Timing Sheet

- 7: CSP Lecture
- 5: CSP Alone
- 5: CSP together
- 7: CSP Go Over

- 10: Game Tree lecture
- 6 Together (Or work in general)
- 7 Go Over Game Trees

New Thing!

I want to make the 10 minutes before section a quick OH for conceptual questions. Don’t be afraid to call me over and ask something!

CS 188: Artificial Intelligence

Discussion 2:
Constraint Satisfaction Problems and Game Trees

TA: Sherdil Niyaz

University of California, Berkeley

Administrivia

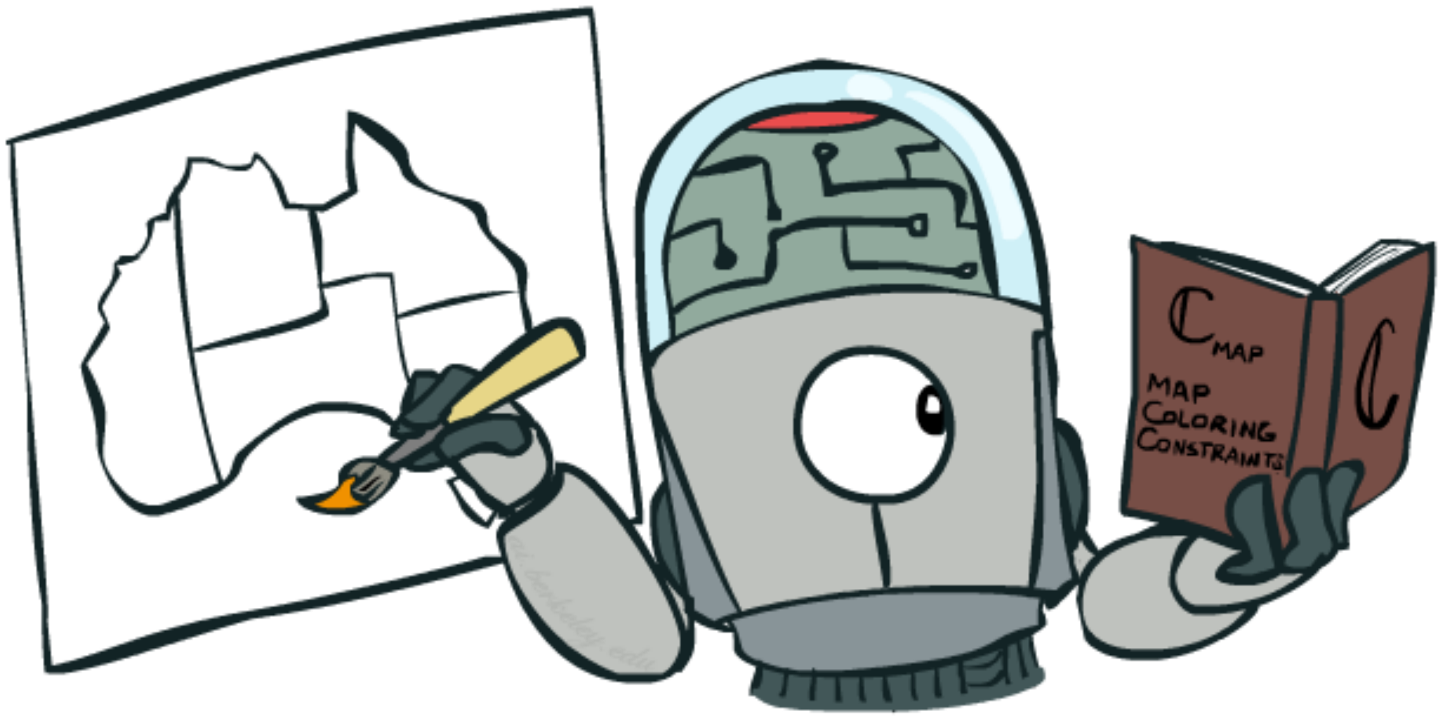
- Project 1: Search
  - Friday 9/16 at 5pm.
  - Start early and ask questions. It's longer than most! There are extra credit "reach" objectives that are tough to beat.
  - Solo or in group of two.
- Contest 1: Search
  - Due Sunday 9/18 at 11:59pm.
  - Solo or in group of two.
  - Friendly competition. Consider joining in, even if you're not the competitive type.
  - Extra credit opportunities if you beat the staff bots. Even more extra credit for top place finishers.

Apologies if I have to squash questions at some point!

I will also probably (unconsciously) start talking fast since we have so much material. Please yell at me to slow down if I talk too fast!

TODAY’S SECTION IS PACKED!

CSPs



CSPs

- Recall: Graph/Tree search
- Order mattered! Going A -> B -> C potentially different than C -> B -> A.

- CSPs are a special type of search problem where order of assignment **does not matter.**
- **Example:** I am scheduling classes. It does not matter whether I give CS61B a room before CS188, or CS188 a room before CS61B.

Backtracking

- With these problems where order doesn't matter, running DFS etc actually wastes lots of time.
- Alternative: Use the **backtracking algorithm**!

- Formulation: I have a bunch of **variables**, each with a **domain** of values I can assign to it.
- I also need to assign each variable with a value in its domain without violating any of the **constraints** that I'm given.

TLDR: Backtracking

- Keep assigning each variable a value in its domain that doesn’t violate the given constraints
- When you can no longer do this and you haven’t solved the CSP, you messed up!

- Time to **back track:** go to the last variable that you assigned a value, and give it a different one **if possible**. Try again.
- **Important:** May need to backtrack **multiple** times!

Backtracking Power-Ups

- **Common misconception from Office Hours:** Forward Checking, AC3, MRV, etc are **not** alternatives to Backtracking!
- Instead they are **power ups!** They are just extras that we strap onto backtracking to make it faster.

- **Forward checking**: When you assign a variable a value, kick out from its neighbors (aka variables that share a constraint with it) any conflicting values.
- **Arc Consistency (AC3)** is a stronger version of this.

Moar Backtracking Power-Ups

- **MRV:** When you pick the variable to assign next in backtracking, pick the one that has the minimum number of legal values left in its domain.

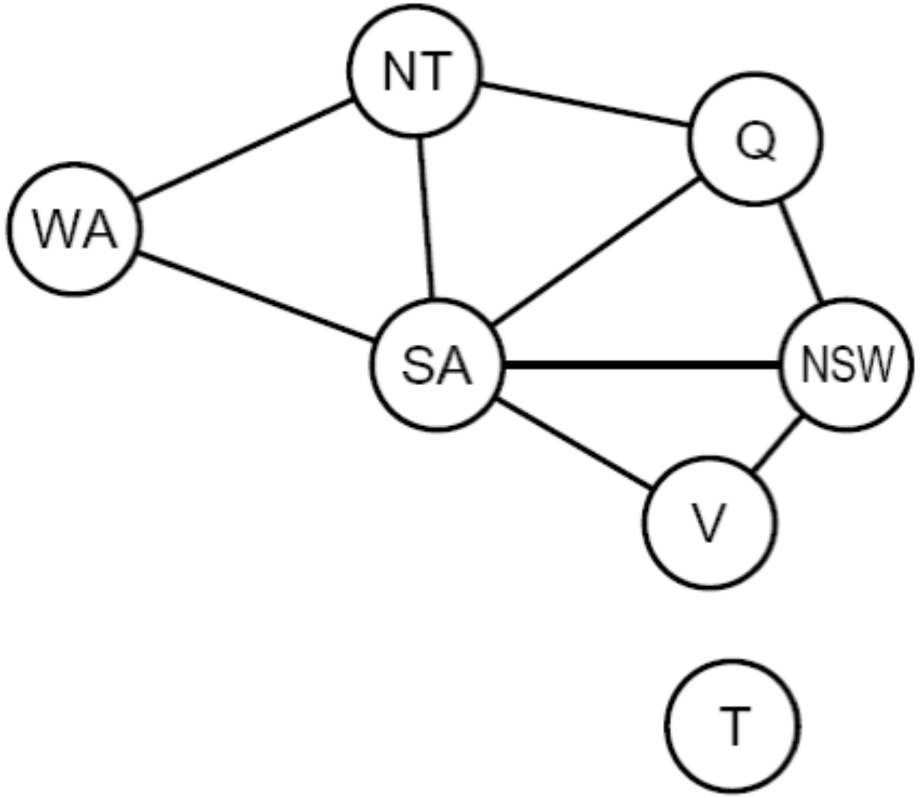
- **LCV:** Now that you've picked the next variable to assign, pick from its domain the **value** that would cause the **minimum** number of values to be ejected when forcing arc consistency.

3-Col Backtracking Animation!

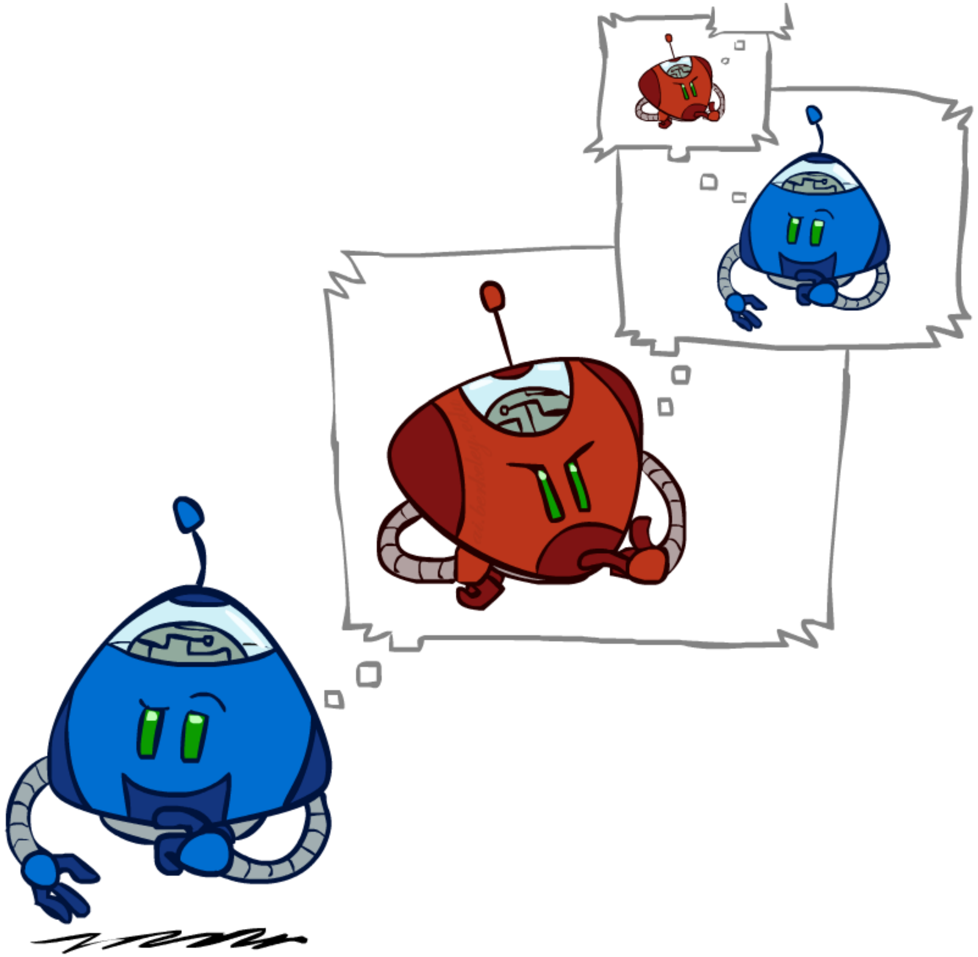
- This is on our section website, by the way!

  (http://sniyaz.weebly.com/cs188.html)

- Note that this animation shows the **constraint graph**, where two variables share an edge if they are bound by a binary constraint.



Game Trees



Game Trees

- **Notice:** In each of the search algorithms we've seen so far, there's only been **one** agent! (i.e. we've had Pacman, but no ghosts!)

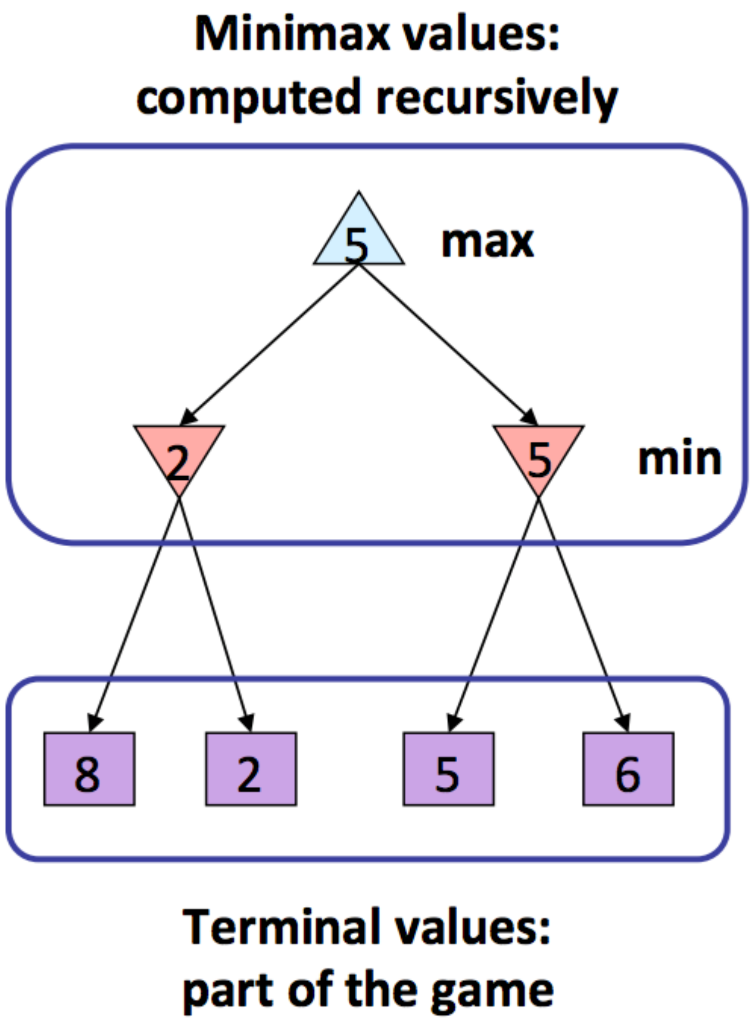
- How do we handle **two** agents playing against eachother?

- Let's assume that we have two players A and B playing a game **optimally**. A wants to maximize the score of the game, and B wants to minimize it.

Mini-Max Algorithm

- Let's assume that we have two players A and B playing a game **optimally**. A wants to maximize the score of the game, and B wants to minimize it.
- We go from the **bottom** of the game tree up, and see which actions that each agent would take at each level.
- Remember, each branch in the game tree represents a different set of moves that might happen.

Example

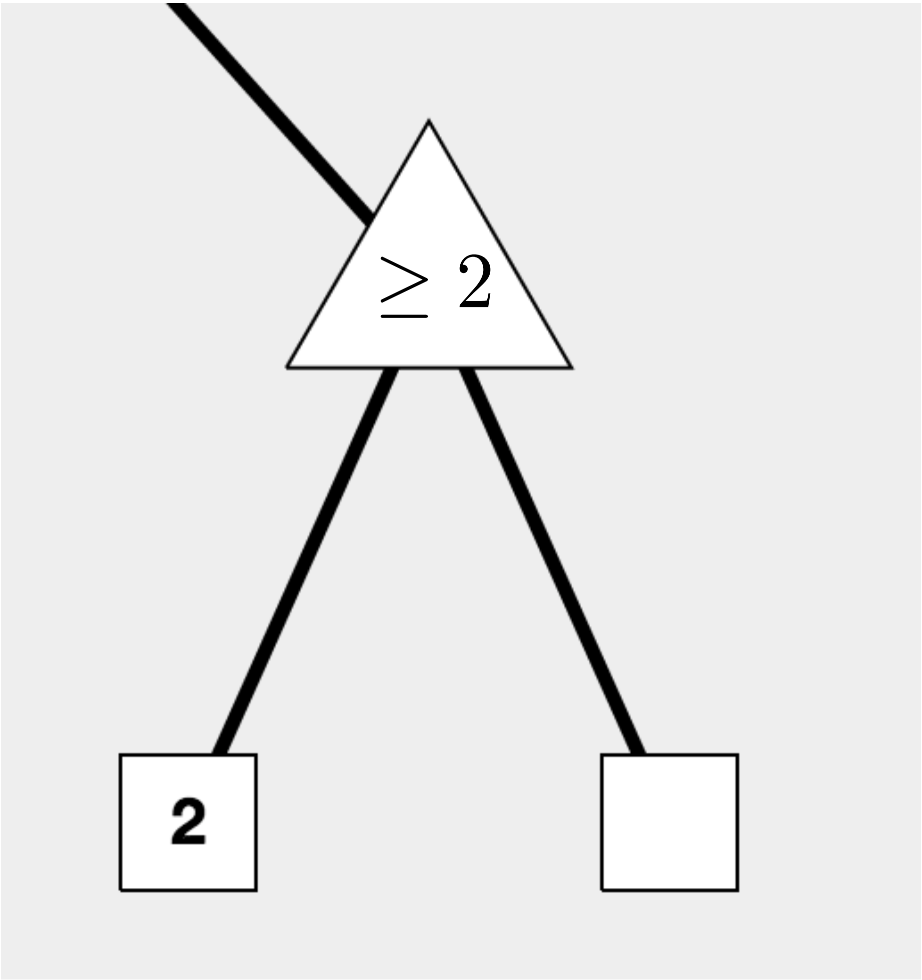

Minimax values:
computed recursively
5
max
2
5
min
8
2
5
6
Terminal values:
part of the game

Pruning

- Sometimes certain parts of the game tree would never be reached since one of the players wouldn't allow it!
- When we know this is the case, we can stop computing on that branch and save time!
- Result: same answer, but faster runtime for Mini-Max

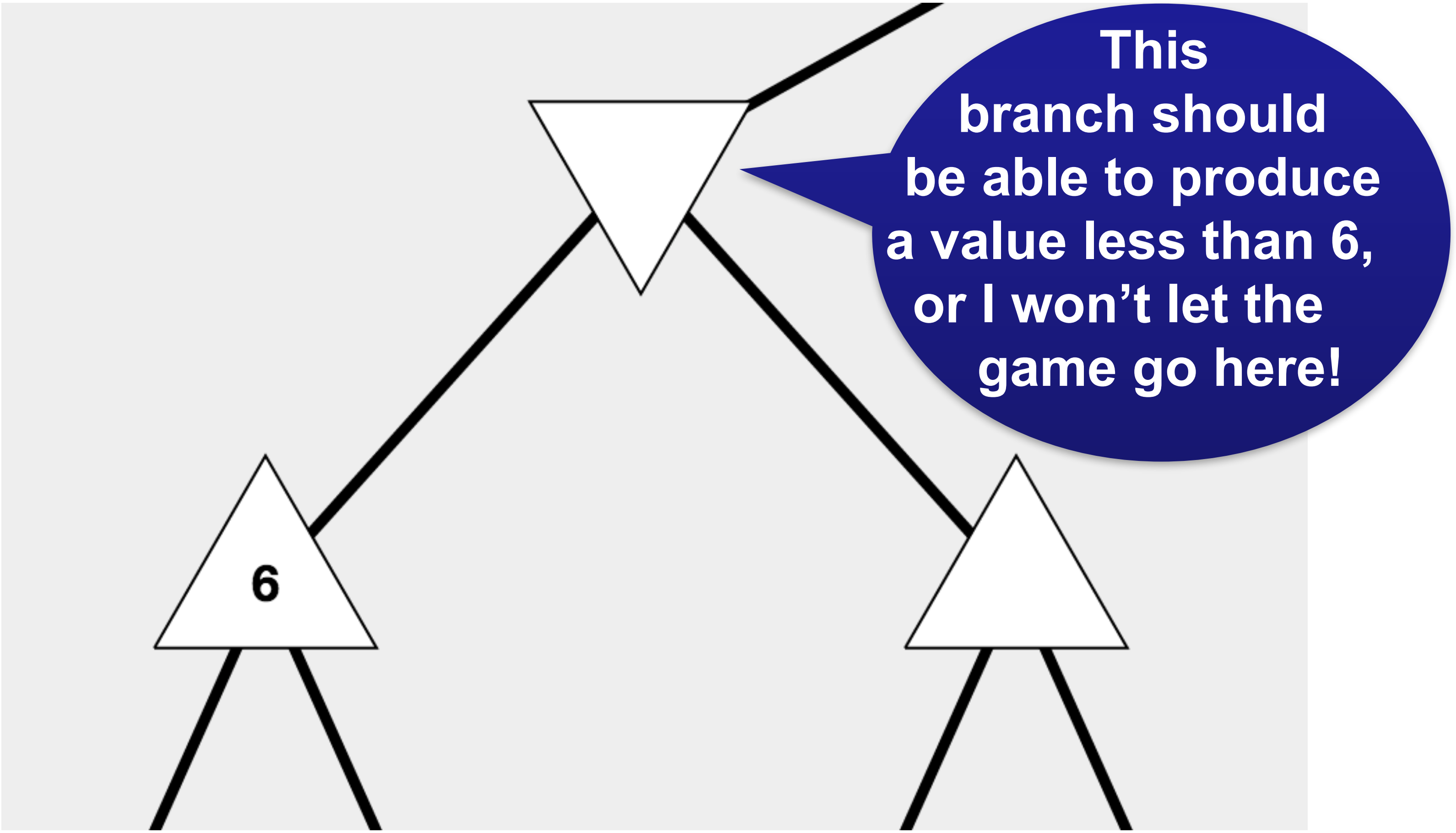
Warm-Up 1: Bounds

- When a maximizer finds a value (2, in this case) the final value returned by the minimizer will be **at least** that value. (Since any choices that are smaller will get rejected).
- This is symmetric for a minimizer.



Warm-Up 2: Allowing Branches

- If a minimizer has already found a value, they will only let the game proceed down branches that can find smaller values than the found one.
- This is symmetric for a maximizer.



Pruning: A “cutoff” approach

- Let’s combine the previous two ideas!

- **Alphas and betas are confusing :/**
  I prefer to think in terms of cutoffs: If my “parent” node has found an option already, and I will only find things they like **less**, then why bother?

- Reasoning: they’ll never let me even get to this point, since they play optimally. They can just force me to the option that’s better for them.

An example!

- This is confusing! Show me an example?

- http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

  (Also linked from section site)

- This animation is awesome! Use it to generate infinite practice problems!