CS 161 Intro. To Artificial Intelligence

Week 4, Discussion 1D

Stolen with permission from previous TAs

Arc Consistency (AC)

Arc consistency (**before** search):

- An arc is unidirectional: (X_i, X_j) or $(X_i \leftarrow X_j) \neq (X_j, X_i)$ or $(X_j \leftarrow X_i)$
- Variable is arc consistent: Every value in its domain satisfies the variable's binary constraints
  - X_i is arc consistent with respect to another variable X_j if for every value in the current domain D_i, there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)
- Network is arc consistent: every variable is arc consistent with every other variable

Arc Consistency (AC) - Example

Q: The initial domain of both X and Y is the set of digits (0~9). $Y = X^2$. What will the domains of X and Y be after enforcing arc consistency?

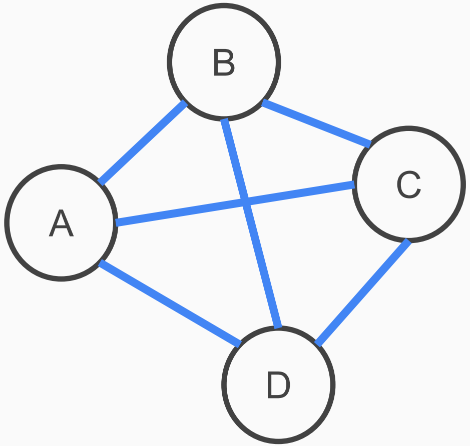
- Consider the arc (X ← Y)
  - Make X arc-consistent with respect to Y
    - For any value in X's domain, there should be at least one value in Y's domain that satisfied the constraint
  - X: {0,1,2,3}
- Consider the arc (Y ← X)
  - Y: {0,1,4,9}

- Result -- X: {0,1,2,3}, Y: {0,1,4,9}

Arc Consistency (AC)

Evaluation of AC:

- Notations:
  - n: # of variables
  - d: largest domain size
  - c: # of constraints
- Time complexity: $\mathbf{O(n^2 d^3)}$
  - Checking consistency of one arc: $O(d^2)$
    - Look at different combinations of values
  - At most $(n^2\text{-}n)$ arcs: $O(n^2)$ or O(c) ← 2*c constraints
  - Each arc (X_i, X_j) can be inserted at most d times
    - X_i has at most d values to delete



Comparison of AC and FC

AC3:

- Before search
- **initialization**: push all arcs in the queue
- **maintain the queue**: when some variable X_i's domain size change, push (X_k <- X_i) into the queue, where X_k is neighbor of X_i. ➔ Do both POP and PUSH!

FC:

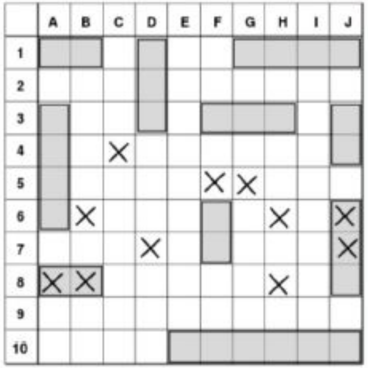
- During search
- **initialization**: when assign value to X, push all X's neighbor into the queue. (neighbor <- X)
- **maintain the queue**: queue won't add anything after initialized. ➔ ONLY POP, NO PUSH!

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Go: Perfect and Deterministic

Monopoly: Perfect, Chance Introduced



Battleship: Imperfect and Deterministic

Bridge: Imperfect, Chance Introduced

Game as a Search Problem

Can we use search strategies to win games?

- Require to make some decision when calculating the optimal decision is infeasible
- The **solution** will be a **strategy** that specifies a move for every possible opponent reply
- Challenges:
  - Very, very large search space
  - Time limits

Game as a Search Problem

- **S0**: The initial state, which specifies how the game is set up at the start.
- **PLAYER(s)**: Defines which player has the move in a state.
- **ACTIONS(s)**: Returns the set of legal moves in a state.
- **RESULT(s, a)**: The transition model, which defines the result of a move.
- **TERMINAL-TEST(s)**: A terminal test, which is true when the game is over and false otherwise.
  - States where the game has ended are called terminal states.
- **UTILITY(s, p)**: A utility function that defines the final numeric value for a game that ends in terminal state s for a player p.
  - Also called an objective function or payoff function.
  - In chess, the outcome is a win, loss, or draw, with values $+1$, 0, or $1/2$.

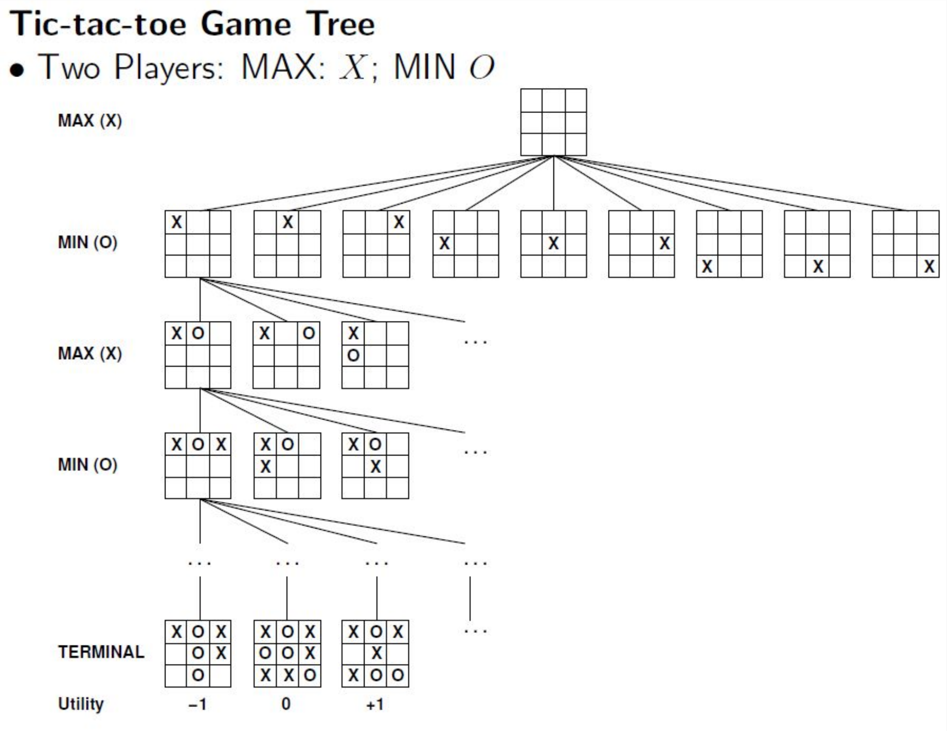
Optimal Decisions in Games

How to find the optimal decision in a deterministic, perfect-information game?

Idea: choose the move with highest achievable payoff against the best play of the other player

Partial Game Tree:

- Top node is the initial state
- Giving alternating moves by MAX and MIN



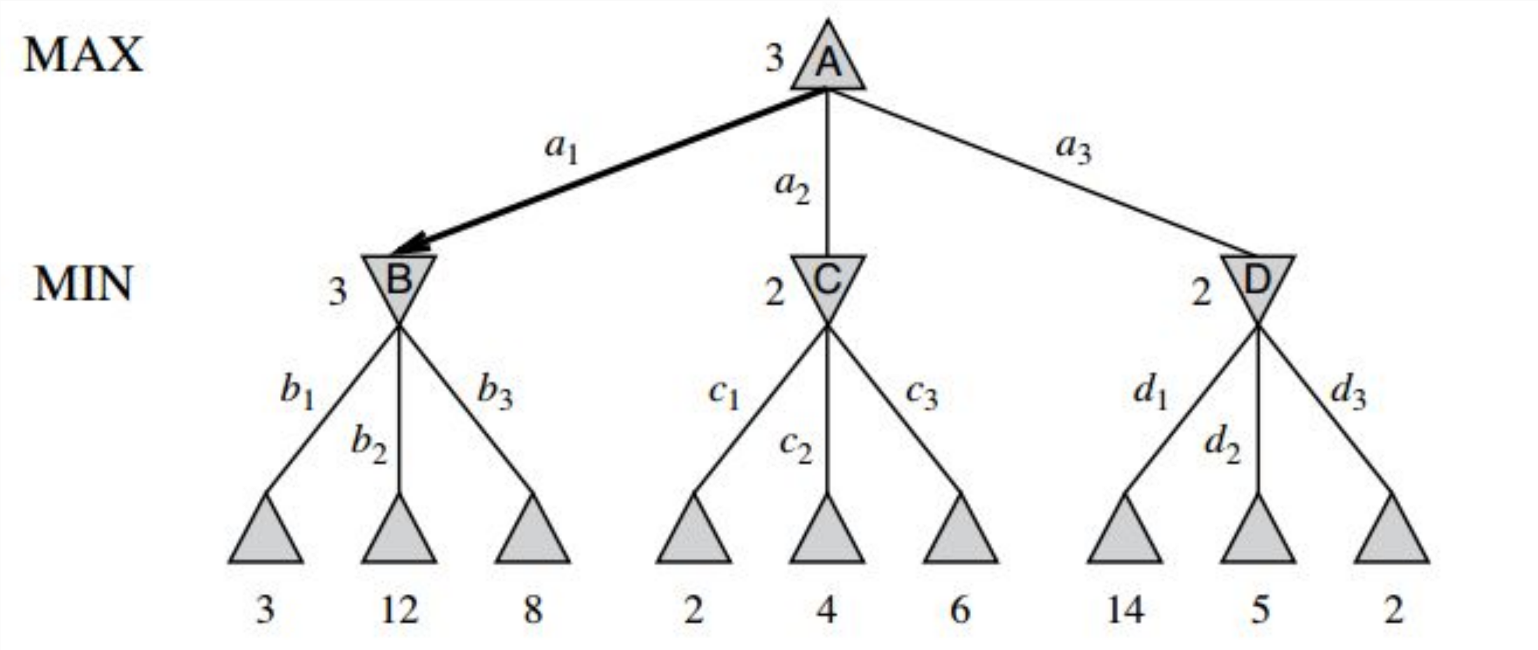
Minimax Algorithm

- Imagine we are MAX
- We refer to the payoff as MINIMAX value, at each step
  - MAX wants MINIMAX value to be as <u>big</u> as possible
  - MIN wants MINIMAX value to be as <u>small</u> as possible

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

Minimax Algorithm

△ : takes max value among child nodes ▽: takes min value among child nodes



Minimax Algorithm

Evaluation:

- Complete (if tree is finite)
- Optimal (since we are against an optimal opponent)
- Time complexity: **O**$(\boldsymbol{b^m})$
  - b: max # of children nodes for one parent node
  - m: max depth of the state space
- Space complexity: **O**$(\boldsymbol{bm})$ ➔ depth-first exploration

Actually don't need to explore every path and compute MINIMAX for every node!

- Increase the efficiency
- Use Alpha-beta pruning

Alpha-beta Pruning

Minimax: a way of finding an optimal move in a two player game.

Alpha-beta pruning: finding the optimal minimax solution while avoiding searching subtrees of moves which won't be selected.

- α : maximum lower bound of possible solutions
- β : minimum upper bound of possible solutions
- If N is estimated value of the node, then $\alpha \leq N \leq \beta$

Alpha-beta Pruning

α : maximum lower bound of possible solutions

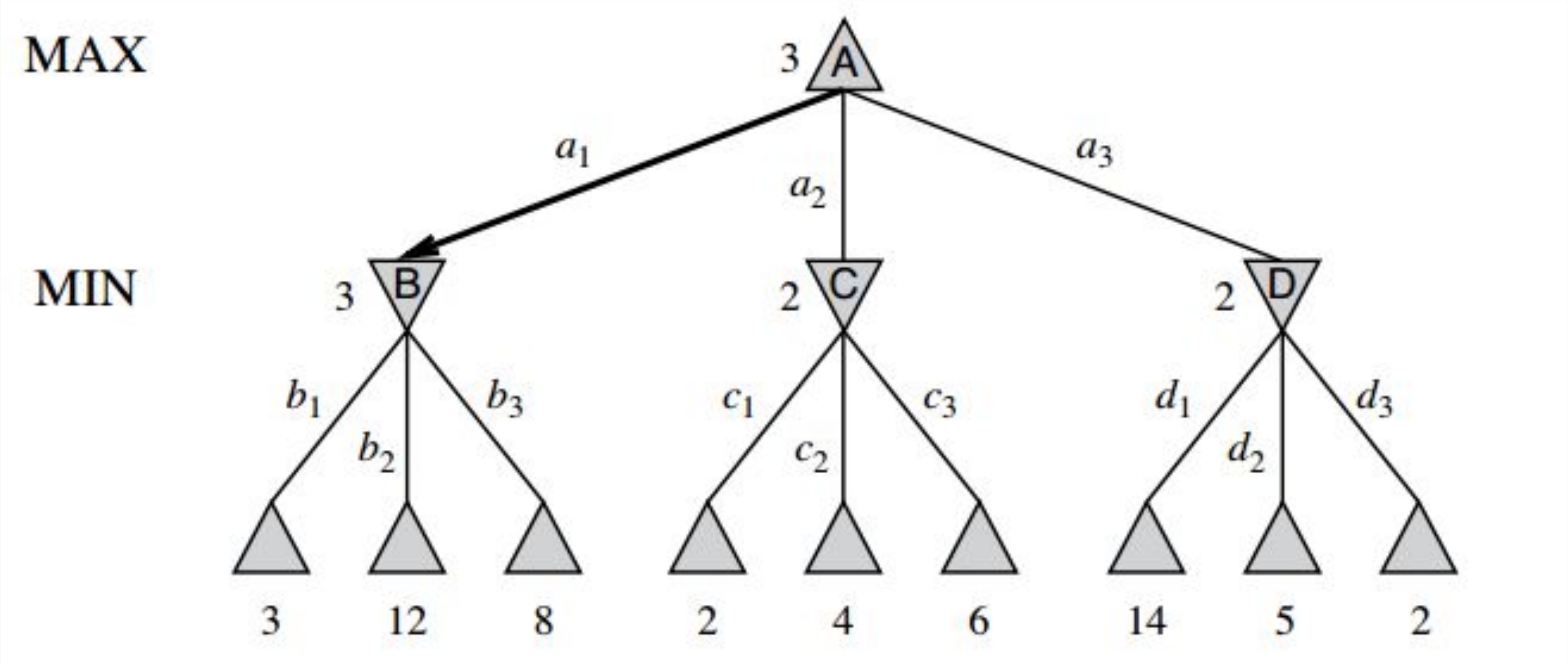
β : minimum upper bound of possible solutions

If N is estimated value of the node, then $\alpha \leq N \leq \beta$

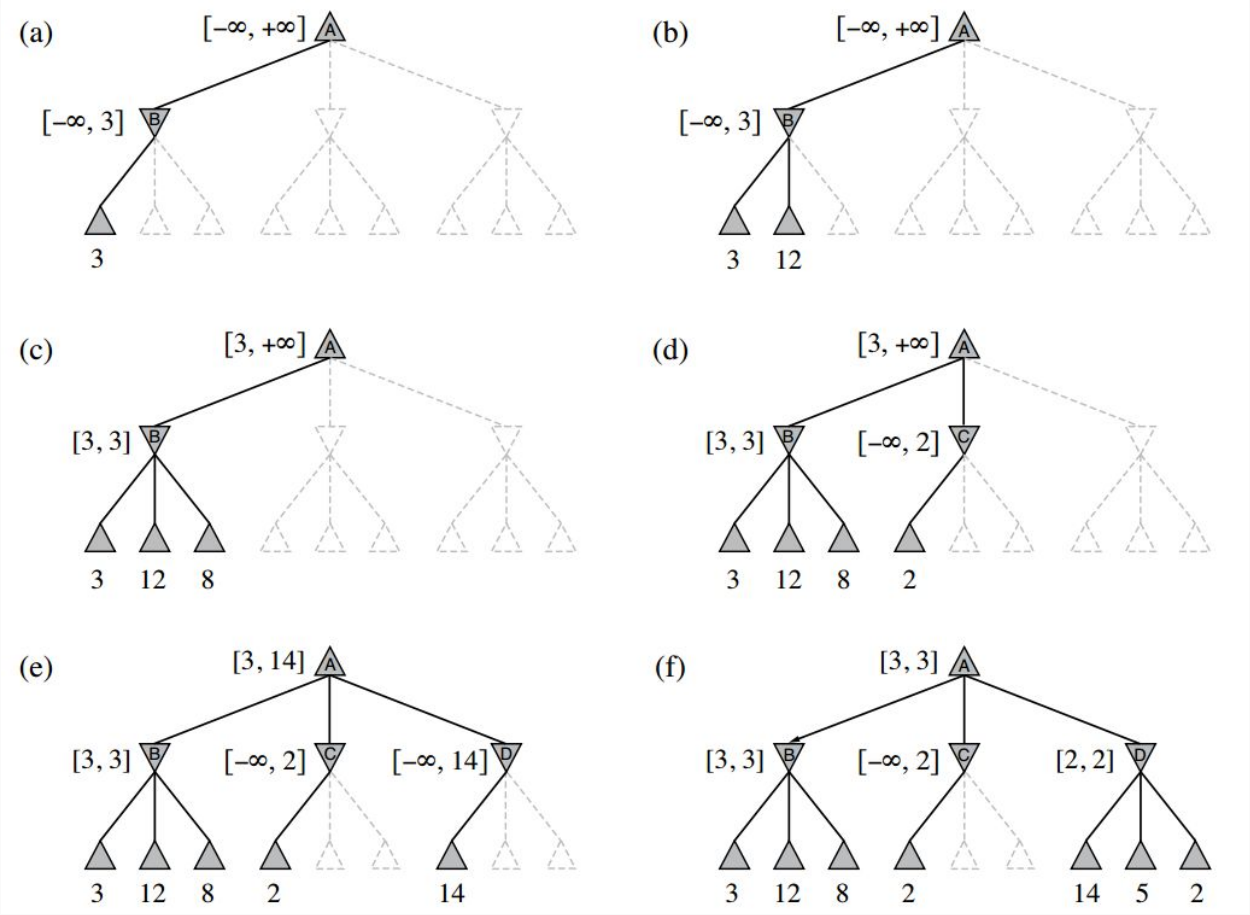
Steps:

- During the search, each node carries an upper bound α a lower bound β
- Pushing bound upward:
  - When a child returns, it pushes its value onto the parent (**always tighten the bound**)
  - Max player will modify its lower bound, and min player will modify its upper bound
- Pushing bound downward (**both lower and upper**) and prune:
  - If min parent, max children, when $\alpha_{children}$ >= β_{parent} , prune!
  - If max parent, min children, when α_{parent} >= $\beta_{children}$, prune!

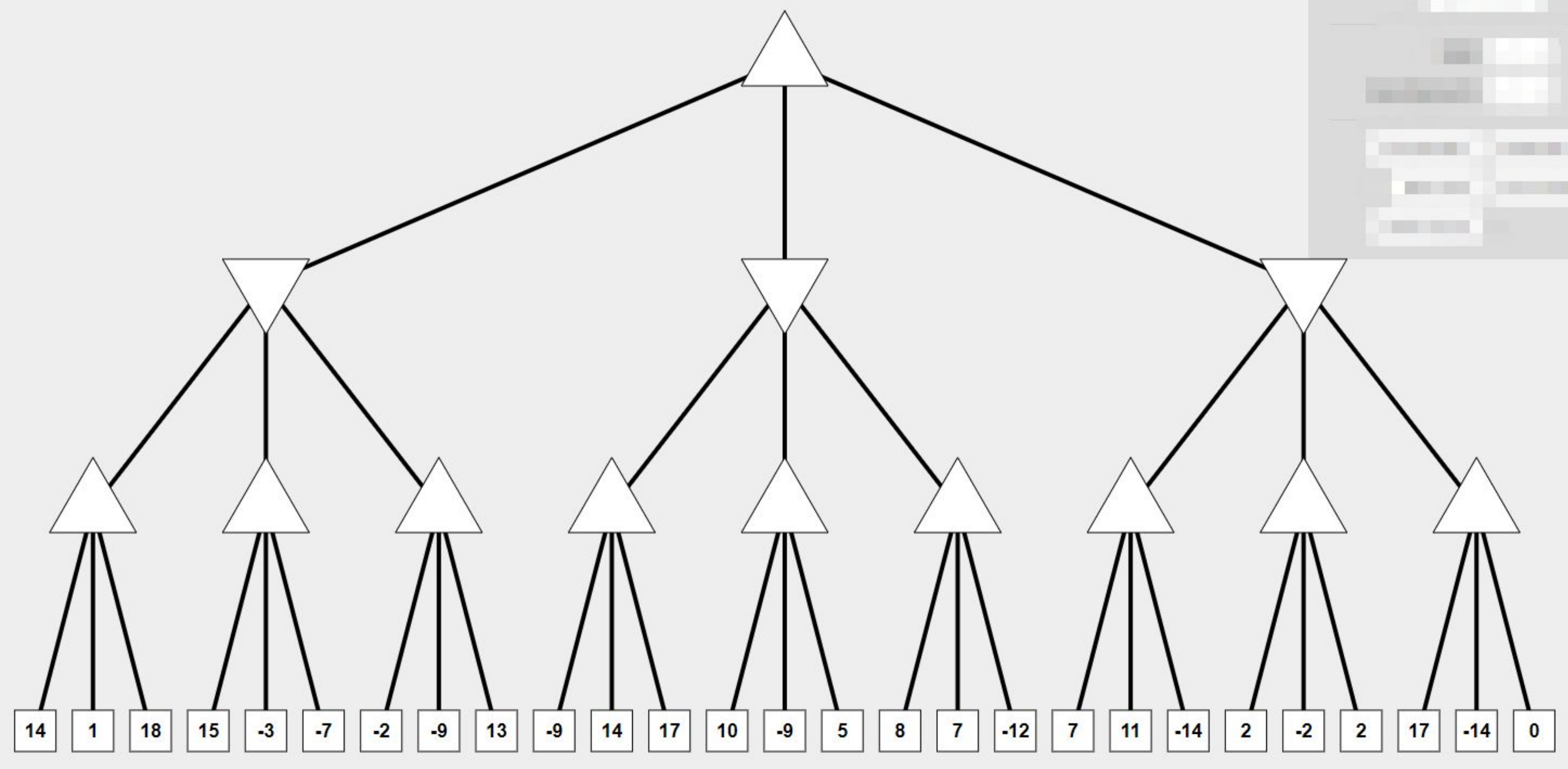
Alpha-beta Pruning - Example



Alpha-beta Pruning - Example



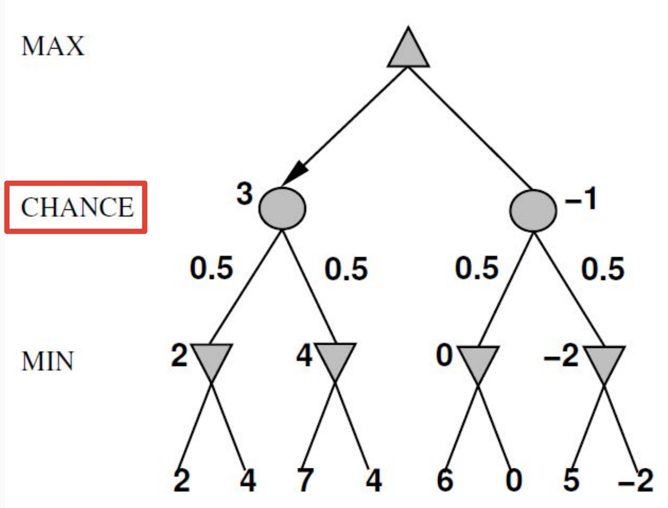
Alpha-beta Pruning - Practices



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

Nondeterministic Game

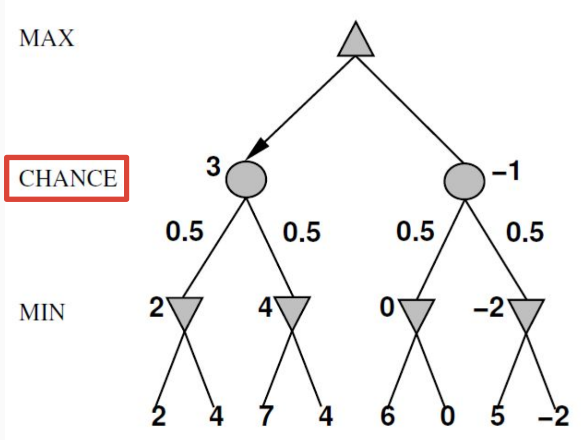
- In deterministic games with perfect information, Minimax Algorithm gives perfect play
- What if the game is nondeterministic with perfect information?
  - In nondeterministic games, chances are introduced
  - For example:
    - Two people MAX and Min play a game
    - Flip a coin after each player make a decision
    - The result of coin flipping changes the state



Expected Minimax Algorithm

- In nondeterministic games, EXPECTMINIMAX gives perfect play
  - Just like MINIMAX, except we must also handle chance nodes

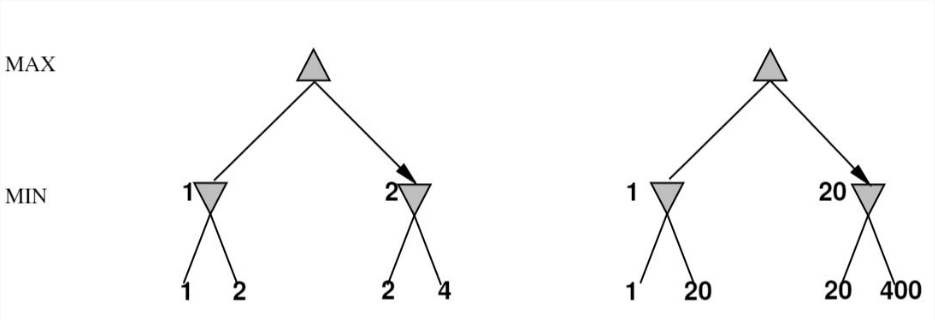
if *state* is a MAX node **then**
  return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a MIN node **then**
  return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
  return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

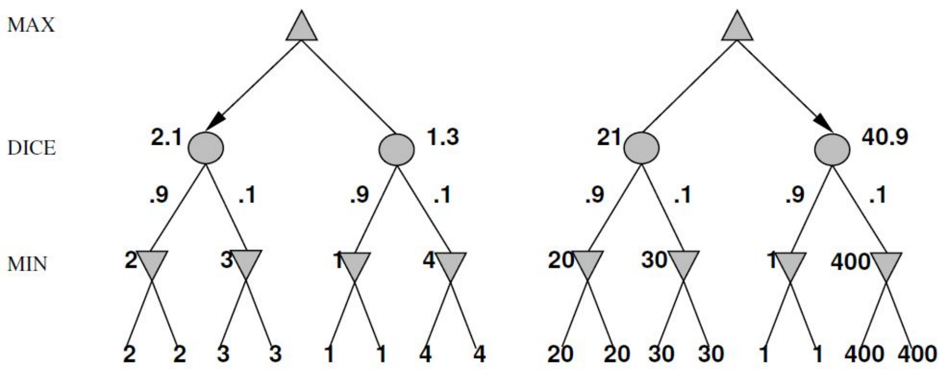


$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r))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

Expected Minimax Algorithm

- Unlike MINIMAX algorithm where only **order** of terminal nodes matters, in EXPECTMINIMAX algorithm, **exact values** of terminal nodes also matter!





Logic

- Logic: knowledge representation language
  - Represent human knowledge as "**sentences**" (a.k.a *axiom*)
    - **Knowledge base (KB)**: a set of sentences
- Examples
  - **Propositional logic**
    - Boolean logic
  - **First-order logic**
    - Quantifiers $\forall$, $\exists$, objects and relations
- Key components in Logic
  - Syntax: how to write sentences
  - Semantics: how to interpret sentences
  - Reasoning/Inference: What new knowledge can be derived from known facts

Propositional Logic - Syntax

Syntax:

- Atomic sentence
  - A single propositional symbol, like A (A can be True or False)
- Logical connectives
  - ¬ not
  - ∧ and (**conjunction**)
  - ∨ or (**disjunction**)
  - ⇒ ($or \rightarrow$) implication
  - ⇔ if and only if
- Complex sentence
  - $A \vee B$, $A \vee \neg C \Rightarrow B$, ...
- A special type of sentence: Horn clause

Syntax Forms

Syntax Forms:

CNF (Conjunction Normal Form): $(A \lor \neg B) \land (A \lor \neg C \lor D)$

- CNF consists of **clauses** that are connected by conjunction.
  - **Clauses:** disjunctions of **literals** (a symbol or its negation).
- $(A \lor \neg B) \land (A \lor \neg C \lor D)$
  - 2 clauses: $(A \lor \neg B)$, $(A \lor \neg C \lor D)$
  - 4 variables: A, B, C, D
  - Literals: A, $\neg B$, $\neg C$, D

DNF (Disjunction Normal Form): $(A \land \neg B) \lor (A \land \neg C \land D)$

- All propositional sentences can be converted to CNF/DNF.
- We will mainly use CNF. For most algorithms, you will need to standardize the sentence by converting

Syntactic Forms - NNF

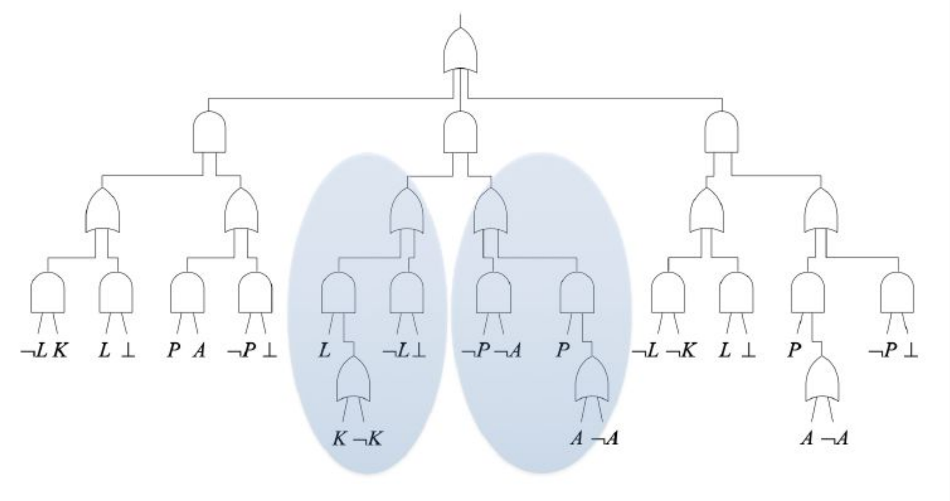
Syntax Forms:

NNF (Negation Normal Form): $((\neg A \vee B) \vee (A \vee \neg C)) \wedge D)$

- Negation signs only appears next to variables
- E.g. $(\neg(A \vee B) \vee (A \vee \neg C)) \wedge D)$ is not NNF

Decomposable NNF:

- Subcircuits/Inputs of an and-gate cannot share variables
- Satisfiability of DNNF can be decided in linear time



Horn Clause

Horn Clause:

- Proposition symbol; or
- (**conjunction** of symbols) ⇒ symbol
- E.g. $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

$A \vee B \vee \neg C$ ✗

$\neg A \vee B \vee \neg C$ ✓ $\equiv A \wedge C \Rightarrow B$

$\neg A \vee \neg B \vee \neg C$ ✓

A typical form of horn clause

Horn Form: When KB (knowledge base) = **conjunction** of Horn clauses

Why do we care about Horn clause?

- It's a special type! If the sentences are Horn clauses, inference can be done in linear time (exponential for general sentences)

Syntactic Forms – Tractable and Universal

Tractable:

- A problem is intractable if the time required to solve it grows exponentially with its size
- If a problem is tractable, it becomes easy to solve in some particular form.
  - Expressing knowledge in a tractable form helps with inference
  - E.g. Solving satisfiability is NP-complete on general propositional sentences, but it’s linear on horn forms □ horn forms is tractable

Universal:

- A form is universal if any sentence of propositional logic can be converted to this form.
  - E.g. CNF is universal because any propositional sentences can be converted to CNF
  - In this class, we just tell you whether a form is universal □ proof can be complicated

Syntactic Forms – Tractable and Universal

CNF: ●

DNF: ● ▲

Horn: ▲

NNF: ●

DNNF circuits: ● ▲

Universal -- ●

Tractable -- ▲

Semantics

- Answers when is a sentence true:

$P \Rightarrow Q$ is equivalent to $\neg P \vee Q$

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *true*.

Semantics

Logical Equivalence:

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

Entailment

Entailment is a relationship between sentences (syntax) that is based on semantics

- Use |= to denote entailment:
  - $\alpha \models \beta$: α entails β
  - α, β can be single sentence or KB

- Properties:
  - $\alpha \models \beta$ iff every model in which α is true, β is also true
  - $\alpha \models \beta \Leftrightarrow M(\alpha) \subseteq M(\beta)$
  - Any two sentences α and β are equivalent only if each of them entails the other
    - i.e., $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$.

Entailment

More properties:

- KB Δ entails a sentence α is denoted as $\Delta \models \alpha$ if $M(\Delta \wedge \alpha) = M(\Delta)$.
- KB Δ is consistent with sentence α if $M(\Delta \wedge \alpha)$ is non-empty.
- KB Δ contradicts sentence α if $\Delta \wedge \alpha$ is not satisfiable.

Sanity check: KB entails α iff it contradicts $\neg\alpha$ → used in inference

Why is entailment so important?

- We have some known facts represented as a knowledge base KB
- Now we make a new claim β
- Does our known facts support this new claim β?

Terminology Semantics

- w: world - some boolean assignment to every variable
- $w \models \alpha$: w entails α - a world where sentence α is true!
- $M(\alpha)$: models of α - set of worlds that entail α
- α equivalent to β
  - $M(\alpha) = M(\beta)$
- α is contradictory/inconsistent
  - $M(\alpha) = \varnothing$
- α is valid/tautology
  - $M(\alpha)$ = set of all worlds
- α and β are mutually exclusive
  - $M(\alpha) \cap M(\beta) = \varnothing$
- α implies β
  - $M(\alpha) \subseteq M(\beta)$