# COSC76/276 Artificial Intelligence
# Fall 2022
# Adversarial search

Soroush Vosoughi
Computer Science
Dartmouth College
Soroush@Dartmouth.edu

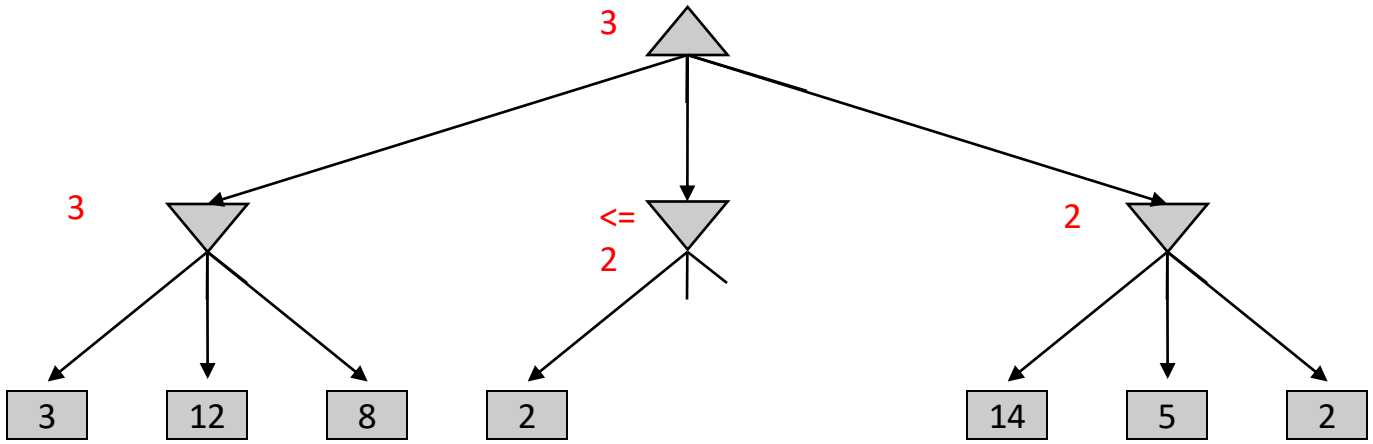
# Reminders

- PA-2 due today at 11:59pm ET

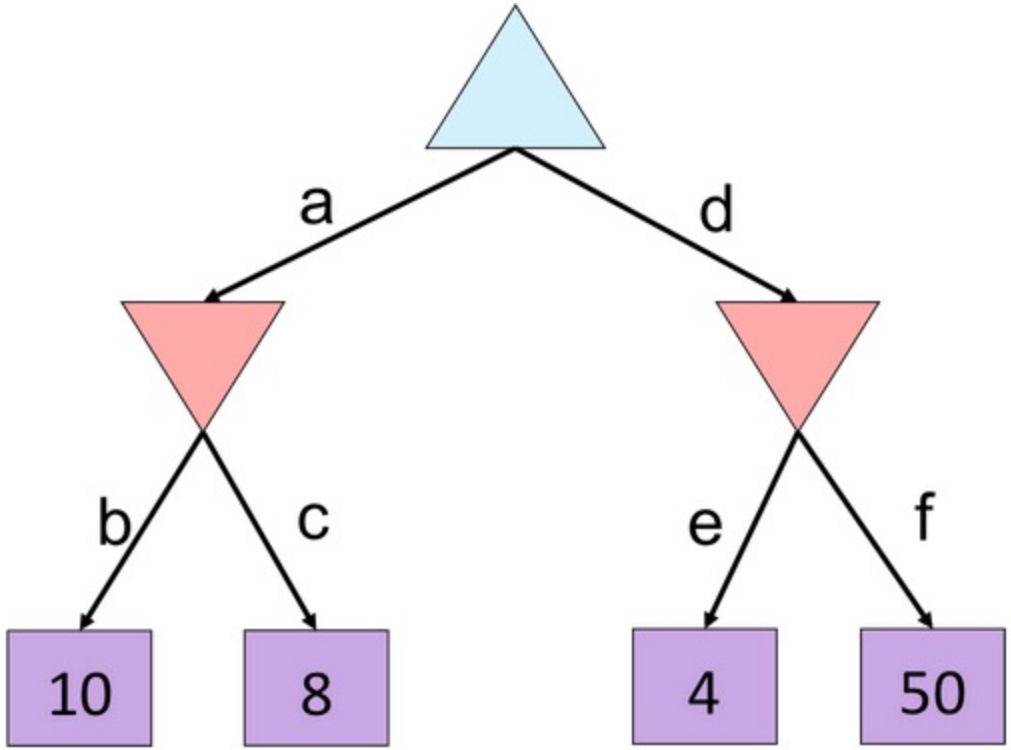
# Recap: Adversarial search and minimax

- Techniques to make the adversarial search problem more tractable: alpha-beta pruning

# Alpha-beta pruning intuition example

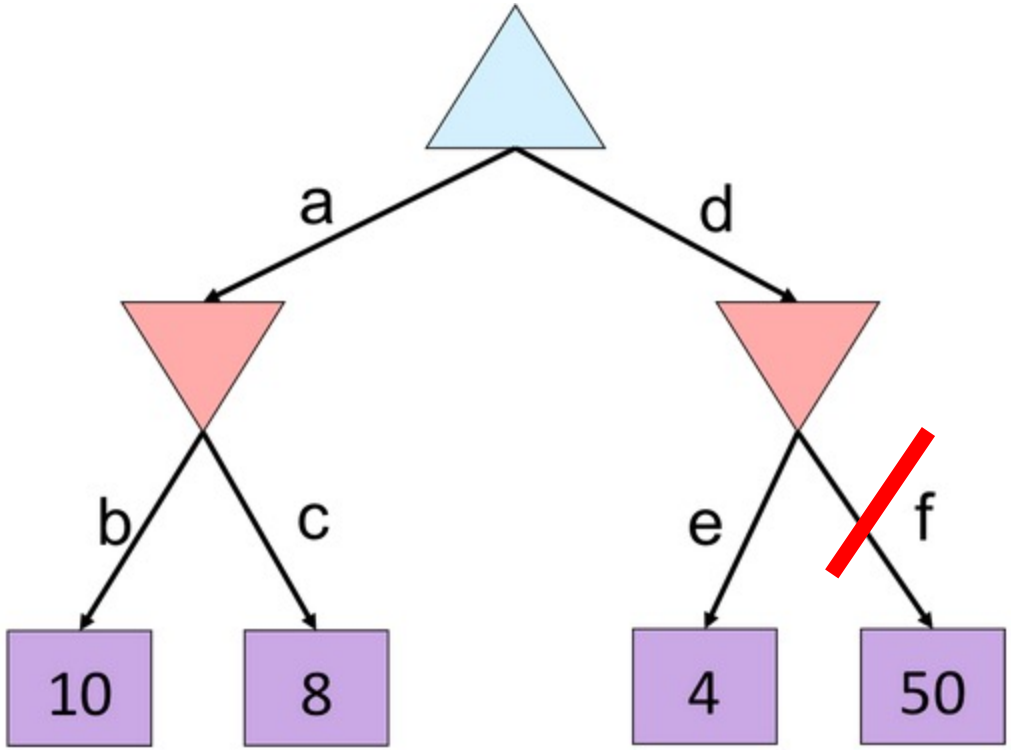


# Alpha-Beta Quiz




Anything that can be pruned?

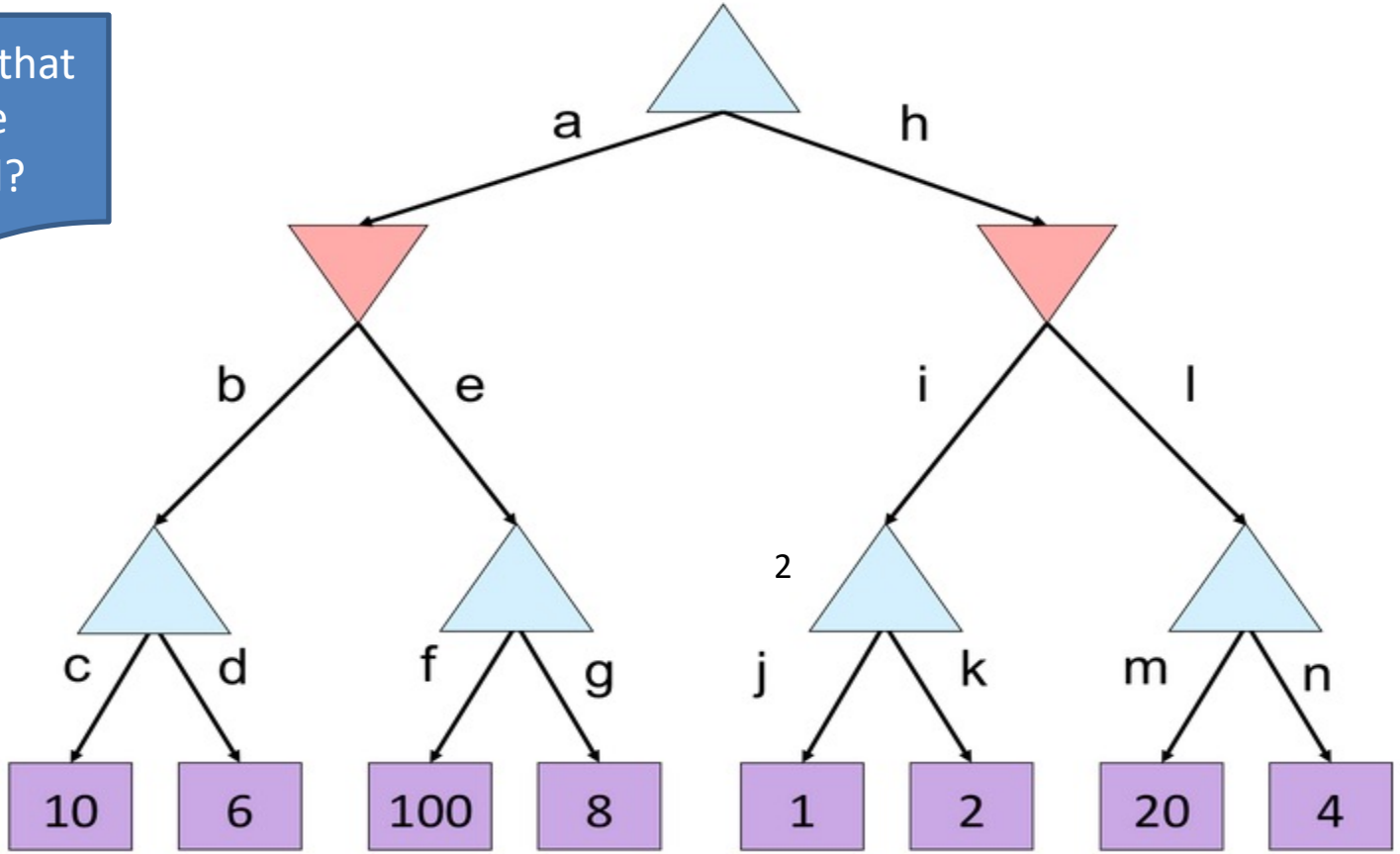
# Alpha-Beta Quiz




Anything that can be pruned?

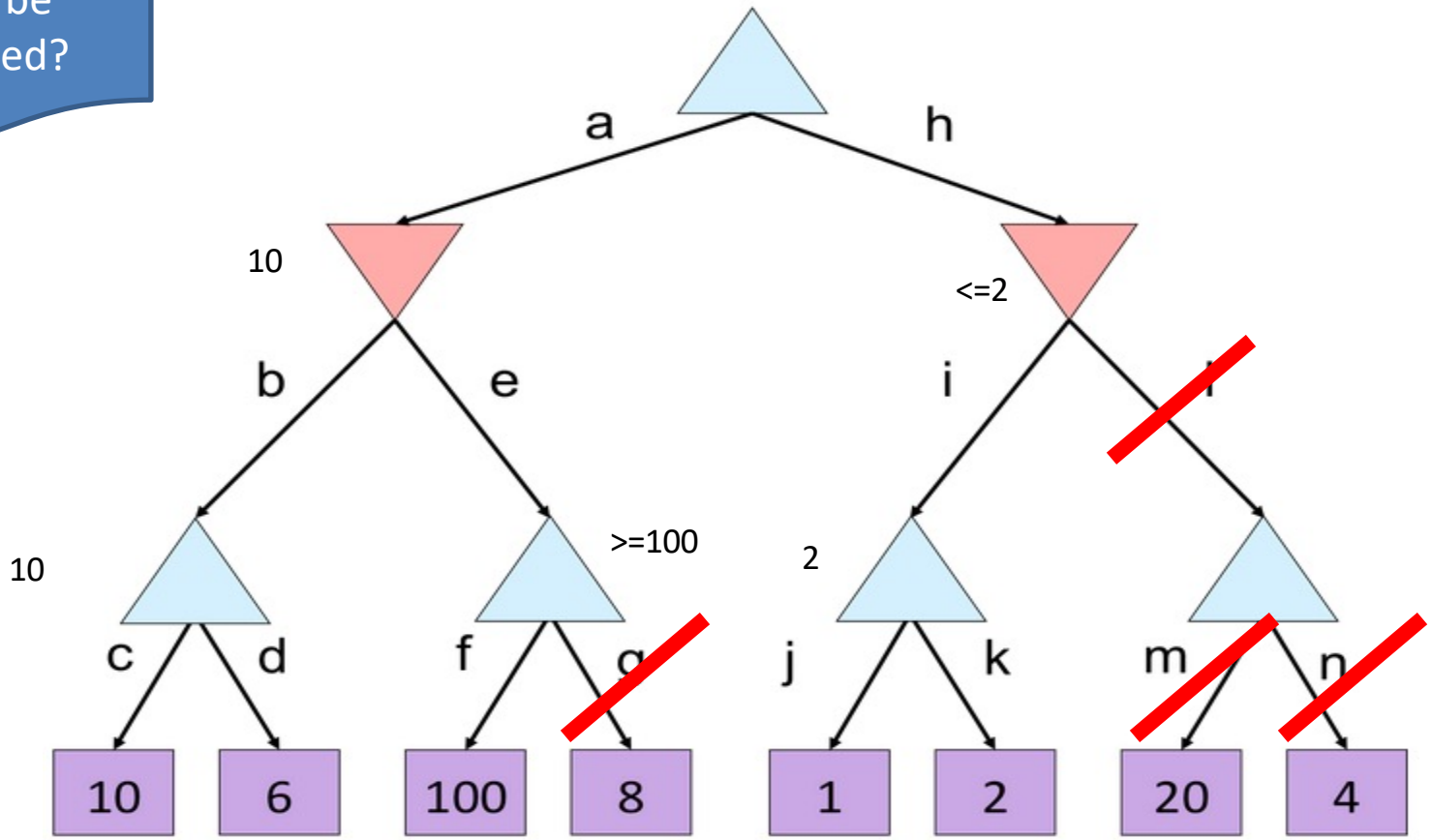
# Alpha-Beta Quiz 2

Anything that can be pruned?



# Alpha-Beta Quiz 2

Anything that can be pruned?

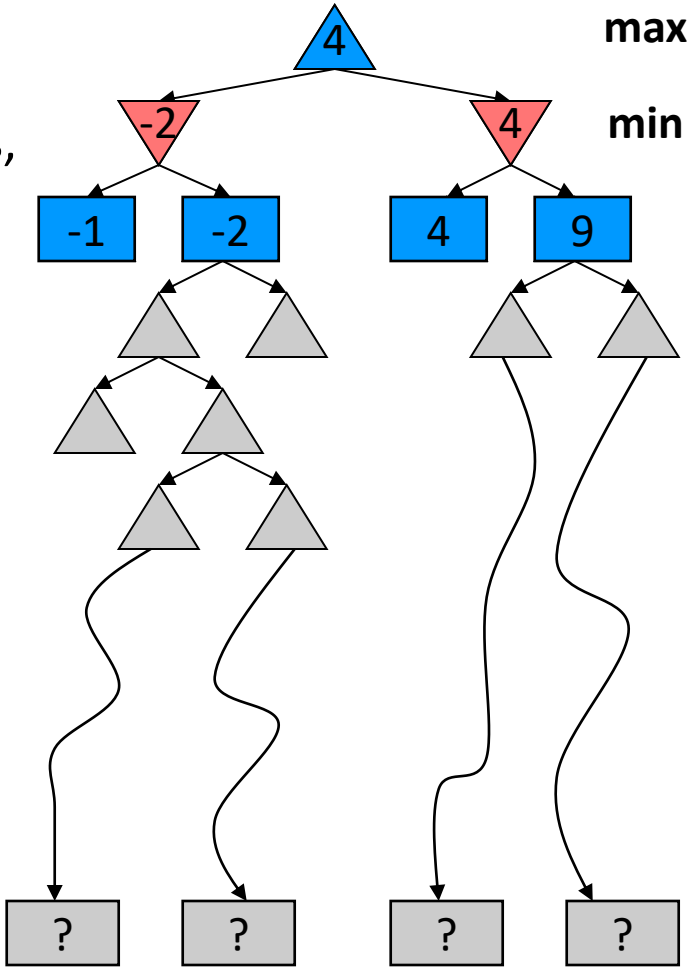


# Alpha-beta pruning Properties

- This pruning has no effect on minimax value computed for the root
- Good child ordering improves effectiveness of pruning
- With “perfect ordering” Time complexity drops to $O(b^{m/2})$

# Resource limits

- Problem
  - In realistic games, cannot search to leaves, as decisions need to happen **real-time**
- Solution
  - Cut-off test instead of terminal test
    - E.g., depth-limited search, iterative deepening
  - Replace terminal utilities with an evaluation function for non-terminal positions to estimate desirability of position
- Consequence
  - Guarantee of optimal play is gone



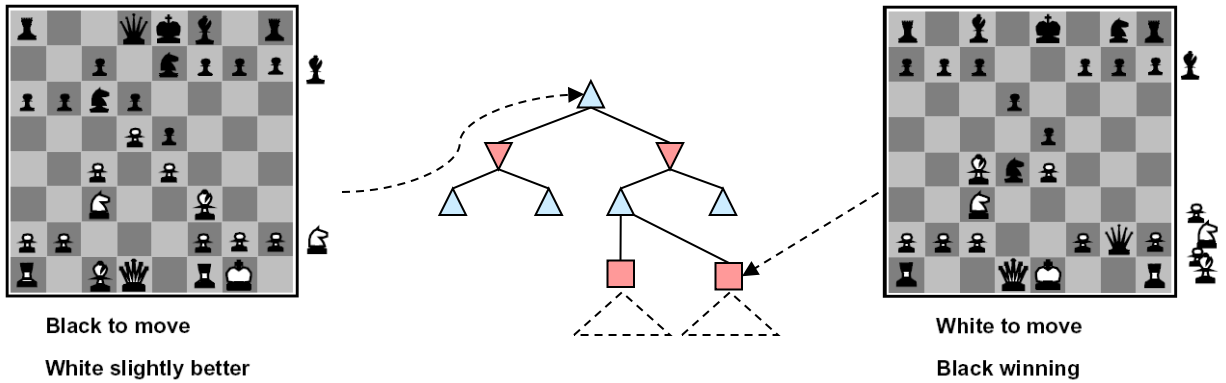
# Cut-off search

- In the recursive call, we need to have bookkeeping of the depth

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

# Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search




- Ideal function: returns the actual minimax value of the position
- In practice, typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

# When to have the cut-off?

- **Quiescent states**, i.e., states that will not have a large change in value, are good candidates for cutoff
  - Quiescence search

- **Horizon effects** are a problem: stalling tactics can push bad states beyond the depth searched

# Repeated states

- In games, repeated states occur frequently because of transpositions – i.e., different permutations of the move sequence end up in the same position
  - e.g., [a1, b1, a2, b2] vs. [a1, b2, a2, b1]
- It's worthwhile to store the evaluation of this position in a hash table – **transposition table** – the first time it is encountered
  - similar to the "explored set" in graph-search
- Tradeoff:
  - Transposition table can be too big
  - Which to keep and which to discard

# PA-3: chess game

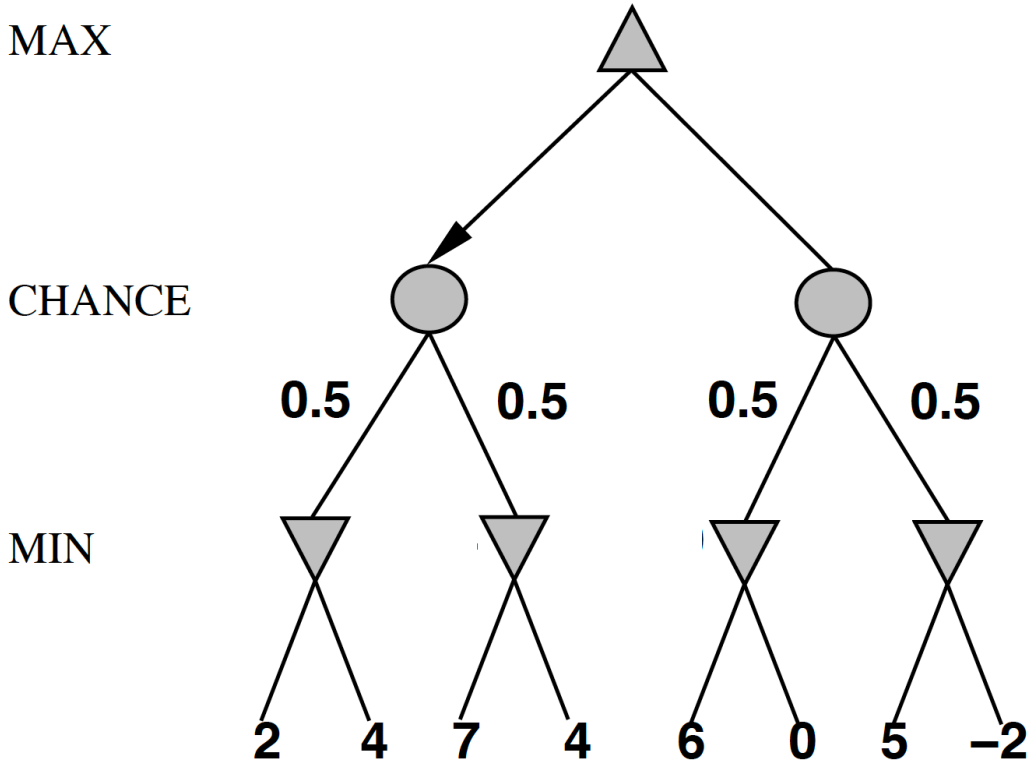
# Summary

- Alpha-beta pruning
  - Alpha: MAX's best option on path to root
  - Beta: MIN's best option on path to root
  - Prune part of the tree that won't be played given the current values of alpha and beta
- Real-time decisions
  - Terminal-test -> Cut-off test
  - Evaluation function to estimate desirability of a position

# Games

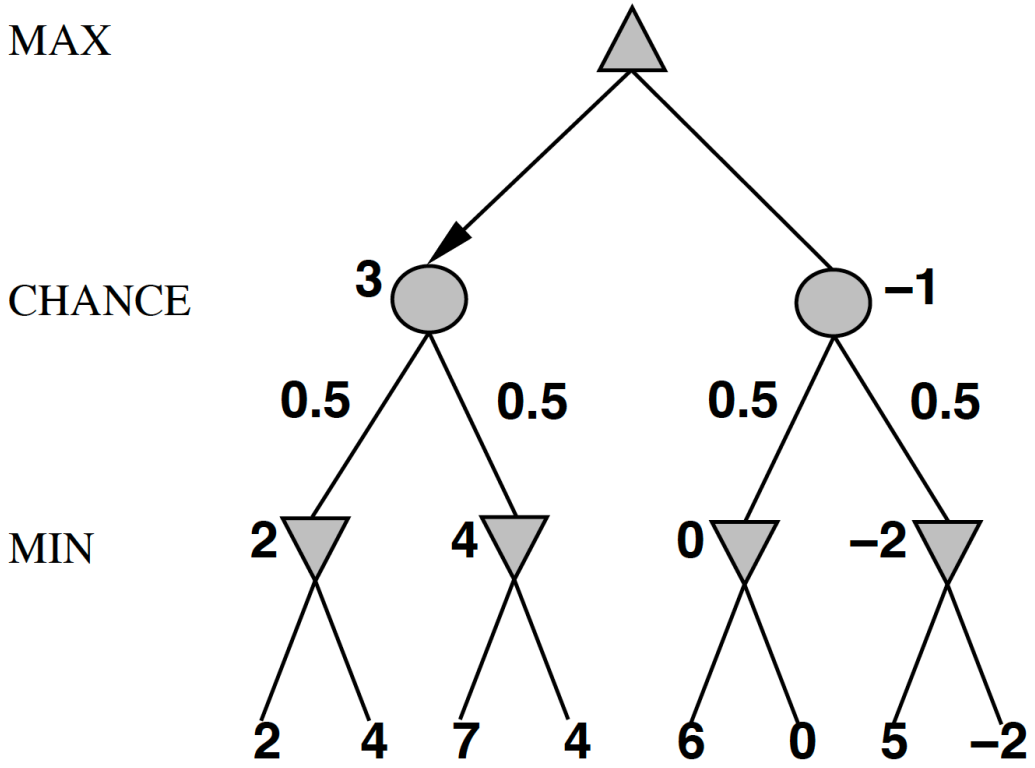
- Classified over different axes:
  - Deterministic or stochastic
  - Perfect or imperfect information
  - Number of players
  - Zero sum

# Stochastic games




In stochastic games, uncertain outcomes controlled by chance, not an adversary (e.g., dice, card shuffling, unpredictable opponents)

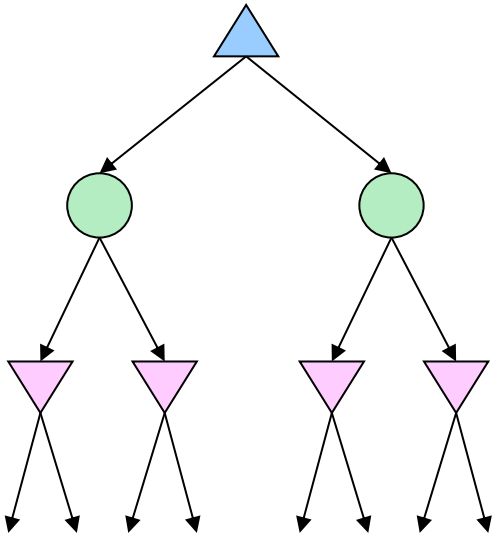
# Why not minimax?




- Worst case reasoning is too conservative
- Need average case reasoning

# Expectiminimax Search

- In stochastic games, values should reflect average-case (expectiminimax) outcomes, not worst-case (minimax) outcomes
- Expectiminimax search: compute the average score under optimal play
  - Max and min nodes as in minimax search
  - Chance nodes are like another player with "actions" with associated probabilities
  - Calculate their expected utilities, i.e. weighted average (expectation) of children

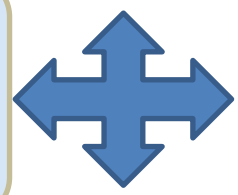


# Expectiminimax

$$\textsc{Expectiminimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

```
def value(state):
        if the state is a terminal state: return the state's utility
        if the next agent is MAX: return max-value(state)
        if the next agent is MIN: return min-value(state)
        if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
        initialize v = -∞
        for each successor of state:
                v = max(v, value(successor))
        return v
```

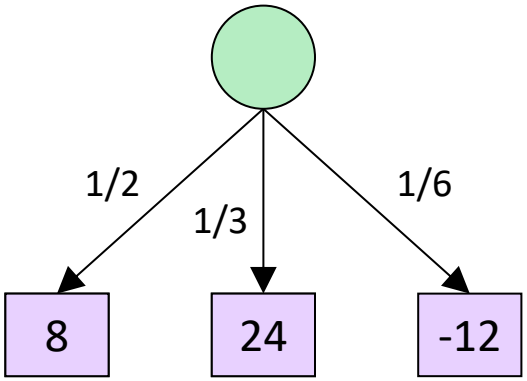


```
def min-value(state):
        initialize v = +∞
        for each successor of state:
                v = min(v, value(successor))
        return v
```

```
def exp-value(state):
        initialize v = 0
        for each successor of state:
                p = probability(successor)
                v += p * value(successor)
        return v
```

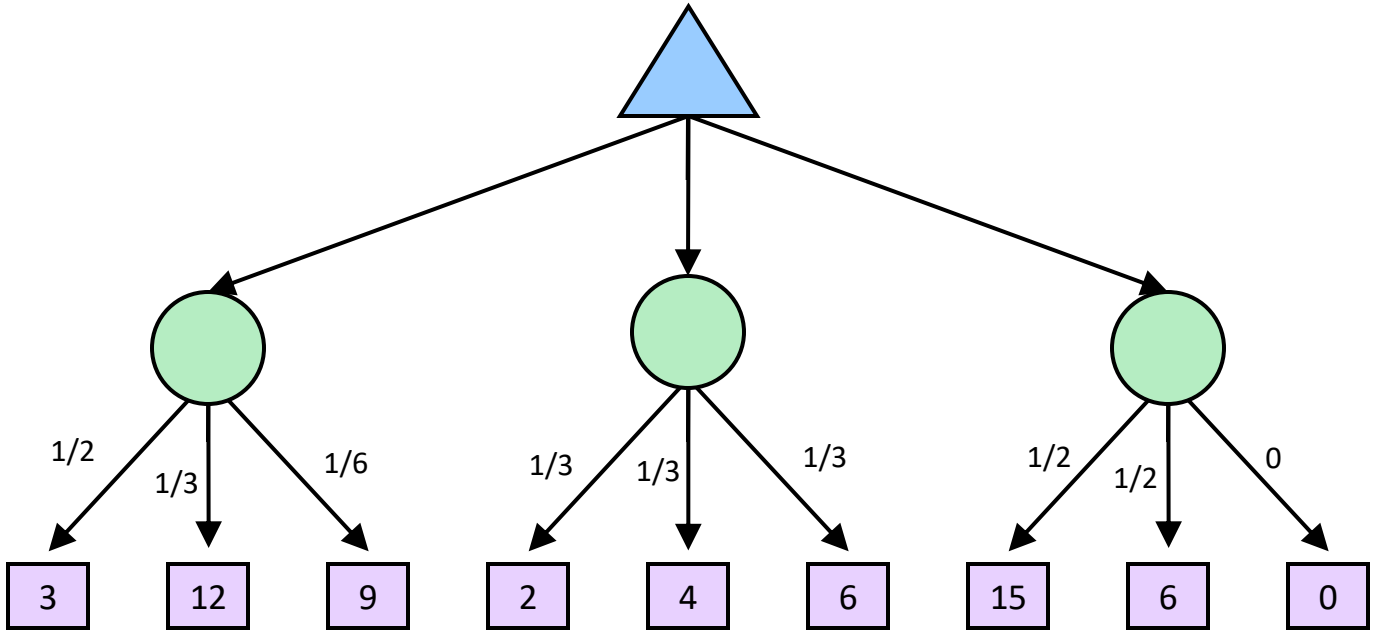
# Expectimax example of chance node

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$



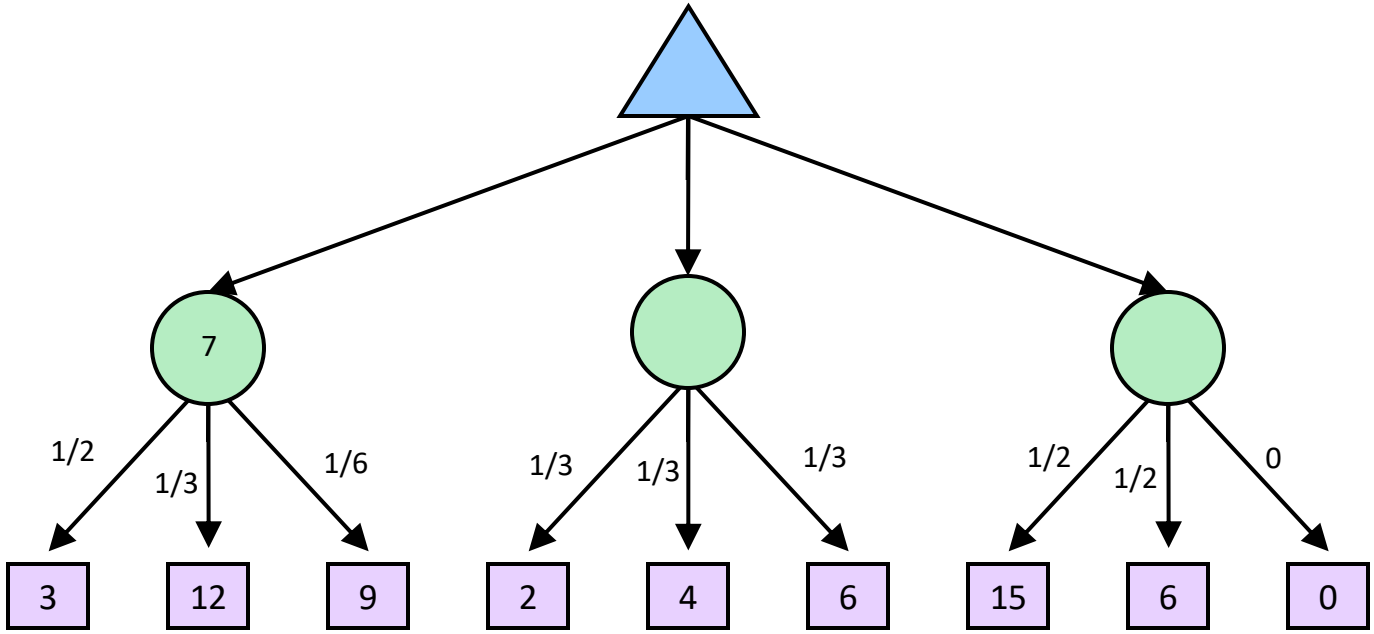

v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

# Expectimax Example

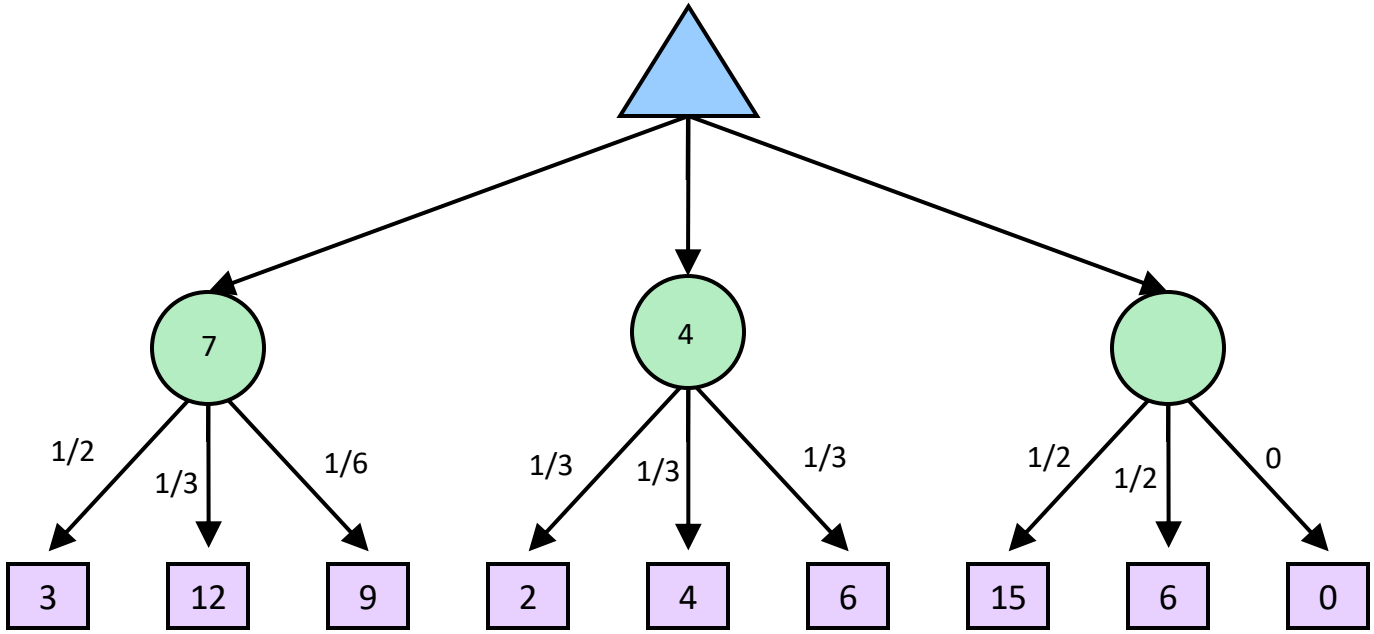



Example

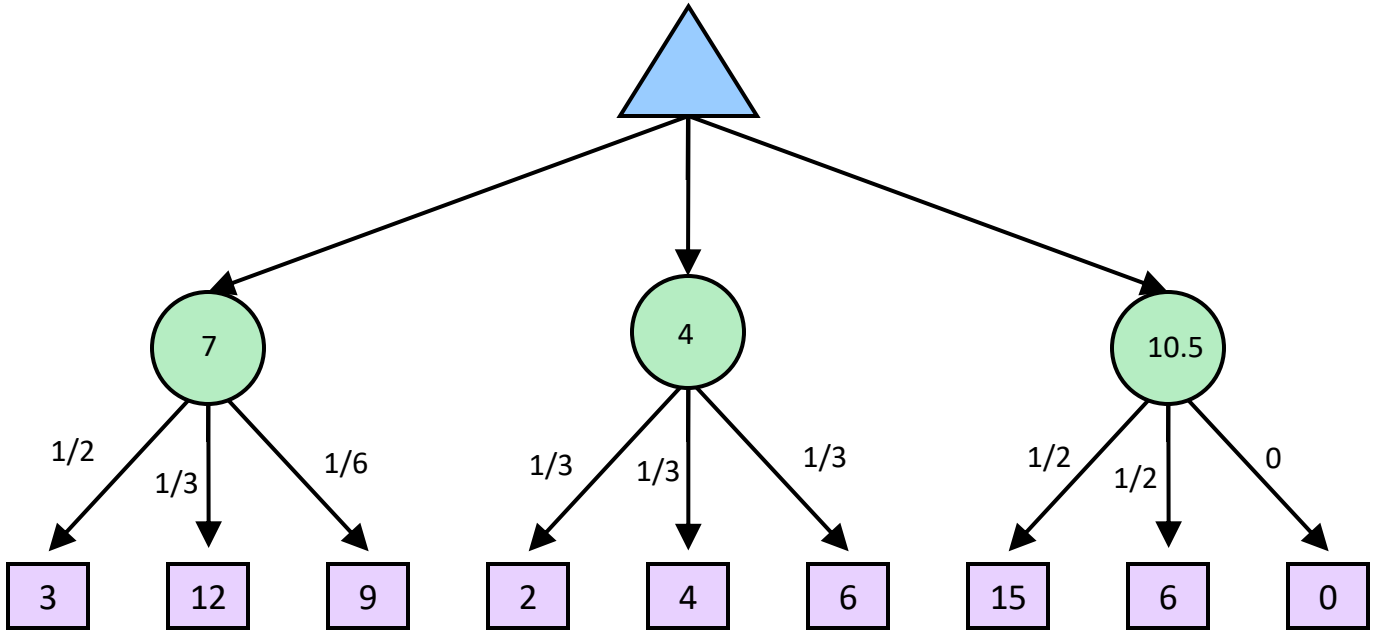
# Expectimax Example



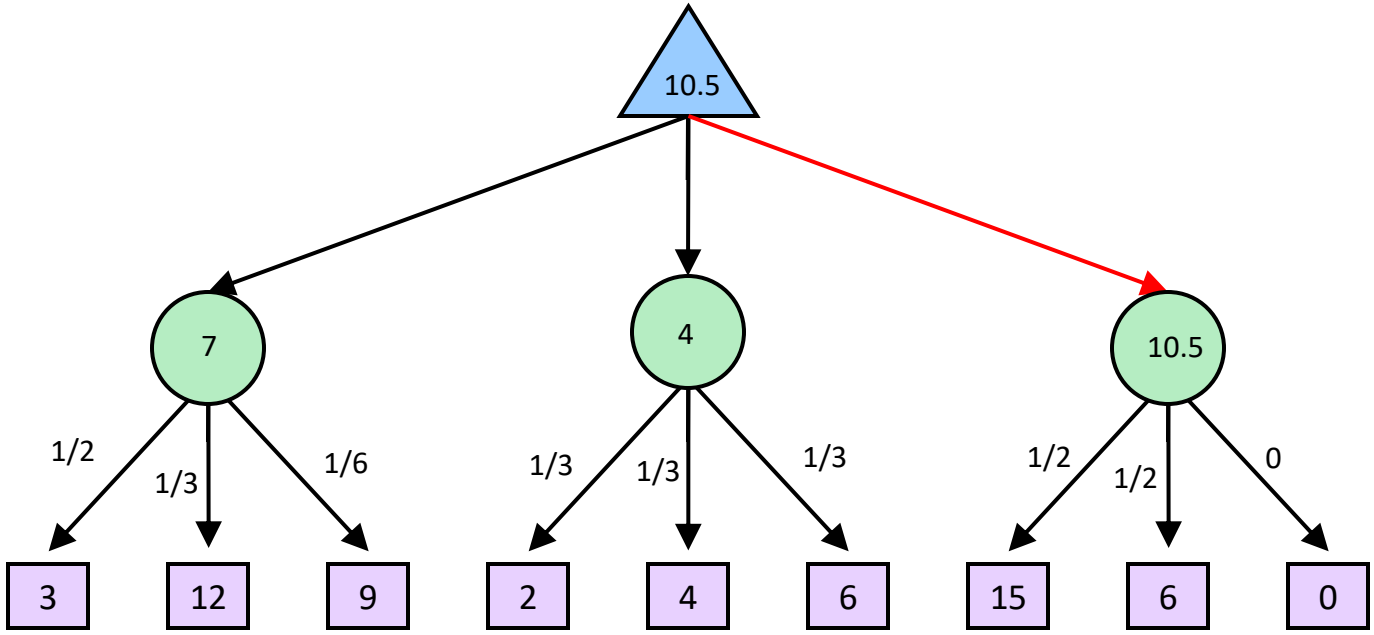
# Expectimax Example



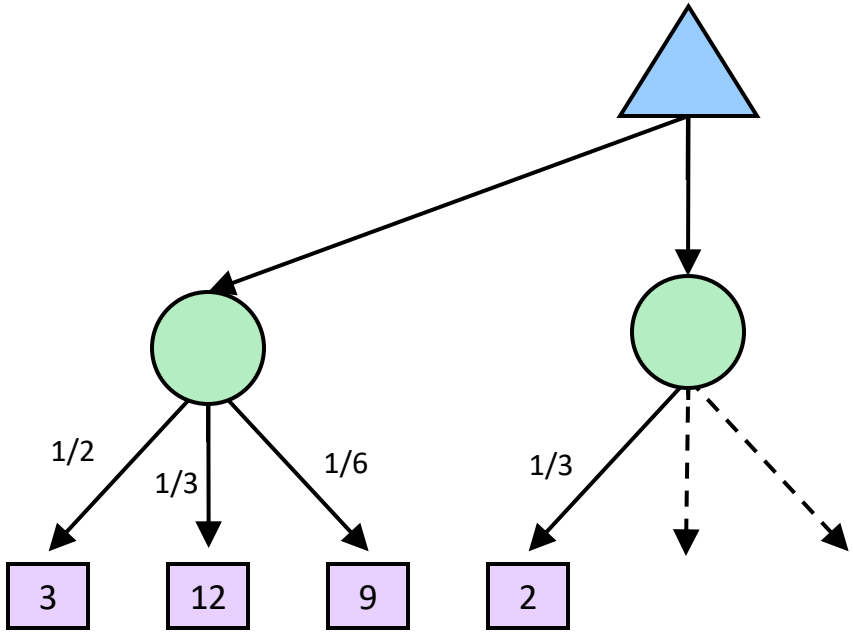
# Expectimax Example



# Expectimax Example



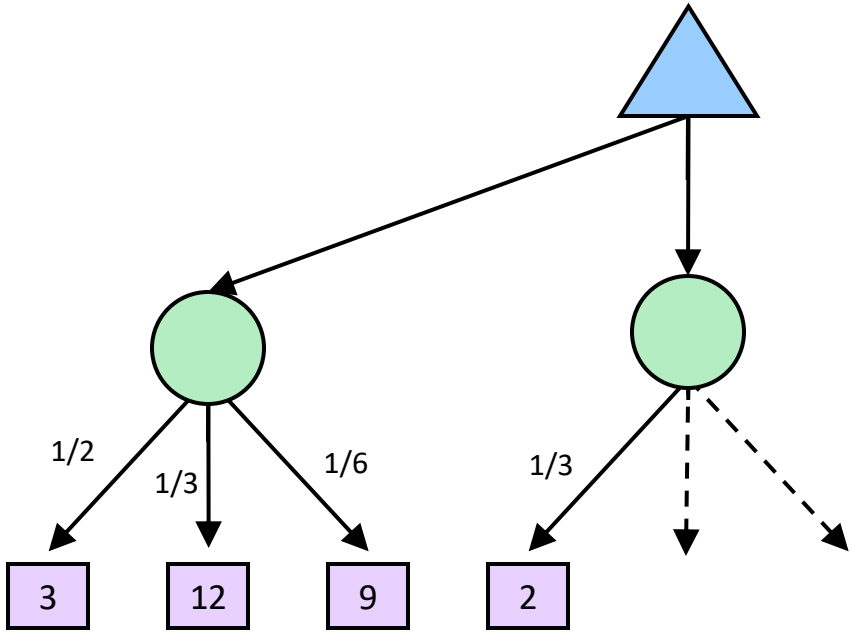
# Expectimax Pruning?




Can we prune?

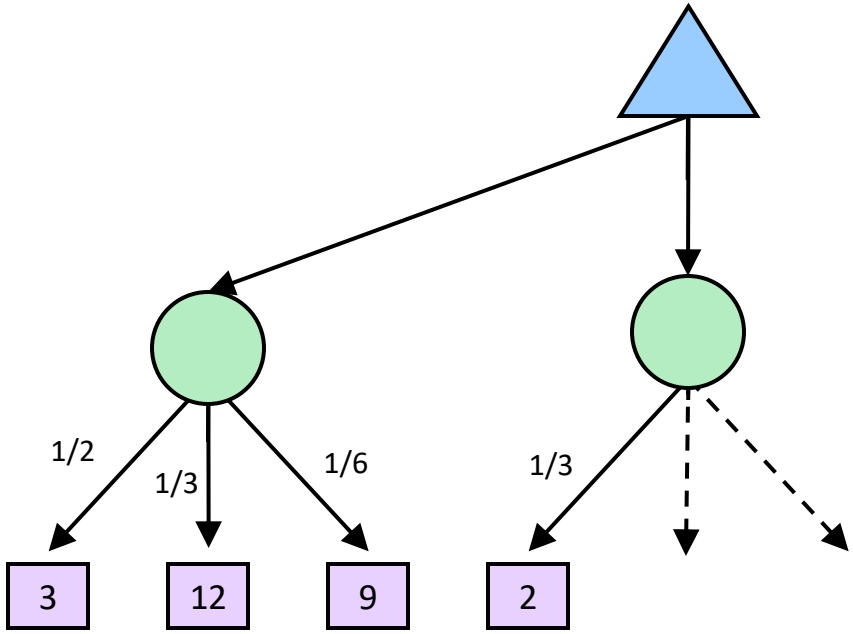
Discussion

# Expectimax Pruning?




Can we prune? Last nodes can be worth a lot.

# Expectimax Pruning?




Can we prune? Last nodes can be worth a lot.

What if the values are bounded?

# Similar tricks for real-time decisions

- Alpha-beta pruning
  - Bound for utility values
- Cut-off tests
  - Evaluation function
  - Monte Carlo simulation

# Model for chance nodes

- Expectiminimax search assumes to have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely

# Games

- Classified over different axes:
  - Deterministic or stochastic
  - Perfect or imperfect information
  - Number of players
  - Zero sum

# Games with imperfect information

- Games could be partially observable
  - Result into belief states
- Solution for each state in the belief as deterministic game (with all techniques we have seen)
  - Instead of solving all of them, randomly samples states in the belief to solve them

# Commonsense - Information is important

Mondays:

- Road A leads to 1 gold. Road B leads to a fork.
- Go left from the fork, 100 gold. Go right, die.
- *Optimal strategy: B*

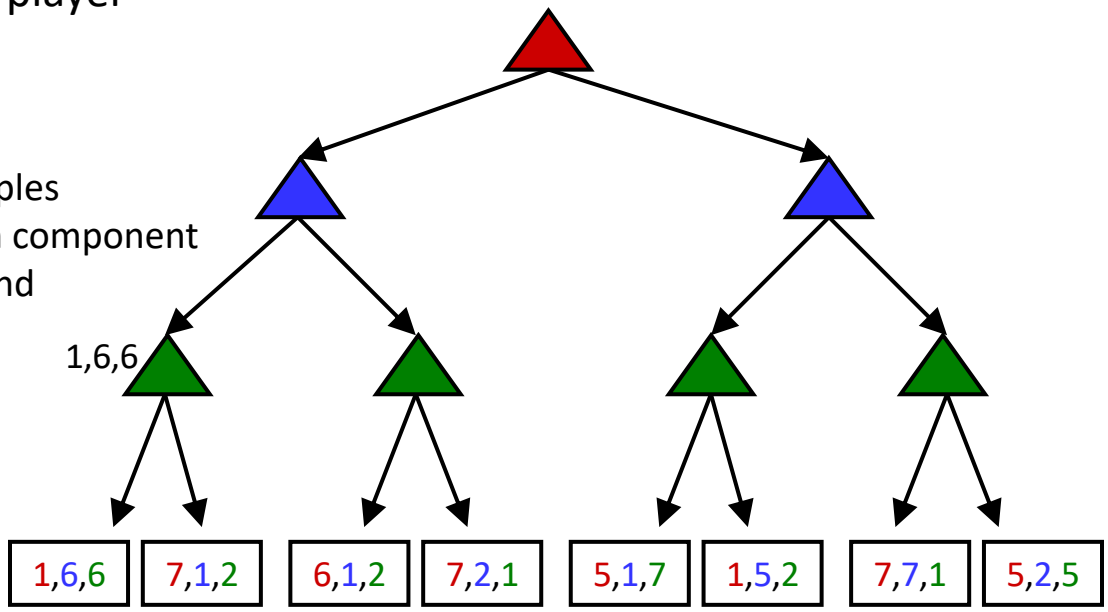
Tuesdays:

- Road A leads to 1 gold. Road B leads to a fork.
- Go left from the fork, die. Go right, 100 gold.
- *Optimal strategy: B*

I can't remember what day it is. *Choose B?*

# Multi-Agent Utilities

- With multiple players, the tree can be extended with additional nodes corresponding to each player
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically



# Summary

- Stochastic games introduce chance nodes
  - Expectiminimax for finding the solution that maximizes the average case
  - A probabilistic model is needed
- Partially observable games result in belief states

# Next

- Non-zero sum games

# Games

- Classified over different axes:
  - Deterministic or stochastic
  - Perfect or imperfect information
  - Number of players
  - Zero sum

# Non-zero sum games

- In a Non-Zero-Sum Game, all parties could gain, or all parties could lose
  - means there's at least one outcome in which (A's PAYOFF + B's PAYOFF) ≠ 0
- Game theory to find a solution
  - Agent design: determine the best strategy against a rational player and the expected return for each player
- Real world applications: negotiations, bandwidth sharing, auctions, bankruptcy proceedings, pricing decisions

# Example: Prisoner's dilemma

- Alice and Bob have both been caught red handed near the scene of a burglary by the police and are interrogated separately
- The police present options:
  - Testify against the partner
  - Refuse to testify against the partner
- Consequences:
  - If you testify against your partner and your partner refuses, you are released and your partner will serve 10 years in jail
  - If you refuse and your partner testifies against you, you will serve 10 years in jail and your partner is released
  - If both of you testify against each other, both of you will serve 5 years in jail
  - If both of you refuse, both of you will only serve 1 year in jail

- No communication

# Prisoner's dilemma

- Payoff matrix to represent the utilities

| | Bob: cooperate | Bob: defect |
|---|---|---|
| Alice: cooperate | A=-1, B=-1 | A=-9, B=-0 |
| Alice: defect | A=0, B=-9 | A=-6, B=-6 |

What would you do if you were Alice?

# Prisoner's dilemma – Normal form

- Payoff matrix to represent the utilities

| | Bob: cooperate | Bob: defect |
|---|---|---|
| Alice: cooperate | A=-1, B=-1 | A=-9, B=-0 |
| Alice: defect | A=0, B=-9 | A=-6, B=-6 |

$n = 2$

$S_1 = \{C,D\}$

$S_2 = \{C,D\}$

Each player has:

- Strategy space Si, i.e., the available actions
- Payoff function, which given the strategies returns utility

$u_1(C,C) = -1$ $\quad u_2(C,C) = -1$

$u_1(C,D) = -9$ $\quad u_2(C,D) = 0$

$u_1(D,C) = 0$ $\quad u_2(D,C) = -9$

$u_1(D,D) = -6$ $\quad u_2(D,D) = -6$

# Normal Form Representation of a Non-Zero-Sum Game with *n* players

Is a set of *n* strategy spaces $S_1$ , $S_2$ …$S_n$
where $S_i$ = The set of strategies available to player *i*

And *n* payoff functions

$u_1$ , $u_2$ … $u_n$

where

$u_i : S_1 \times S_2 \times \ldots S_n \rightarrow \Re$

is a function that takes a combination of strategies (one for each player) and returns the payoff for player *i*

# Constraint Satisfaction

- Model problems as Constraint Satisfaction Problems
- Solve them through “backtracking” (depth-first search)

# Outline

- CSP model
- CSP search

# Outline

- CSP model
- CSP search

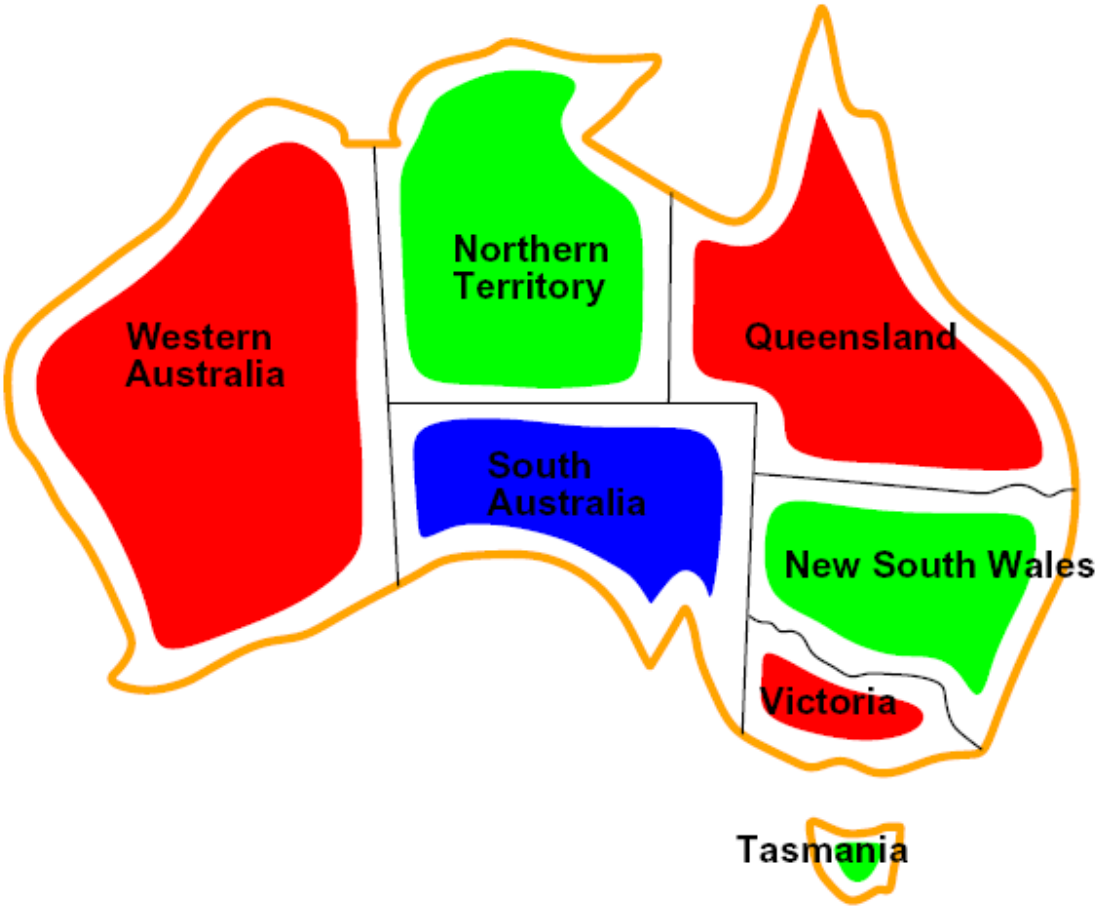
# Constraint satisfaction

- For some problems the “path” does not matter but the goal is important

- Real-world examples
  - Scheduling (job shops, section leader assignments, …)
  - Placing component on chips
  - …

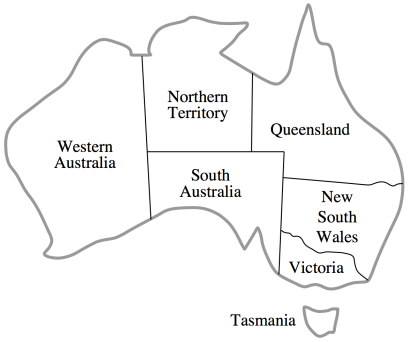
# Example: Map coloring

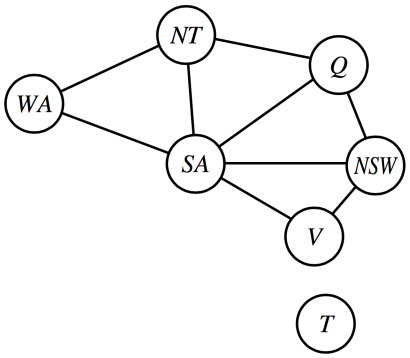
# Example: Map coloring

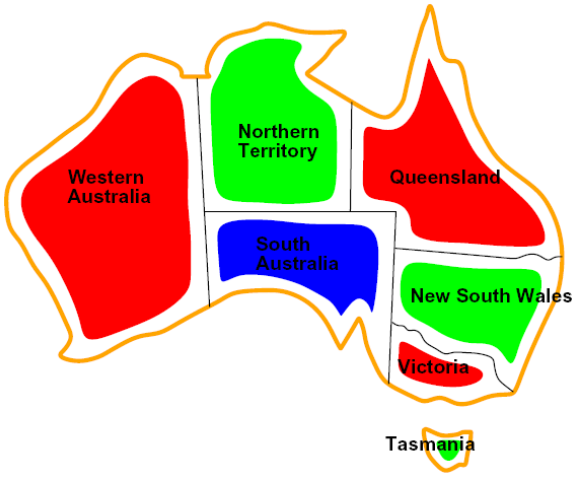


# Example: Map coloring

- Variables: WA, NT, SA, Q, NSW, V, T
- Domains: Di: r, g, b
- Constraints:
  - Explicit
    - (WA,NT)∈{(r,g),(r,b),(g,r),(g,b),(b,r),(b,g)}
    - (NT,SA)∈{(r,g),(r,b),(g,r),(g,b),(b,r),(b,g)}
    - …
  - Implicit
    - WA != NT
    - WA != SA
    - …
- Assignment:
  {WA=r, NT=g, Q=r,
  NSW=g, V=r, SA=b, T=g}



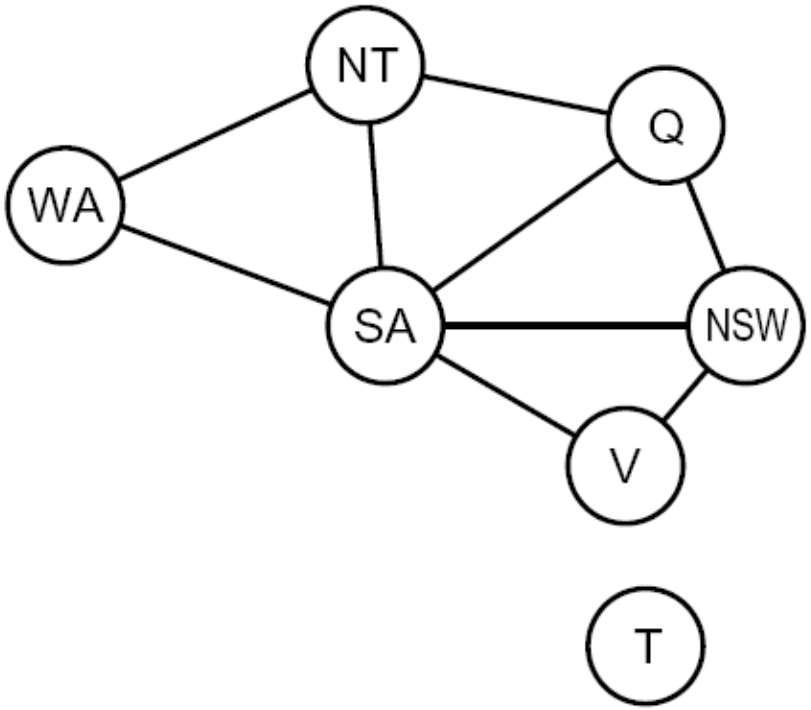



# CSP definition

- $n$ variables: $X_1, \ldots, X_n$
- For each variable, a domain Di of possible values. Example: $D_1 : X_1 \in v_1, v_2, v_3$
- $m$ constraints $C_1, \ldots, C_n$, each specifying allowable combinations of values for some set of variables
- An assignment of values to variables is the state of the problem.
  - We want to find a **complete** assignment (all variables with a value) **consistent** with the constraints

# Constraint graphs

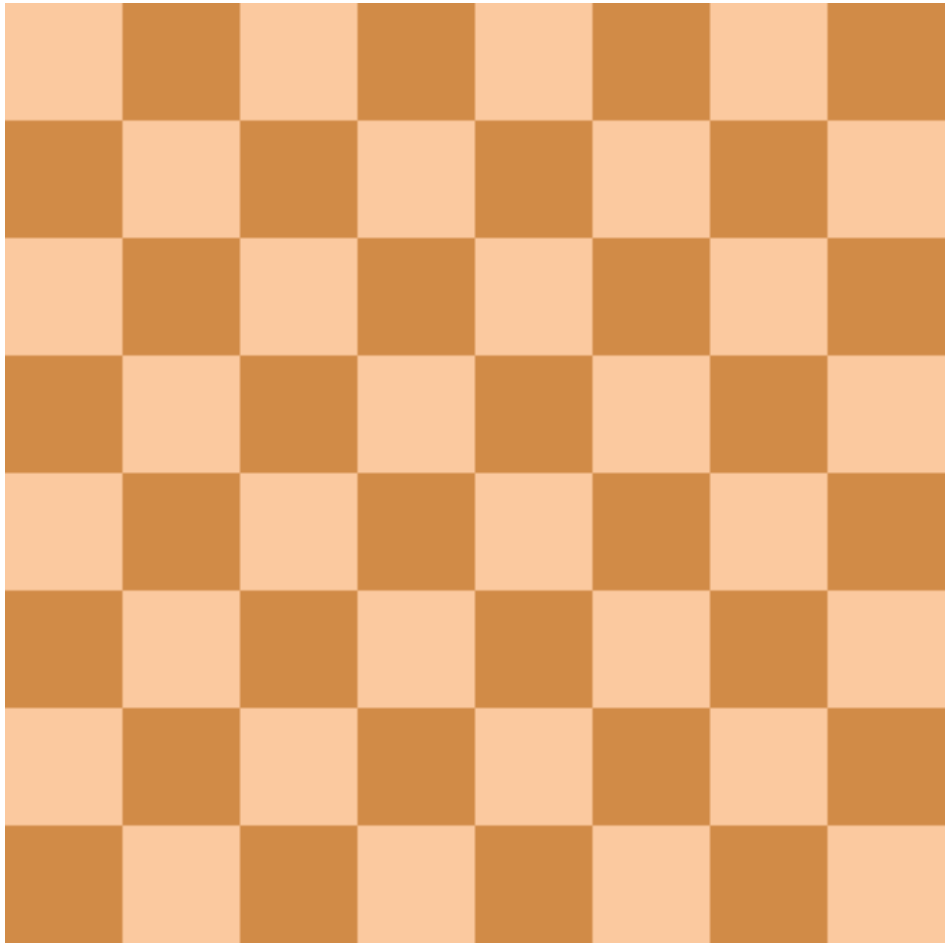
- Binary CSP: each constraint relates at most two variables
- Nodes are variable, arcs show constraints



# Varieties of constraints

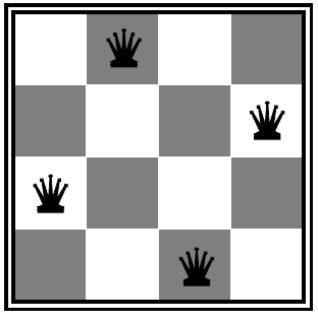
- Unary constraints involve a single variable (equivalent to reducing domains)
  - E.g., in the map coloring problem SA != g
- Binary constraints involve pairs of variables, e.g.:
  - E.g., in the map coloring problem SA != WA
- Higher-order constraints involve 3 or more variables (alldiff):
  - e.g., cryptarithmetic column constraints
- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - We won't include them in the CSP

# Example: N-Queens



# Example: 4-Queens

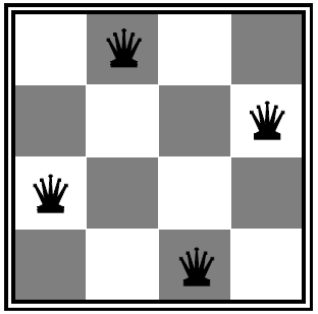
- Formulation 1:
  - Variables: Q1, Q2, Q3, Q4
  - Domains: Qi = {(x,y) | x in [0,3] and y in [0,3]}
  - Constraints
    - Any queen not in the same x
      - x1!=x2, x1!=x3, …
    - Any queen not in the same y
      - y1!=y2 …
    - Any queen not on the diagonal
      - abs(y2-y1) != abs (x2-x1) …



Discussion

# Example: N-Queens

- Formulation 2:
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints



$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Discussion

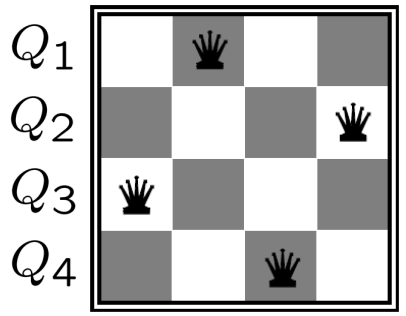
# Example: N-Queens

- Formulation 3:
  - Variables: $Q_k$
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:

Implicit: $\forall i, j$ non-threatening$(Q_i, Q_j)$

Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$

$\ldots$



# Example: Cryptarithmetic

$X_1$

```
  T W O
+ T W O
-------
F O U R
```

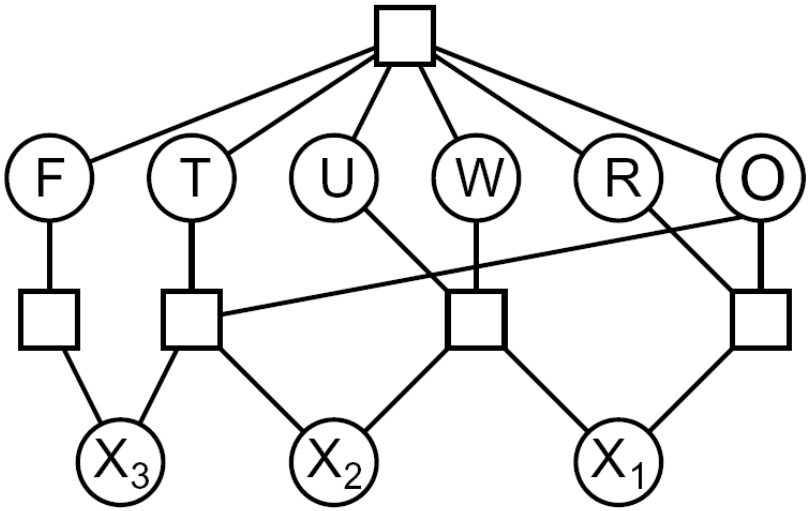
- Variables

  $F \; T \; U \; W \; R \; O \; X_1 \; X_2 \; X_3$

- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints

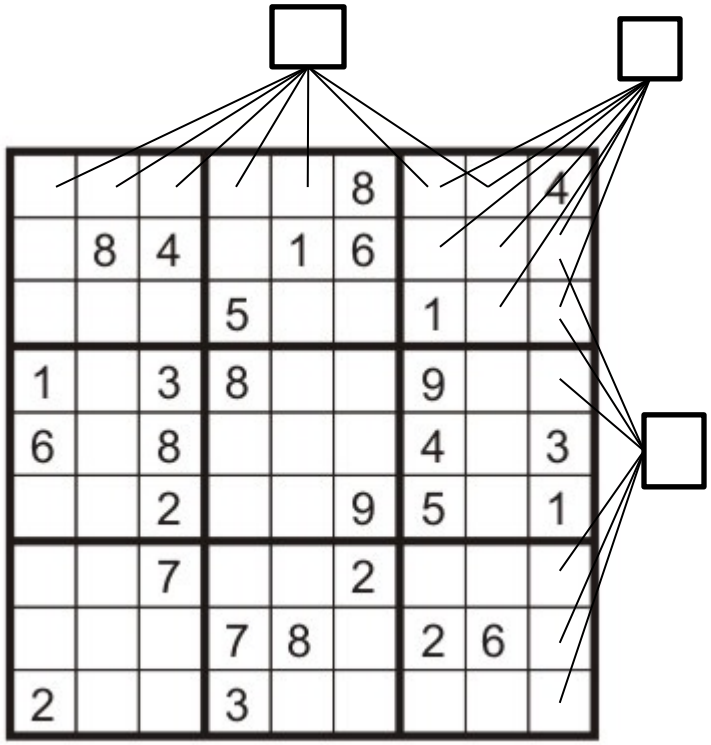
  alldiff$(F, T, U, W, R, O)$

$$O + O = R + 10 \cdot C_{10}$$
$$C_{10} + W + W = U + 10 \cdot C_{100}$$
$$C_{100} + T + T = O + 10 \cdot C_{1000}$$
$$C_{1000} = F \; ,$$




Discussion

# Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

Discussion

# Example: Assignment problem

- Assign four workers W1,W2,W3,W4 to four products such that each worker works on one product and each product is produced by one worker.
- Effectiveness of production is given by the following table (e.g., worker W1 produces P1 with effectiveness 7) and the total effectiveness must be 19 at least

| | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| W1 | 7 | 1 | 3 | 4 |
| W2 | 8 | 2 | 5 | 1 |
| W3 | 4 | 3 | 7 | 2 |
| W4 | 3 | 1 | 6 | 3 |

Discussion

# Example: assignment problem

- Variables
  - W1, W2, W3, W4
- Domains:
  - {P1, P2, P3, P4}
- Constraints
  - All_diff(W1, W2, W3, W4)
  - E1(W1)+E2(W2)+E3(W3)+E4(W4)>=19

| | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| W1 | 7 | 1 | 3 | 4 |
| W2 | 8 | 2 | 5 | 1 |
| W3 | 4 | 3 | 7 | 2 |
| W4 | 3 | 1 | 6 | 3 |

# Example: The circuit-board layout problem

- Variables:
  - a variable for each piece
- Domains:
  - Possible placements over the board
- Constraints:
  - Needs to be in the board
  - Needs not to overlap with other pieces