Adversarial Search (Games)

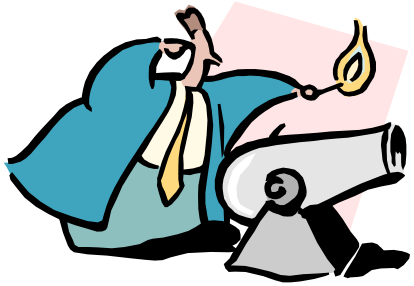
Department of Electrical and Electronics Engineering
Spring 2005
Dr. Afşar Saranlı

Thanks to Professor Andrew W. Moore (Carnegie Mellon University) http://www.cs.cmu.edu/~awm/tutorials
Also: Artificial Intelligence: A Modern Approach, 2nd Ed., Russel & Norvig

Overview

- “Adversarial Search” and game terminology,
- Game trees and definitions,
- Optimal moves in games: MiniMax algorithm,
- Optimal with savings: Alpha-Beta Pruning,
- Real-time decisions and realistic game playing (*Evaluation functions* and *imperfect decisions*),
- Taste of non-deterministic (probabilistic) games

“Adversarial Search” (Games)



- A multi-agent environment,
- More: Competitive!! Goals are in conflict
- Multiple agents with significant impact to each other: ***Mathematical Game Theory,***
- A lot of agents with very small direct impact to each other: ***An Economy.*** (not our scope)

Our Games for Now

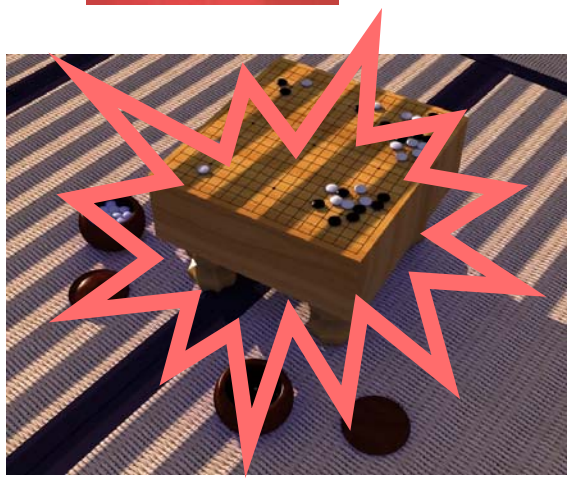
- Two player, zero-sum, discrete, finite, deterministic games of perfect information,
  - Two player: Hmmm…
  - Zero-sum: In any outcome of any game, Player P1's gains equal player P2's losses. (Does not mean fairness)
  - Discrete: All game states and decisions are discrete values.
  - Finite: Only a finite number of states and decisions.
  - Deterministic: No chance (no die rolls). We will briefly talk about it.
  - Perfect information: Both players can see the state, and each decision is made sequentially (no simultaneous moves).

Which of these fits our definition?

- Two player
- Zero-sum
- Discrete
- Finite
- Deterministic
- Perfect information

Which of these fits our definition?

- Two player
- Zero-sum
- Discrete
- Finite
- Deterministic
- Perfect information



Definition

A "Two-player zero-sum discrete finite deterministic game of perfect information" is a quintuplet: *(S , I , Succs , T , V)* where

S	=	a finite set of game states (includes info determining who is due to move)
I	=	the initial state
Succs	=	a function which takes a state as input and returns a set of possible next states available to whoever is due to move
T	=	a subset of S. It is the terminal states: the set of states at which the game is over
V	=	a *utility function* mapping terminal states to real numbers. It is the amount that A wins from B. (If it's negative A loses that amount to B).

Convention: assume Player P1 moves first.

Also: assume turns alternate.

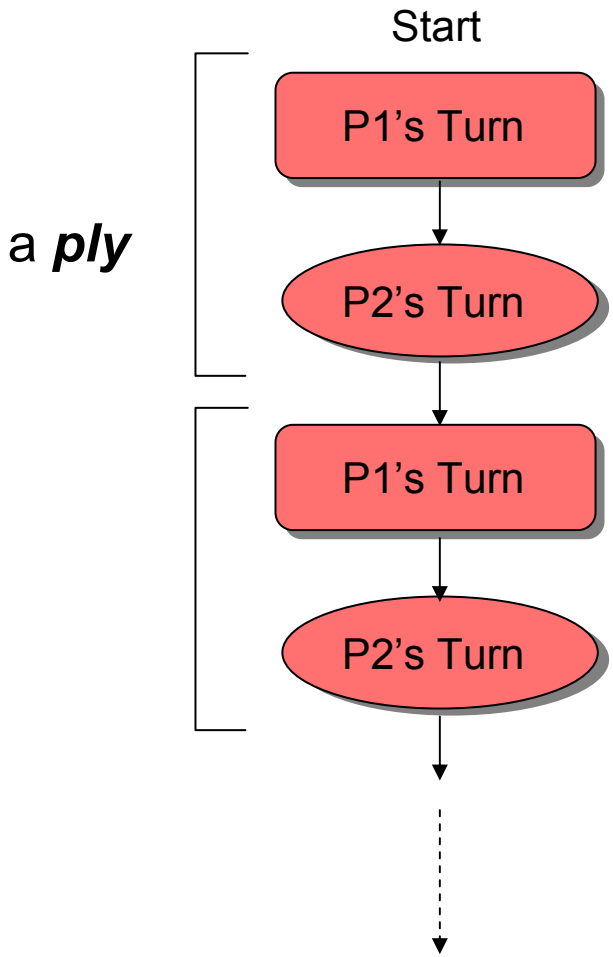
Optimal Play? How?

- We know how to search optimally to reach goal.
- But now:

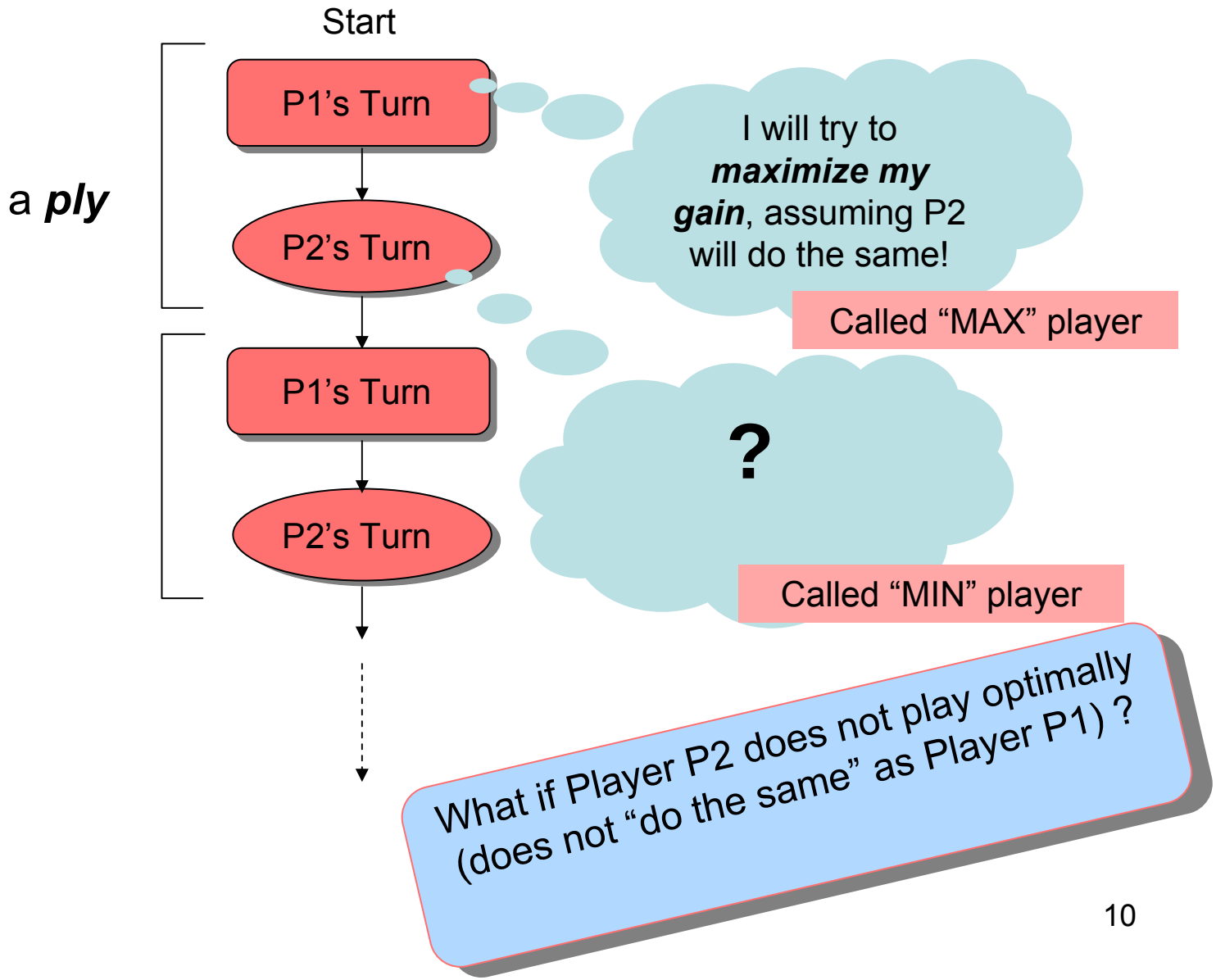
An opponent has something to say!!

- Player P1 must find a ***strategy***!

Optimal Play? How?

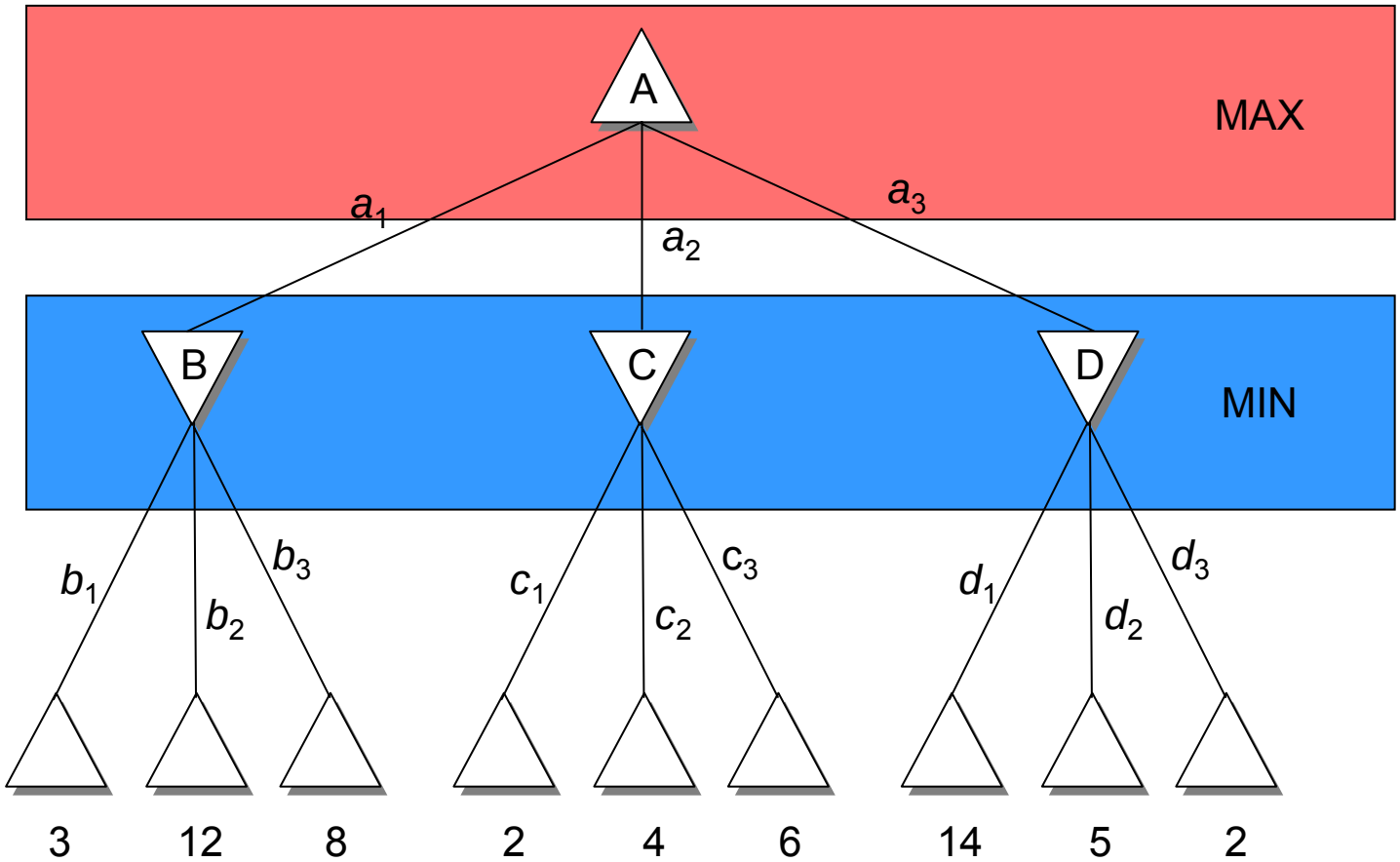


Optimal Play? How?



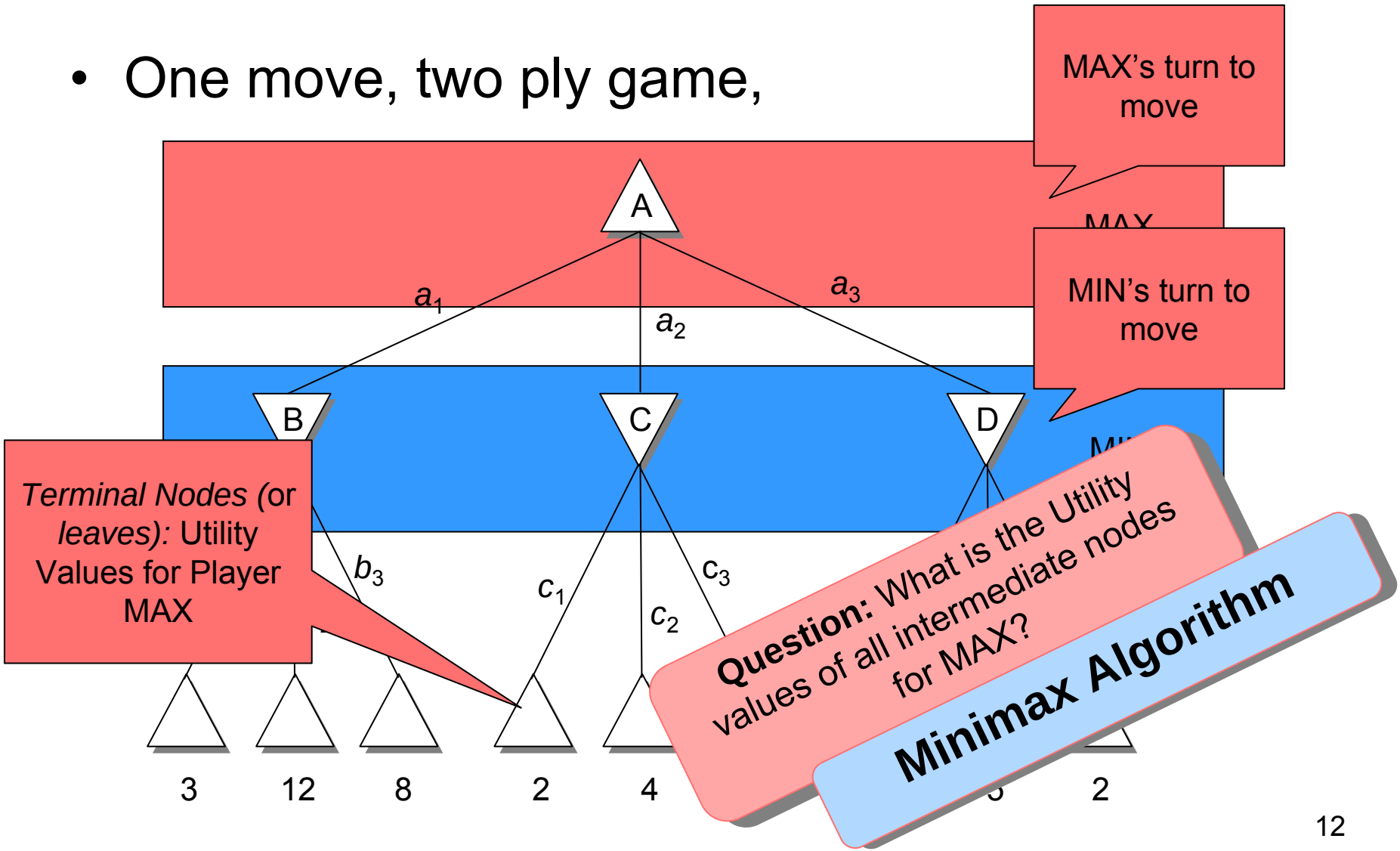
Example: A one move game

- One move, two ply game,



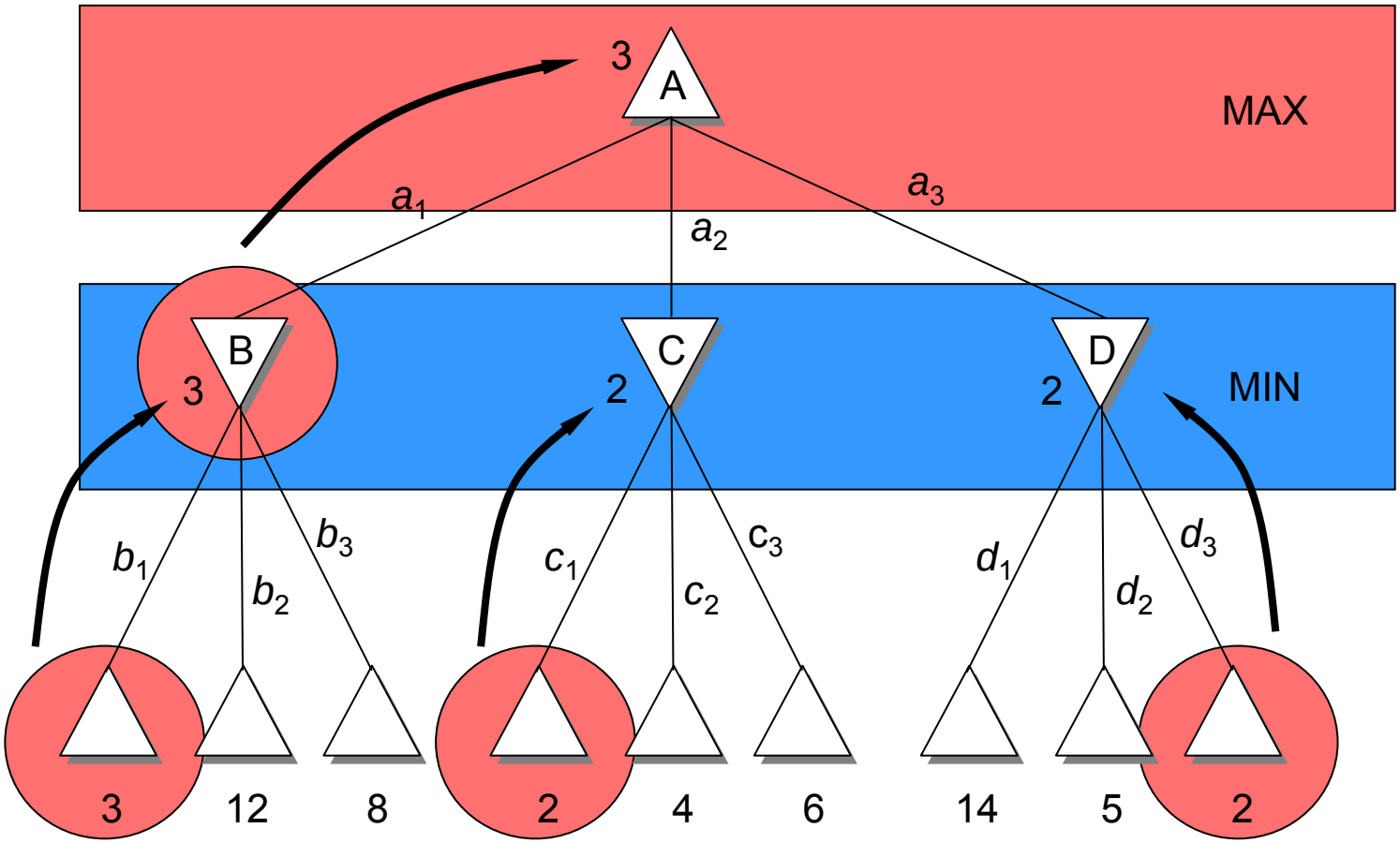
Example: A one move game

- One move, two ply game,



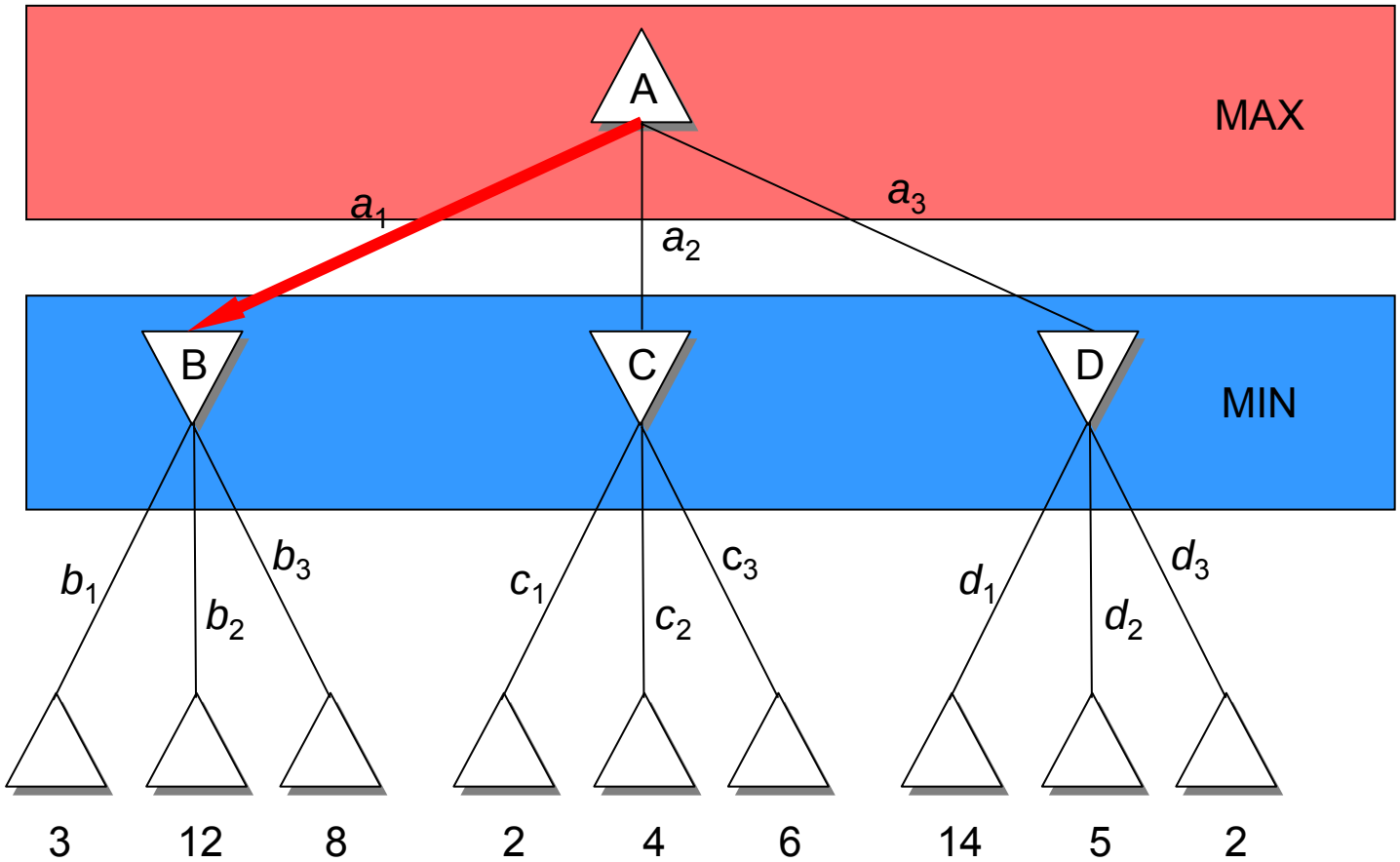
Example: A one move game

- Minimax values (utility for MAX) for all nodes



Example: A one move game

- Finally: Optimal Move for Player P1 (or “MAX”)



Minimax Algorithm

- BAD: For branching factor *b* and *D* moves in the game: Exponential time and space $O(b^D)$!!
- But we can avoid storing the entire tree.
- How? Ideas?

Minimax Algorithm – Multiplayer?

- How can minimax be extended to the multi-player case?

Recursive Minimax Algorithm

- A recursive procedure to compute Minimax values for each node in the game tree

MinimaxValue(S)=
    If (S is a terminal)
        return V(S)
    Else
        Let { S_1, S_2, … S_k } = Succs(S)
        Let v_i = MinimaxValue(S_i) for each i
        If Player-to-move(S) = A
            return $\max_{i \in \{1,2 \ldots k\}} V_i$
        else
            return $\min_{i \in \{1,2 \ldots k\}} V_i$

Recursive Algorithms Refresher

- Solving a part of the problem requires exactly the same procedure as solving the original problem
- But: Starting at different place with different values,
- Then: Perform recursive calls to the same function,
- Requires a termination criteria to end the procedure!!
- **Example: Compute the factorial**

$$5! = 5 * 4 * 3 * 2 * 1$$
$$Fact(5) = 5 * Fact(4)$$

Function FACT(*n*) **returns** a number
inputs: *n*, number to compute factorial of

if (n=1) **then return** 1
else return n * FACT(n-1)

Recursive Minimax Algorithm

- Questions to think about

```
MinimaxValue(S)=
        If (S is a terminal)
                return V(S)
        Else
                Let { S1, S2, … Sk } = Succs(S)
                Let vi = MinimaxValue(Si) for each i
                If Player-to-move(S) = A
                        return  max_{i∈{1,2…k}} Vi
                else
                        return  min_{i∈{1,2…k}} Vi
```

- What if there are loops possible in the game?

- This is a depth-first search algorithm.
  Would a breadth-first version be possible?
  How would it work?

Recursive Minimax Algorithm

- Questions to think about

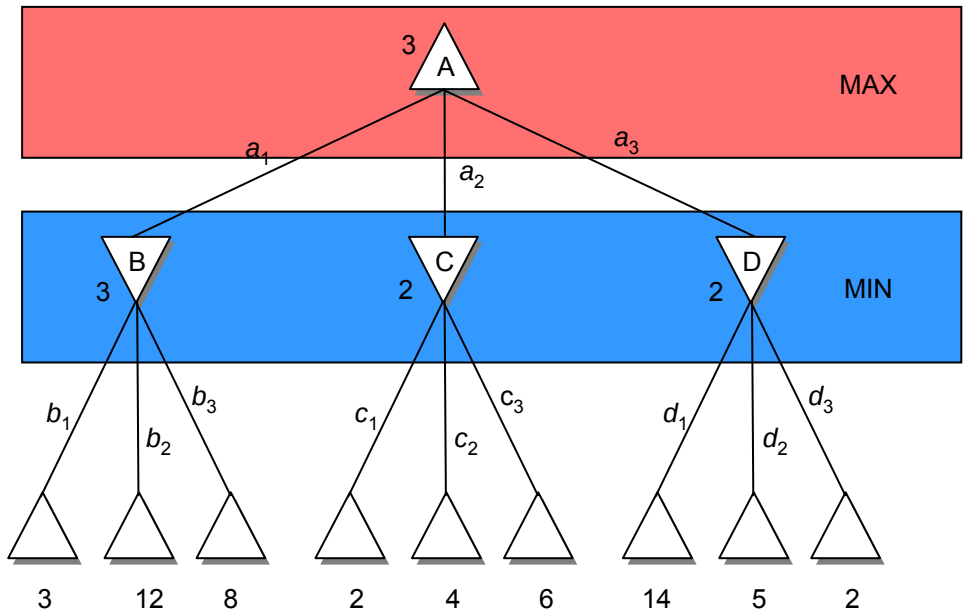
```
MinimaxValue(S)=
    If (S is a terminal)
        return V(S)
    Else
        Let { S_1, S_2, … S_k } = Succs(S)
        Let v_i = MinimaxValue(S_i) for each i
        If Player-to-move(S) = A
            return  max_{i∈{1,2…k}} V_i
        else
            return  min_{i∈{1,2…k}} V_i
```

- This is a depth-first search algorithm. Would a breadth-first version be possible? How would it work?

- What if there are loops possible in the game?
  - Is our recursive-minimax guaranteed to succeed?
  - Is our recursive-minimax guaranteed to fail?
  - What problems do loops cause for our definition of minimax value (i.e. game-theoretic value)?
  - How could we fix our recursive minimax program?

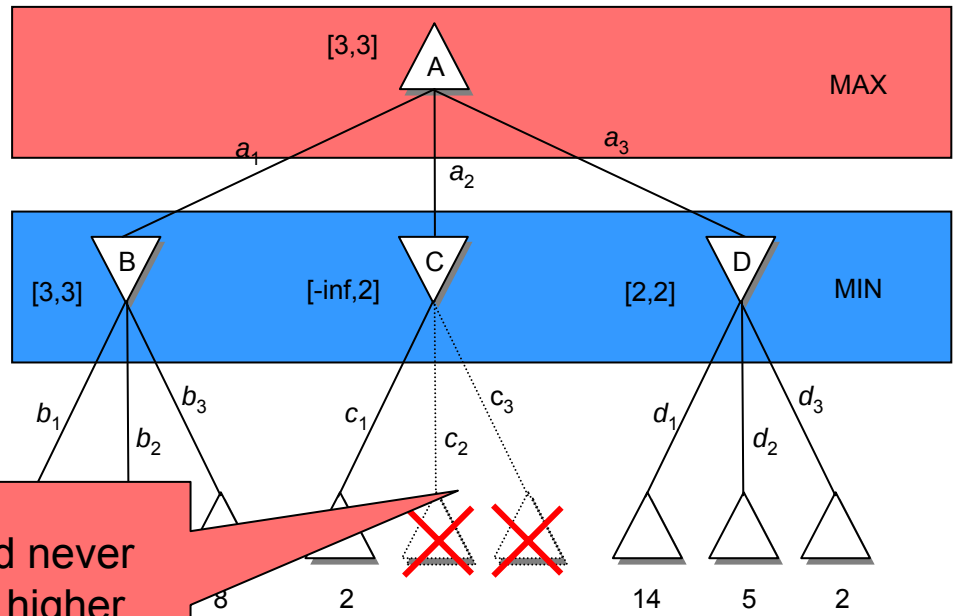
The Minimax Algorithm

- Is there cases when we can save from expanding some nodes?
- When?
- We can trace what we know as the Algorithm progresses.



Alpha-Beta Pruning

- Pruning is to cut-away branches because they are not needed for the solution
- We can prune when we can ***conclude that a branch would never be selected***

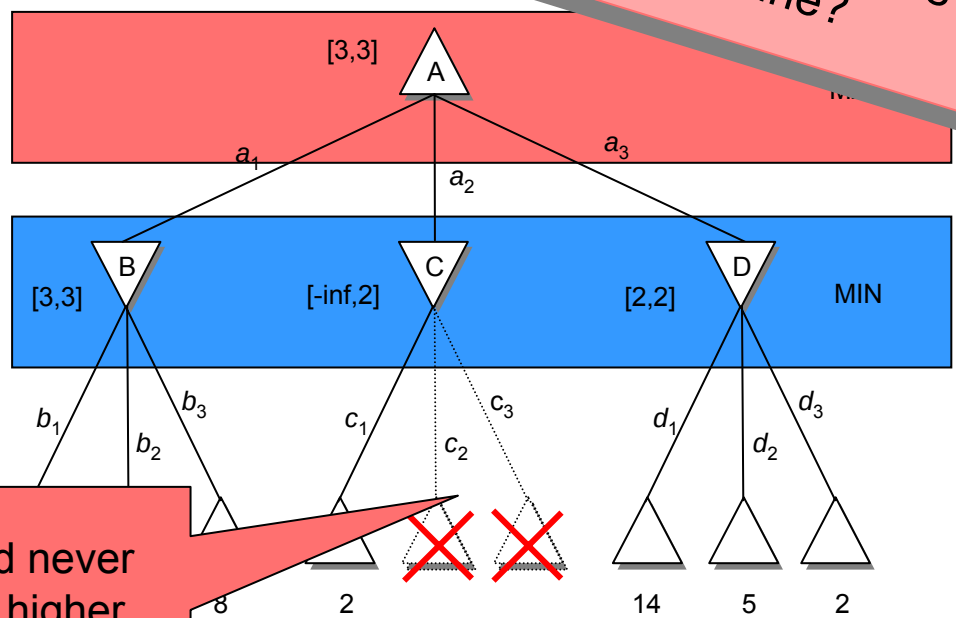


Since MIN would never select a branch higher than 2, ***MAX would never select branch C!!***

Alpha-Beta Pruning

- Pruning is to cut-away branches because they are not needed for the solut
- We can prune when we can ***conclude that a branch would never be selected***

Question: Is it enough to consider only the parent for deciding whether to prune?



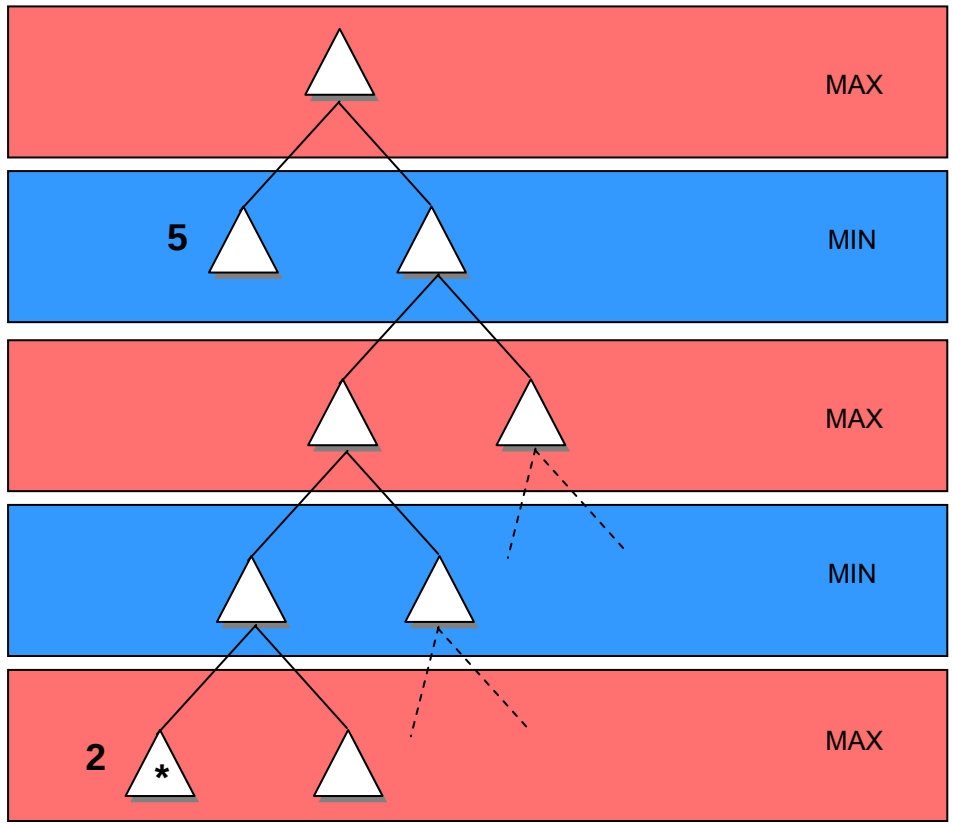
Since MIN would never select a branch higher than 2, ***MAX would never select branch C!!***

Ancestor Causing a Cut-Away

- Suppose we've so far done a full depth first search, expanding left-most successors first, and have arrived at the node marked * (and discovered its value is 2).
- What can we cut off in the rest of the search, and why?

- ***General rule:***

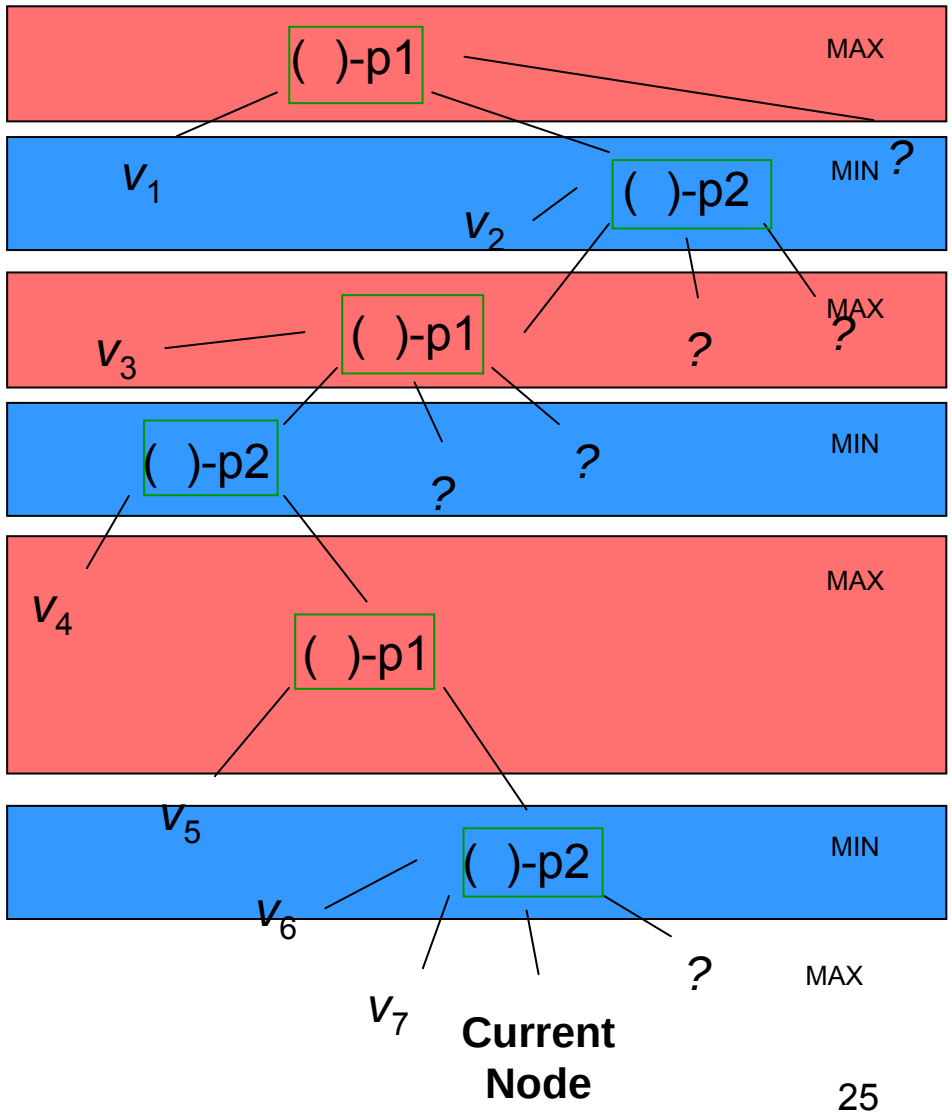
  We can be sure a node will not be visited if we're sure that either player has a better alternative at any ancestor of that node.



The general Cut-Away rule

In example: let α = max(v_1, v_3, v_5). If min(v_6, v_7)≤α, then we can be certain that it is worthless searching the tree from the current node or the sibling on its right.

In general: if at a B-move node, let α = max of all A's choices expanded on current path. Let β = min of B's choices, including those at current node. Cutoff is β ≤ α.



How useful is Alpha-Beta Pruning?

- What is the best possible case performance of alpha beta? Suppose that you were very lucky in the order in which you tried all the node successors. How much of the tree would you examine?
- In the best case, the number of nodes you need to search in the tree is $O(b^{d/2})$… i.e., we have half the exponent

Question:

- What can we do about large realsized games with huge numbers of states (e.g. chess)?

Game Solving vs Game Playing

- Two very different activities.
- So far, we have been solely concerned with finding the true minimax value of a state.
- But what do real chess-playing programs do?
- They have a couple of interesting features that the search problems we've discussed so far don't have:
  - They cannot possibly find guaranteed solution.
  - They must make their decisions quickly, in real time.
  - It is not possible to pre-compute a solution.
- The very popular solution to these problems are the ***heuristic evaluation functions for games***.

Evaluation Functions for Games

An evaluation function maps states to a number. The larger the number, the larger the estimate for the true minimax value.

- Search a tree as deeply as affordable.
- Leaves of the tree you search are not leaves of the game tree, but are rather intermediate nodes.
- The value assigned to the leaves are from the evaluation function.

Intuitions

Visibility: the evaluation function will be more accurate nearer the end of the game, so worth using heuristic estimates from there.

Filtering: if we used the evaluation function without searching, we'd be using a handful of inaccurate estimates. By searching we are combining thousands of these estimates, & we hope, eliminating noise.

Shaky intuition. Counter-examples. But often works very well in real games.

Other Important Issues

- How to decide how far to search if you only have a fixed time to make a decision. What's a sensible answer?
- **Quiescence.** What if you stop the search at a state where subsequent moves dramatically change the evaluation?
    - The solution to the quiescence problem is a sensible technique called *quiescence search*.
- **The horizon problem.** What if *s* is a state which is clearly bad because your opponent will inevitably be able to do something bad to you? But you have some delaying tactics. The search algorithm won't recognize the state's badness if the number of delaying moves exceeds the search horizon.
- Endgames are easy to play well. How?
- Openings fairly easy to play well. How?

Solving Games: Can it be done?

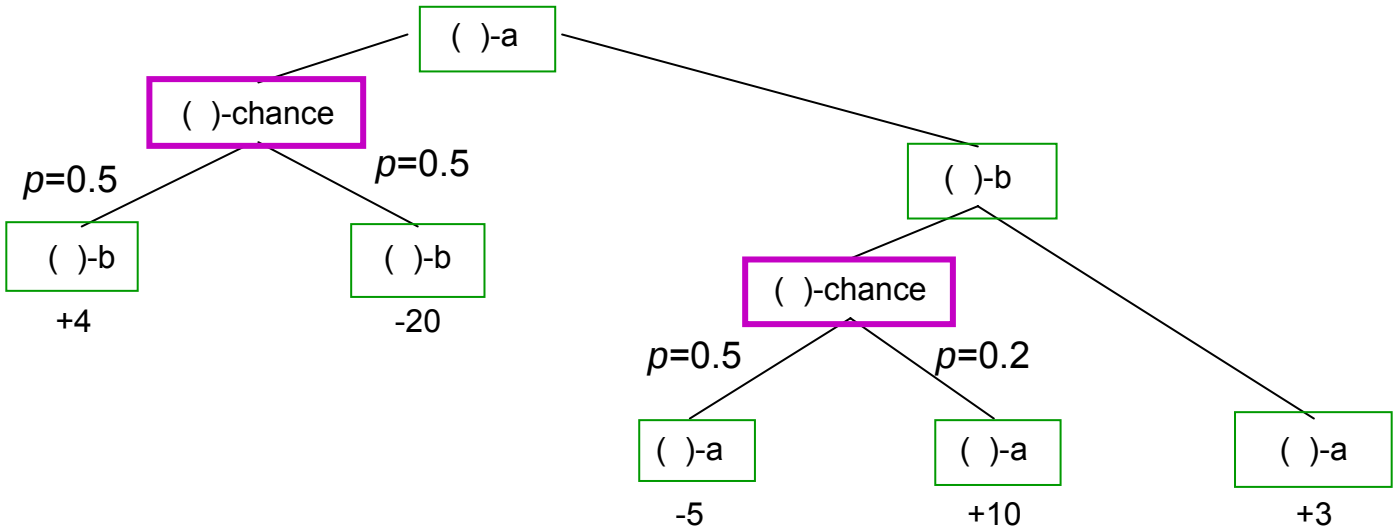
- Solving a game means proving the minimax value of the start state.
- Not possible for most of the games but some games have been solved.
- Work back from end of game, to create an end-game database, in combination with alpha-beta search from the start of the game.
- Checkers may not be far from being solved.

- ***Solving a game is often very different from playing well at the game.***

2 player, zero-sum, finite, non-deterministic games of perfect information

Nondeterministic = stochastic

The search tree now includes states where neither player makes a choice, but instead a random decision is made according to a known set of outcome probabilities.



ExpectiMinimax value of a state is the expected final value if both players are optimal.

If no loops, computing this is almost as easy as recursive minimax. Is there alpha-beta version?

Reading Assignment

Read “State-of-the-Art Game Programs” and Discussion sections from Chapter 6. Russel & Norvig.