Artificial Intelligence

CSC 665

tyler dae devlin

Search IV

2.15.2024

Recap

- **Search:** make decisions by looking ahead
- **Logic:** deduce new facts from existing facts
- **Constraints:** find a way to satisfy a given specification
- **Probability:** reason quantitatively about uncertainty
- **Learning:** make future predictions from past observations

Search

Modeling: start state, actions, costs, transition model, goal test

Inference:

- Uninformed: backtracking, DFS, BFS, UCS
- Informed: greedy search and A* with heuristics via problem relaxation

A* Search

UCS

- Maintains a frontier of uniform PastCost
- **Correct** but **slow**.

Greedy search

- Chooses the node that minimizes h
- **Incorrect** but **potentially fast**.

A*

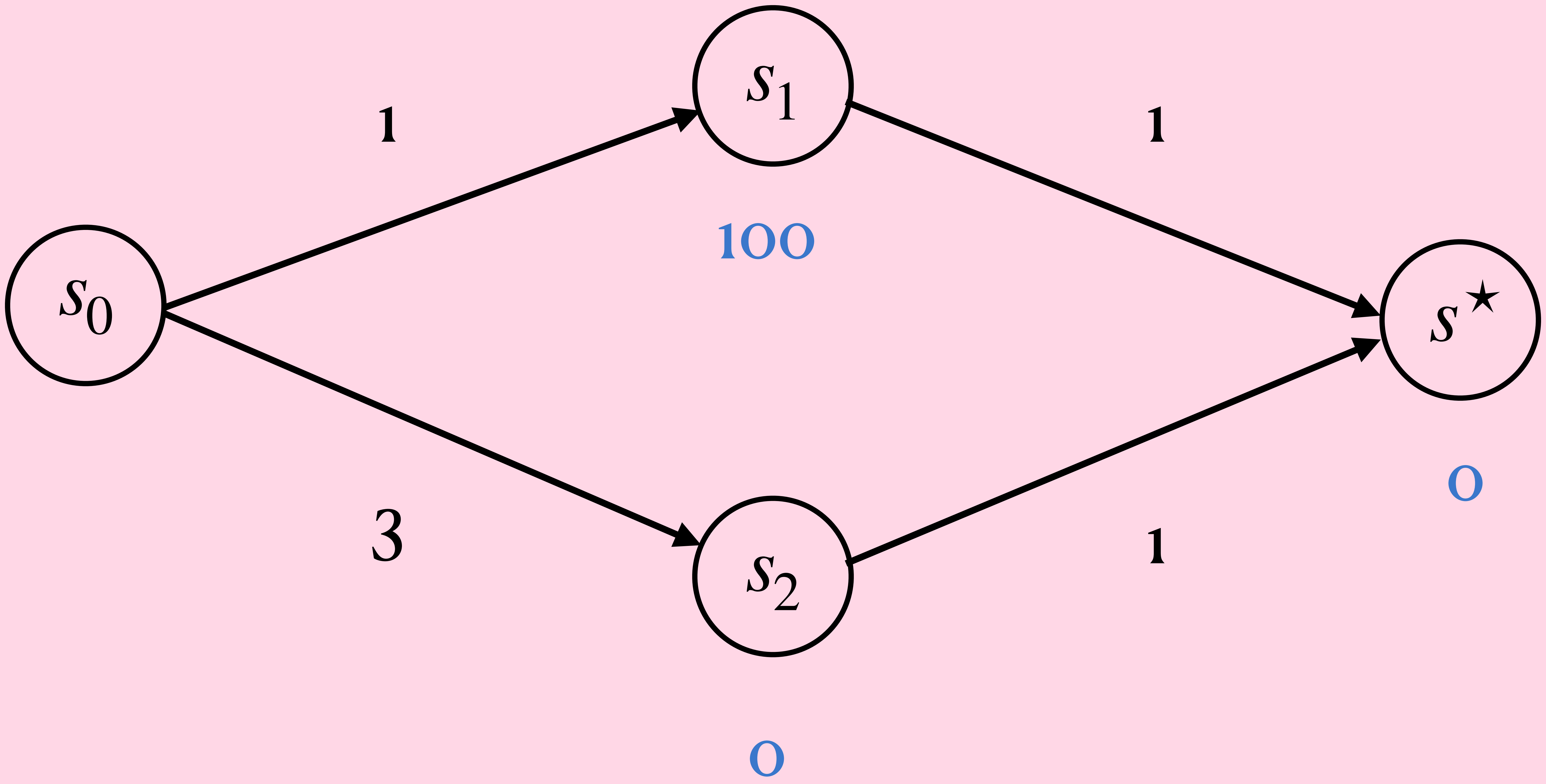
- Maintains a frontier of uniform PastCost + h
- **Sometimes correct** and **potentially fast**.

A* vs. Greedy

Problem: short-term greediness can get you into long-term trouble (true for all greedy algorithms in computer science and in life).

Key insight: computing PastCost is easy (just accumulate edge costs), and helps us realize when a prior greedy decision has led us astray.

A* can be wrong



Action costs

h

When is A* correct?

Definition: A heuristic is *admissible* if it **never overestimates** the cost to the goal. That is, $h(s) \leq \text{FutureCost}(s)$ for every $s \in S$.

Theorem: A* with heuristic function h is correct if h is admissible.

When is A* correct?

Proof: For contradiction, assume A* returns a path with cost C, but the optimal path has cost $C^\star < C$. Then there is a node s on the optimal path that was not expanded by A*. Focusing on this node,

$$\begin{aligned} C &< \text{PastCost}(s) + h(s) \\ &\leq \text{PastCost}(s) + \text{FutureCost}(s) \\ &= C^\star \end{aligned}$$

This is a contradiction. Thus, A* returns an optimal path.

How fast is A*?

Theorem: A* explores all states s satisfying $\text{PastCost}(s) \leq \text{PastCost}(s^\star) - h(s)$.

Proof: A* explores all states s satisfying $\text{PastCost}(s) + h(s) \leq \text{PastCost}(s^\star)$

Takeaway: Want h to be as large as possible, because this means we explore fewer states. But can't be too large or we lose admissibility (and thus correctness)!

Problem Relaxation

- How to choose h?
- Create a **"relaxed"** version of the problem by **removing constraints**.
- Set the **estimate** h in the original problem to be the **exact** FutureCost in the relaxed problem.
- *Such a heuristic is guaranteed to be* ***admissible****.*
- **Example:** for mazes, remove the constraint that you can't travel through walls. Then FutureCost(s) is simply the Manhattan distance from s to $s^\star$.
- What is the relaxation for Google maps? for the Roomba?

Backtracking search (last time)

Global state: minimum cost path, set of explored ~~nodes~~

function search(s, path) :

- **if** IsEnd(s) :
  - update the minimum cost path
- **for each** action $a \in$ Actions(s) :
  - **if** Succ(s, a) hasn't been explored yet:
    - add ~~it~~ to the explored set
    - extend path with Succ(s, a) and Cost(s, a)
    - recurse: search(Succ(s, a), path)

Backtracking search (revised)

Global state: minimum cost path, set of explored **(node, cost)** pairs

function search(s, path) :

- **if** IsEnd(s) :
  - update the minimum cost path
- **for each** action $a \in$ Actions(s) :
  - **if** Succ(s, a) hasn't been explored at Cost(s, a) yet:
    - add (Succ(s, a), Cost(s, a)) to the explored set
    - extend path with Succ(s, a) and Cost(s, a)
    - recurse: search(Succ(s, a), path)

[live coding: backtracking search]

Adversarial Game-Playing

Connect Four®
The Classic Vertical Four-In-A-Row Game
AGES 7 and Up
MB® MILTON BRADLEY

Can we model Connect Four as a search problem?

[modeling attempt on board]

Need to make some changes…

Modeling a game

Start state: $s_0 \in S$

Possible actions: $\text{Actions}(s) \subseteq A$

Transition model: $\text{Succ}(s, a) \in S$

Goal test: $\text{IsEnd}(s) \in \{\text{True}, \text{False}\}$

Agent utility: $\text{Utility}(s) \in \mathbb{R}$

Whose turn: $\text{Player}(s) \in P$

state space S, action set A, player set P, real numbers $\mathbb{R}$

Example: chess

s_0 = starting chess board

Actions(s) = legal chess moves available to Player(s)

Succ(s, a) = board state resulting from taking action a

IsEnd(s) = whether s is a checkmate or stalemate

$$\text{Utility}(s) = \begin{cases} +\infty & \text{if white wins} \\ -\infty & \text{if black wins} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Player}(s) = \begin{cases} \text{white} & \text{if an even number of turns have passed} \\ \text{black} & \text{if an odd number of turns have passed} \end{cases}$$

Two key characteristics of games

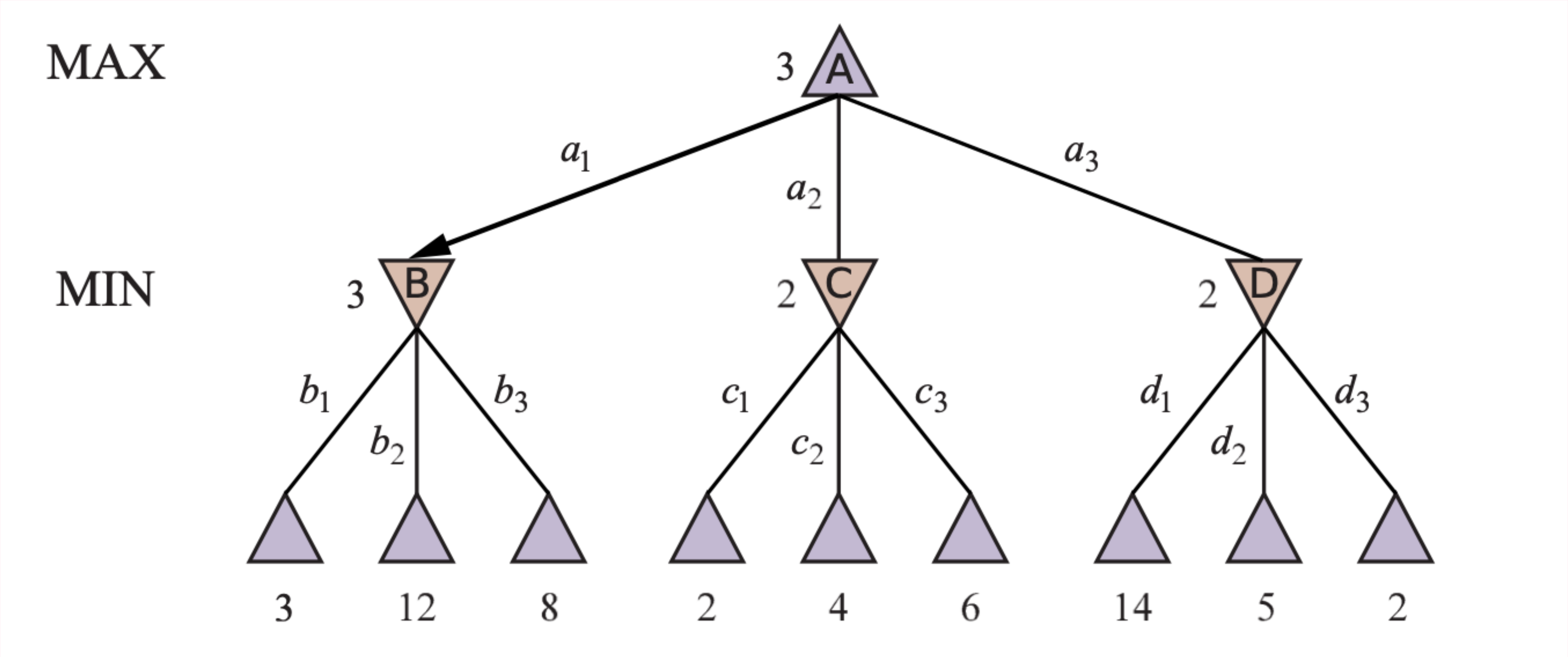
Different players in control at different nodes — one maximizing player and one minimizing player.

All **utility is concentrated at terminal nodes** (i.e. leaves in a tree) — don't know whether a move is good or bad until the game is over.

What should you do?

- Given a game state *s*, what action in Actions(*s*) should you take?
- Depends on who you are — assume you are the maximizing player, *max*
- *max*'s best action depends on what *min* does on the next turn
- But *min*'s best action depends on *max*'s move on the next next turn
- … which depends on *min*'s move on the next next next turn
- And so on …

[minimax game tree on board]


MAX
MIN
3 A
3 B
2 C
2 D
a_1
a_2
a_3
b_1
b_2
b_3
c_1
c_2
c_3
d_1
d_2
d_3
3
12
8
2
4
6
14
5
2