Artificial Intelligence

CSC 665

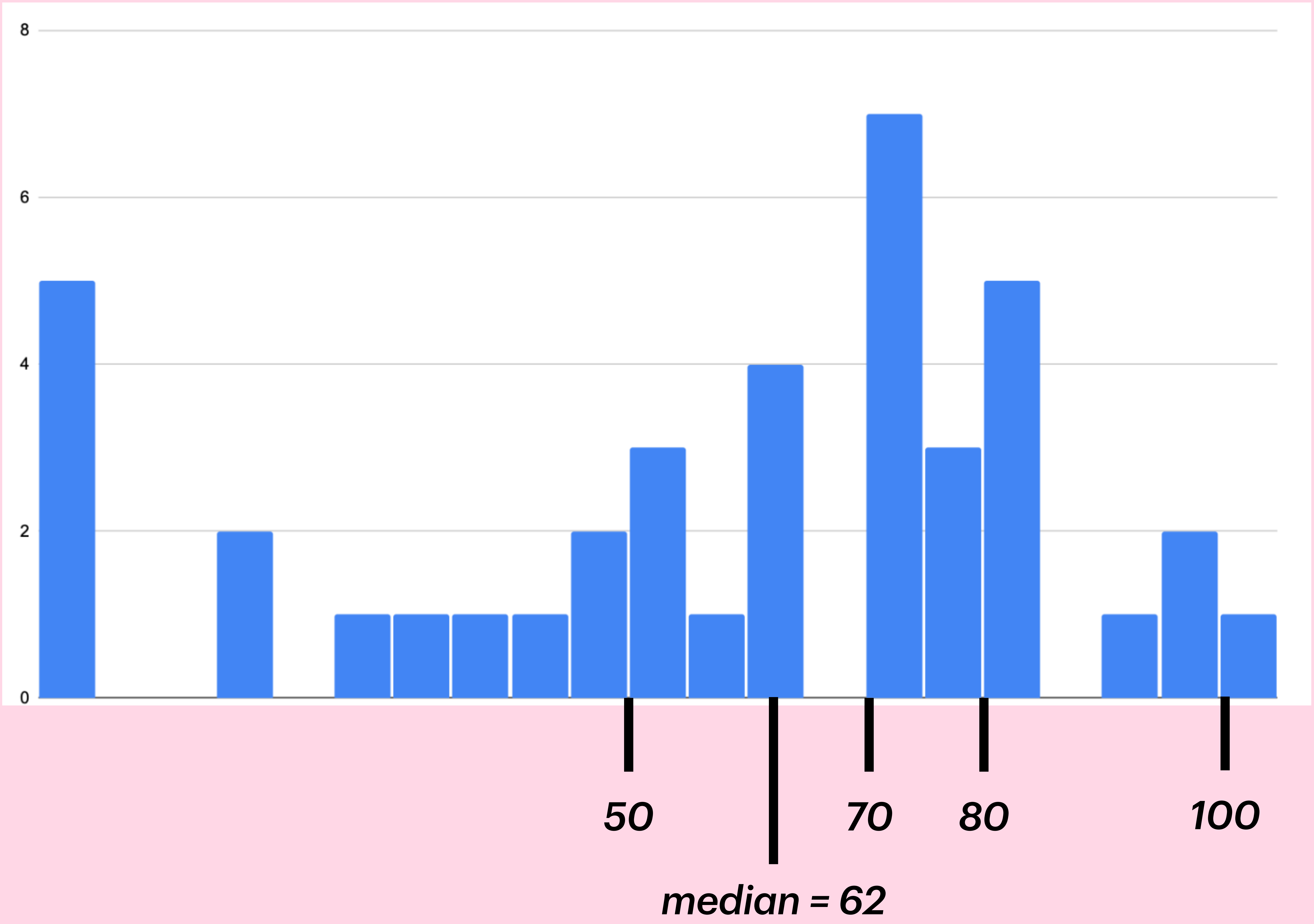
tyler dae devlin

CSPs II

9.26.2023

Administrivia

- Homework 1 graded
- See Canvas announcement for grading details and study advice
- First midterm next Tuesday 10/3
- See Canvas announcement for details and logistics
- Cheat sheet allowed!

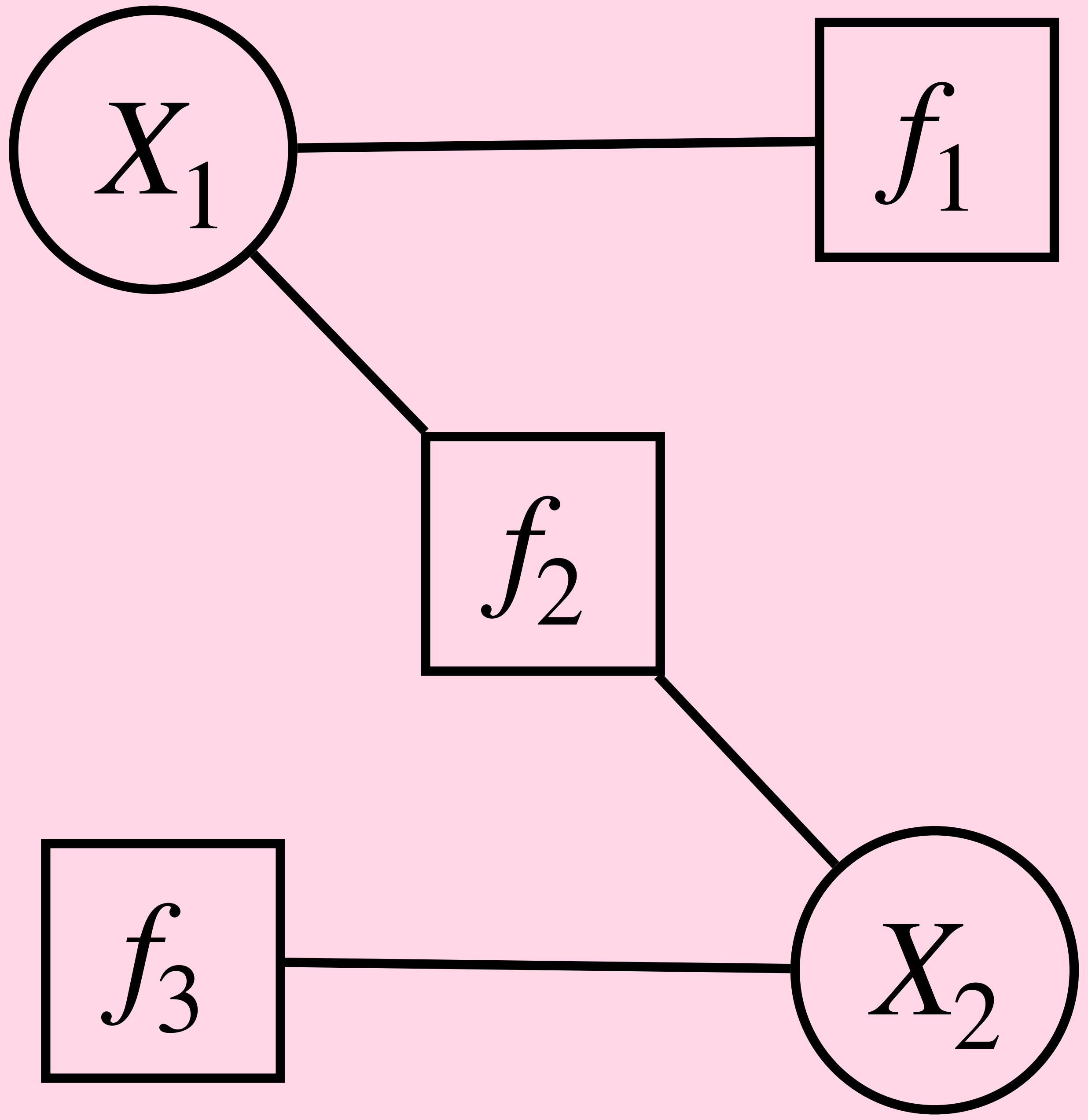


- **Search:** make decisions by looking ahead
- **Logic:** deduce new facts from existing facts
- **Constraints:** find a way to satisfy a given specification
- **Probability:** reason quantitatively about uncertainty
- **Learning:** make future predictions from past observations

New representation: factor graphs

A **factor graph** consists of

- Variables $X = (X_1, X_2, \ldots, X_n)$
- Domains $D = (D_1, D_2, \ldots, D_n)$, where $X_i \in D_i$
- Factors $f_1, f_2, \ldots, f_m$ where $f_j(X) \geq 0$



Assignment weight

The assignment of values $x = (x_1, \dots, x_n)$ to variables $X = (X_1, \dots, X_n)$ has **weight**

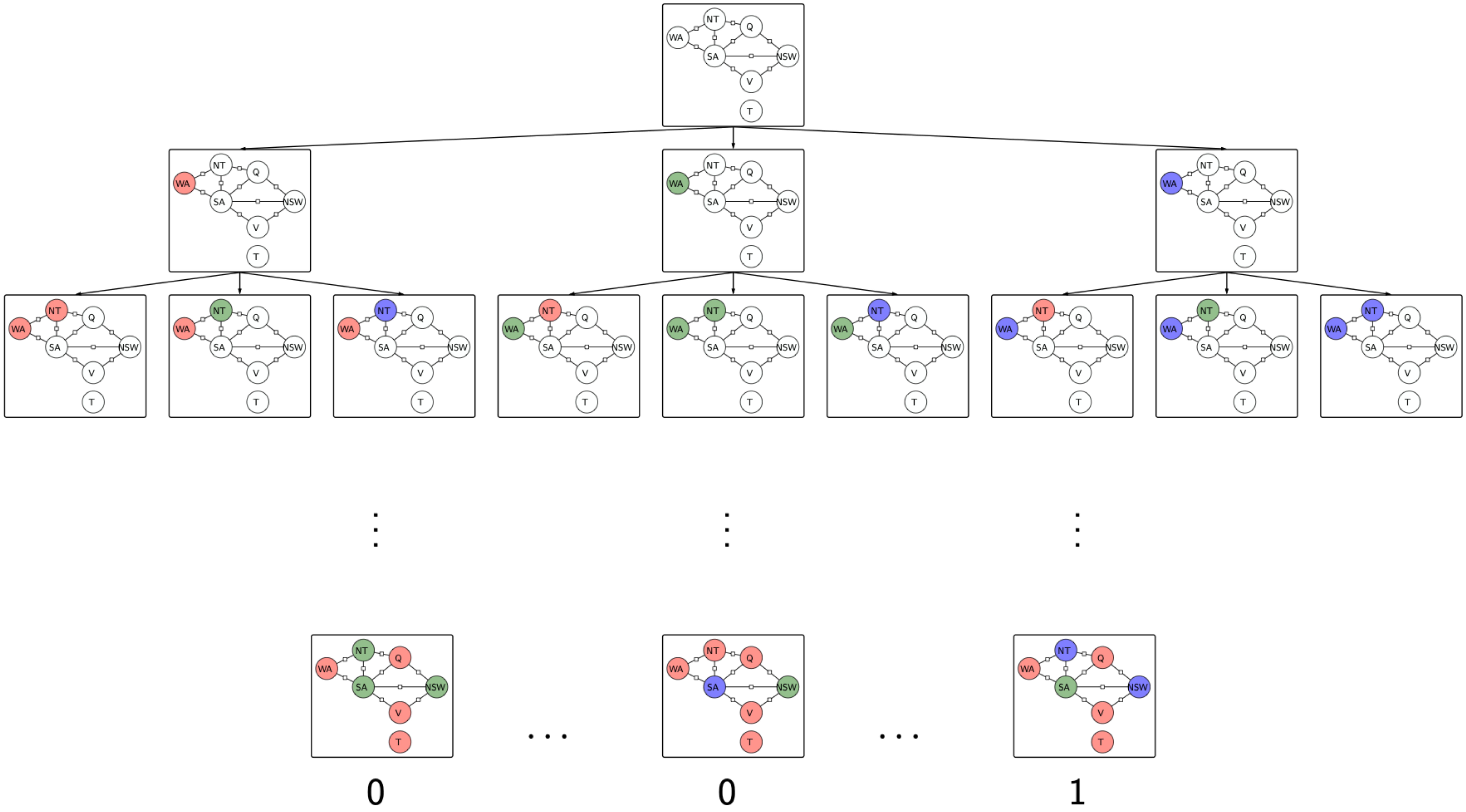
$$\begin{aligned} W(x) &= f_1(x) f_2(x) \dots f_m(x) \\ &= \prod_{j=1}^{m} f_j(x) \end{aligned}$$

As assignment x is **consistent** if $W(x) > 0$

If multiple assignments are consistent, want to find the **highest weight** assignment

A CSP is **satisfiable** if there is some consistent assignment

Inference


WA
NT
Q
SA
NSW
V
T
0
0
1

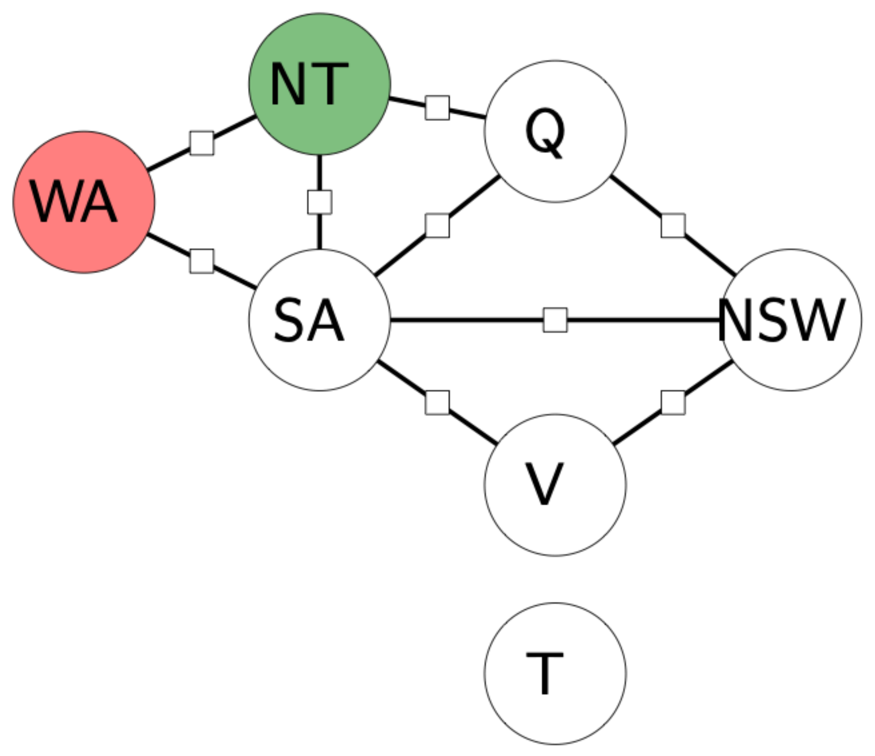
Partial assignments

Current assignment is $x = \{\text{WA} : R, \text{NT} : G\}$

Weight is $w = 1$

How do I update the weight after making an assignment to

- Queensland?
- South Australia?



Hard vs. soft constraints

- A **hard constraint** is a factor whose value is either 0 or 1. Encodes satisfiability.
- A **soft constraint** is a factor that can take on positive values other than 1. Encodes preference orderings.

Backtracking search for hard constraints

function backtrack(x, D) :

- **if** x is a complete assignment: **return** x
- choose unassigned variable X_i
- pick an order for the values of X_i's domain D_i
- **for** each value v in that order:
  - **if** $x \cup \{X_i : v\}$ is consistent:
    - $D' \leftarrow$ pruned domains via lookahead
    - **if** any domain in D' is empty: **continue**
    - $x^\star \leftarrow$ backtrack($x \cup \{X_i : v\}, D'$)
    - **if** $x^\star$ is not None: **return** $x^\star$
- **return** None

Backtracking search for soft constraints

function backtrack(x, w, D) :

- **if** x is a complete assignment: update best and **return**
- choose unassigned variable X_i
- pick an order for the values of X_i's domain D_i
- **for** each value v in that order:
  - $\delta \leftarrow \prod_{f \in U} f(x \cup \{X_i : v\})$ where U is the set of factors that depend on X_i and x but not unassigned variables
  - **if** $\delta = 0$: **continue**
  - $D' \leftarrow$ pruned domains via lookahead
  - **if** any domain in D' is empty: **continue**
  - backtrack($x \cup \{X_i : v\}, w\delta, D'$)

Hard vs. soft backtracking

Backtracking with exclusively **hard constraints**

- Trying to find assignment with **nonzero** weight
- **Any solution** is as good as any other
- Can stop once we find a single solution
- Don't need to keep track of the weight value; if we haven't terminated it must not be zero

Backtracking with some **soft constraints**

- Trying to find **maximum** weight assignment
- Need to explore **all possible solutions** and compare their weights
- Need to maintain and update the weight of a partial assignment

Both versions of backtracking have three underspecified steps

Backtracking search: hard constraints

function backtrack(x, D) :

- **if** x is a complete assignment: **return** x
- choose unassigned variable X_i
- pick an order for the values of X_i's domain D_i
- **for** each value v in that order:
  - **if** $x \cup \{X_i : v\}$ is consistent:
    - $D' \leftarrow$ pruned domains via lookahead
    - **if** any domain in D' is empty: **continue**
    - $x^\star \leftarrow$ backtrack($x \cup \{X_i : v\}, D'$)
    - **if** $x^\star$ is not None: **return** $x^\star$
- **return** None

Backtracking search: soft constraints

function backtrack(x, w, D) :

- **if** x is a complete assignment: update best and **return**
- choose unassigned variable X_i
- pick an order for the values of X_i's domain D_i
- **for** each value v in that order:
  - $\delta \leftarrow \prod_{f \in U} f(x \cup \{X_i : v\})$ where U is the set of factors that depend on X_i and x but not unassigned variables
  - **if** $\delta = 0$: **continue**
  - $D' \leftarrow$ pruned domains via lookahead
  - **if** any domain in D' is empty: **continue**
  - backtrack($x \cup \{X_i : v\}, w\delta, D'$)

Design choices

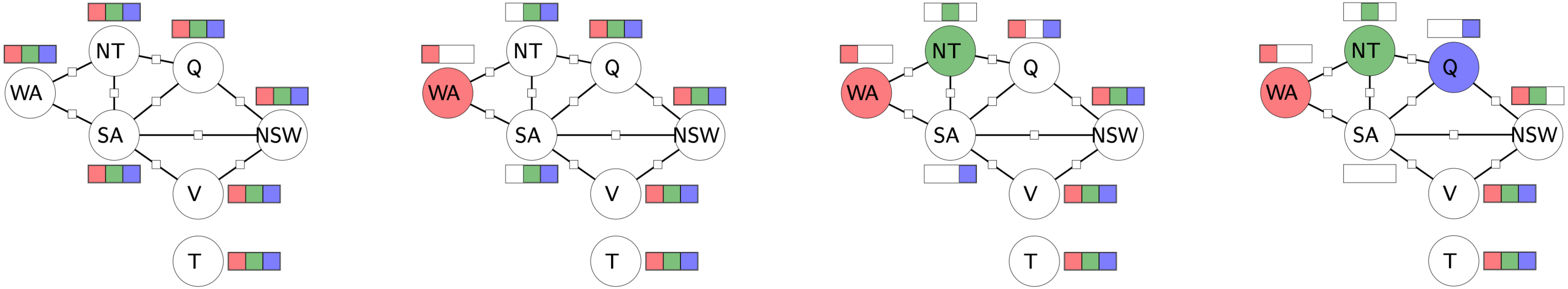
1. How to choose which variable to work on next?
2. How to order the values of that variable?
3. How to prune the domains?

Design choices

1. How to choose which variable to work on next? *Most constrained variable heuristic*
2. How to order the values of that variable? *Least constrained value heuristic*
3. How to prune the domains? *Lookahead (forward checking or AC-3)*

Lookahead: forward checking

- After assigning a variable X_i, eliminate inconsistent values from the domains of X_i's neighbors
- If any domain becomes empty, return (no possible solution)



Arc consistency

- A variable X_i is arc consistent with respect to X_j if for each $x_i \in D_i$ there exists $x_j \in D_j$ such that $f(x_i, x_j) \neq 0$ for all binary factors f on X_i and X_j.
- Enforcing arc consistency on X_i w.r.t. X_j means removing values from D_i until X_i is arc-consistent w.r.t. X_j
- *Forward checking enforces arc consistency on the neighbors of X_i after assigning a value to X_i*

AC-3

- $Q \leftarrow \{X_j\}$
- **while** Q is nonempty:
  - pop some X_j off of Q
  - **for** every neighbor X_i of X_j :
    - enforce arc consistency on X_i w.r.t. X_j
    - **if** D_i changed: add X_i to Q

AC-3 on Australia