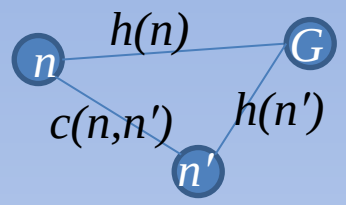
- So $f(n) \leq g(TG) < f(SG)$, and SG will not be expanded

Example


A
10
1
1
B
20
C
1
1
100
D
50
100
G1
G2


11
B
21
C
21
C
61
D
110
G1
52
D
110
G1
102
G2
110
G1

Effect of consistency

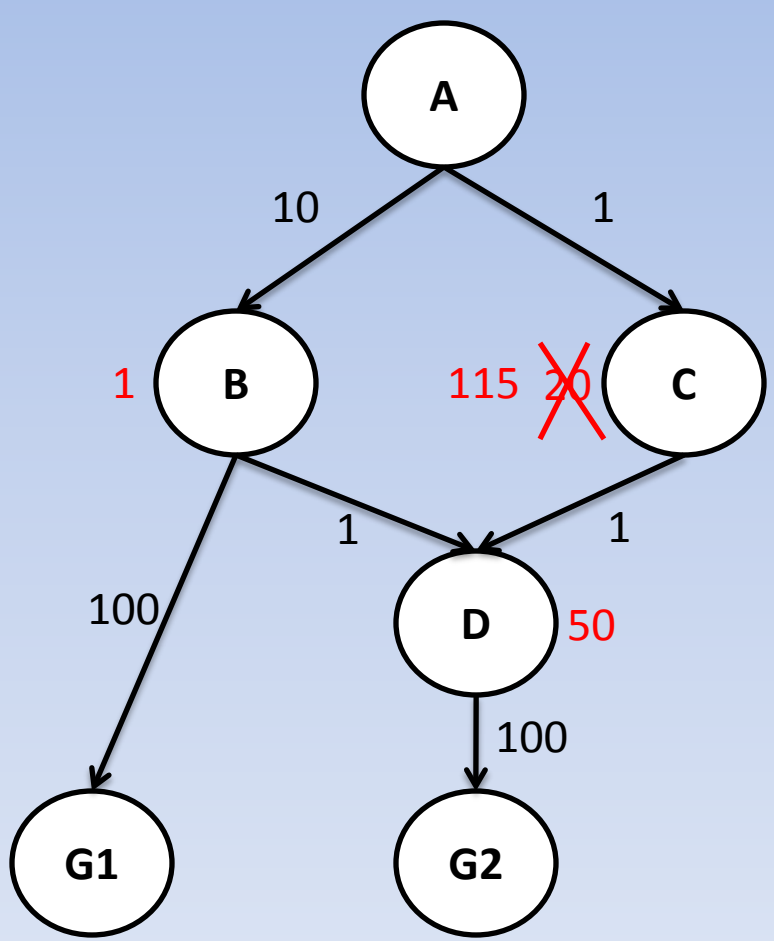


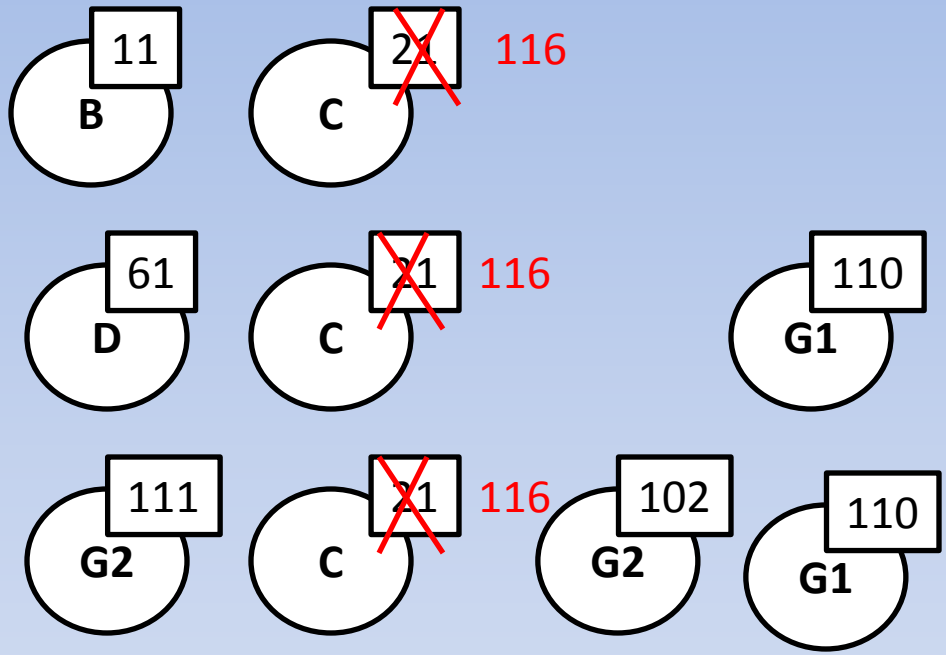
- Consistency implies that *f* increases monotonically with depth
- For any node and successor (n,n'),

$$\begin{aligned} f(n') &= g(n')+h(n') \\ &= g(n)+c(n,n')+h(n') \\ &> g(n)+h(n) \\ &> f(n) \end{aligned}$$

- So first time a node is expanded, we have the optimal path *to that node*
  - Corollary: The first time we expand a goal, we have the optimal path to it

Example


A
10
1
1
B
115
20
C
1
1
100
D
50
100
G1
G2


11
B
C
116
61
D
C
116
110
G1
111
G2
C
116
102
G2
110
G1

Optimal Efficiency of A*

- A* expands all nodes such that $f(n) < C_{opt}$
  - It also expands some nodes for which $f(n) = C_{opt}$
  - It does not expand any nodes for which $f(n) > C_{opt}$
    - These nodes are pruned from the search space

- Consider any other optimal forward-search algorithm with the same heuristic. This algorithm will expand at least as many nodes as A* (modulo tie-breaking on $f(n) = C_{opt}$)
  - "Optimal Efficiency" of A*

So are we done?

- More or less
  - A* is complete and optimal ☺
  - It is also optimally efficient ☺
  - But what about time and space complexity? ☹

- Turns out unless $h(n)$ is really really accurate (within a logarithmic factor of the optimal), the space explored by A* will be exponential in the length of the solution path
  - So, still something like $O(b^d)$
  - So people have been working on variants

Designing Heuristic Functions

- The availability of a heuristic makes a big difference in search
- But how do we get them?
- And once we have one how can we tell if they are any good?
- And if we have *two*, how do we tell which one is better?

3. Comparing heuristics

- Suppose we have obtained two admissible heuristics, h_1 and h_2, for the same problem
- We say h_2 *dominates* h_1 if for all n, $h_2(n) \geq h_1(n)$
- Then, A* with h_2 will never expand more nodes than A* with h_1 (modulo tie breaking)

3. Comparing Heuristics

- A* expands all nodes such that $f(n) < C_{opt}$
  - i.e. all nodes such that $h(n) < C_{opt} - g(n)$

- Since $h_1(n) \leq h_2(n)$, if $h_2(n) < C_{opt} - g(n)$, so is $h_1(n)$

2. Performance of a heuristic

- We can measure how effective a heuristic is by calculating an effective branching factor b

Nodes expanded → $N+1 = 1 + b + b^2 + \ldots + b^d$ ← Depth of Goal

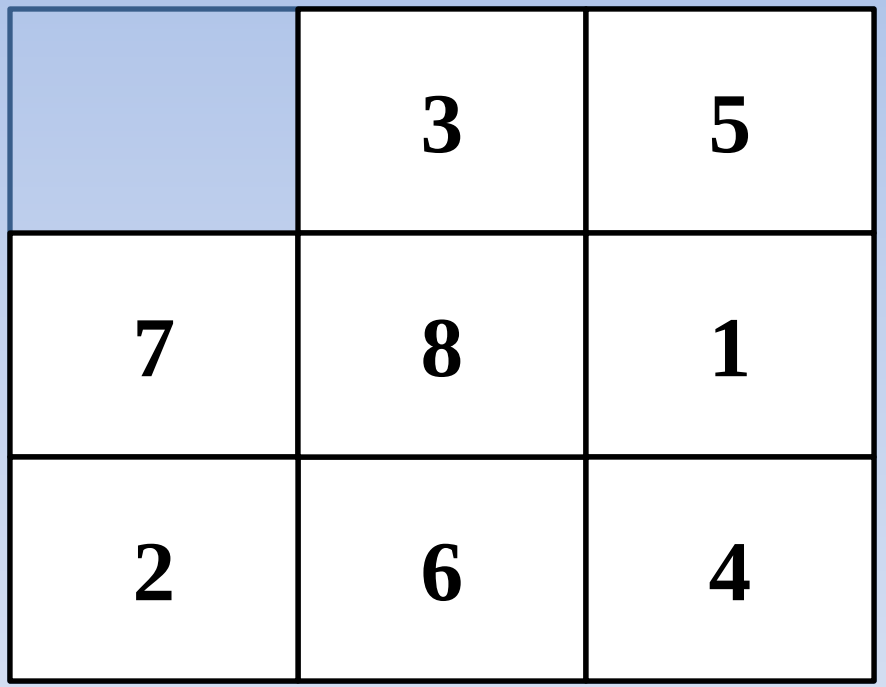
- This is the branching factor a balanced tree of depth d would have to contain $N+1$ nodes

- The better the heuristic, the closer to 1 this value will be

1. Designing a Heuristic Function

- How do we create an admissible heuristic?
  - Remember this means we just need to underestimate the true cost of a solution from the current node
- One way to do this is by relaxing the problem
  - Remove some constraints from the problem so the problem is easy to solve
  - Then the optimal solution for the relaxed problem becomes an admissible heuristic for the real problem
  - The straight line distance heuristic is like this

Example: 8-puzzle



Combining heuristic functions

- Suppose we come up with a bunch of different relaxations of a problem, and none of the heuristics dominate any other
- Construct the function

$$h(n)=max(h_1(n), h_2(n),\ldots h_k(n))$$

- This function dominates all the individual heuristics and is also admissible

Using subproblems

- If we solve a subproblem of the overall problem exactly, we can use its cost as the heuristic

- This is the idea behind pattern databases

Example: 8-puzzle

	3	*
*	*	1
2	*	4

Learning heuristics

- We could also have an agent try and learn a heuristic
- In this case, an agent would need to solve lots of search problems of a certain kind and record the actual costs at various states
- Then the agent would need to use machine learning to construct a heuristic function (later)

Advanced Search

- It is possible to use search as a solution strategy even for
  - Dynamic (*nearly static*) environments
  - Stochastic (*nearly deterministic*) environments

Summary

- We learned about:
  - Informed Search: Greedy and A*
  - Characteristics of A*
  - Designing Heuristic functions