Outline

Adversarial Search

Optimal decisions

Minimax

α-β pruning

Case study: Deep Blue

UCT and Go

Adversarial Reasoning: Games

Mathematical Game Theory

Branch of economics that views any multi-agent environment as a game, provided that the impact of each agent on the others is "significant", regardless of whether the agents are cooperative or competitive.

First step:

- Deterministic
- Turn taking
- 2-player
- Zero-sum game of perfect information (fully observable)
  "my win is your loss" and vice versa; utility of final states opposite for each player. My +10 is your -10.

Game Playing vs. Search

Multi-agent game vs. single-agent search problem

"Unpredictable" opponent need a strategy: specifies a move
for each possible opponent reply.
E.g with “huge” lookup table.

Time limits ✉ unlikely to find optimal response, must
approximate

Rich history of game playing in AI, in particular in the area of chess.

Both Turing and Shannon viewed chess as an important challenge for machine intelligence because playing chess appears to require some level of intelligence.

Human-computer hybrid most exciting new level of play. Computers as smart assistants are becoming accepted.
Area referred to as "Assisted Cognition."

Why is Game-Playing a Challenge for AI?

- Competent game playing is a mark of some aspects of “intelligence”
  - Requires planning, reasoning and learning
- Proxy for real-world decision making problems
  - Easy to represent states & define rules
  - Obtaining good performance is hard
- “Adversary” can be nature
- PSPACE-complete (or worse)
  - Computationally equivalent to hardware debugging, formal verification, logistics planning
  - PSPACE believed to be harder than NP.

Traditional Board Games

Finite

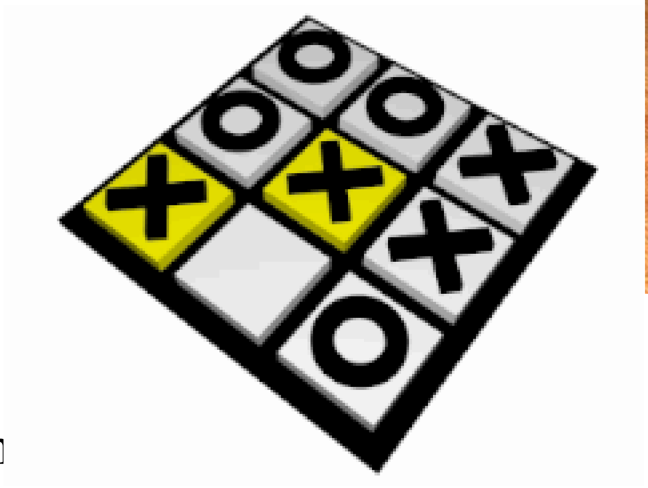
Two-player

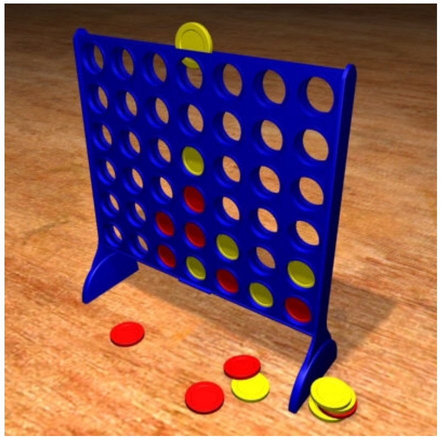
Zero-sum

Deterministic

Perfect Information

Sequential



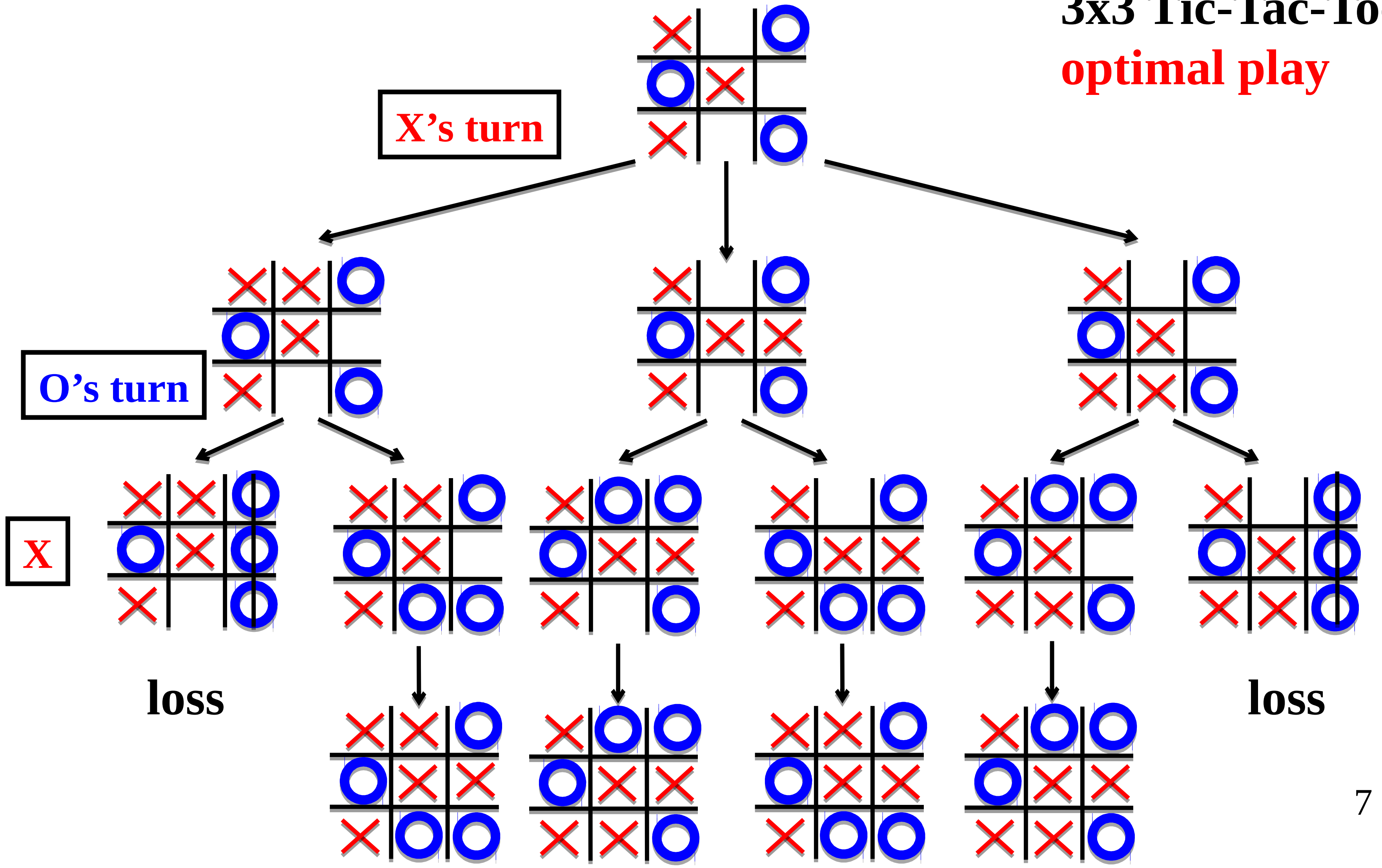


Tic-tac-toe (or Noughts and crosses, Xs and Os)

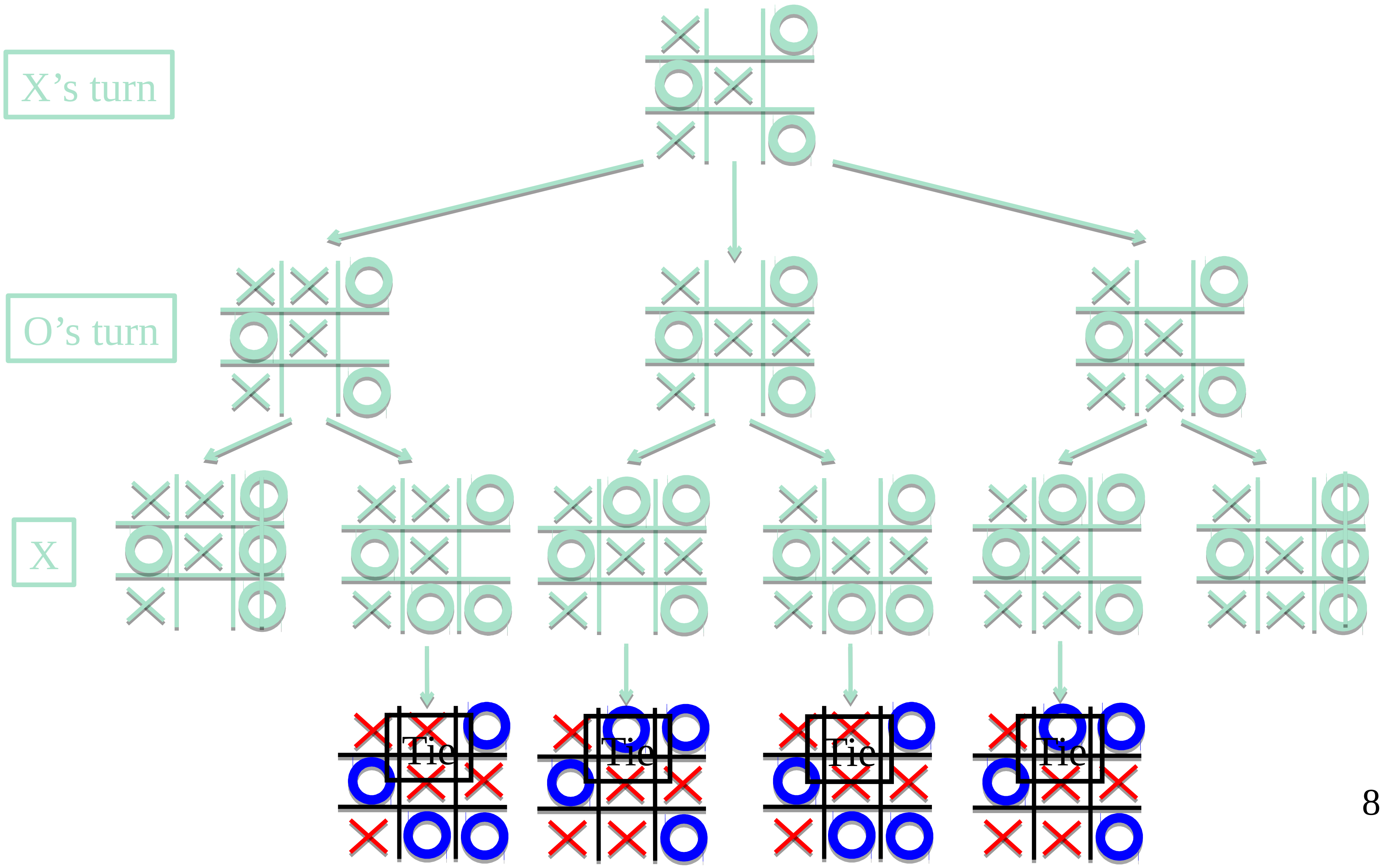
Key Idea: Look Ahead

We start 3 moves per player in:

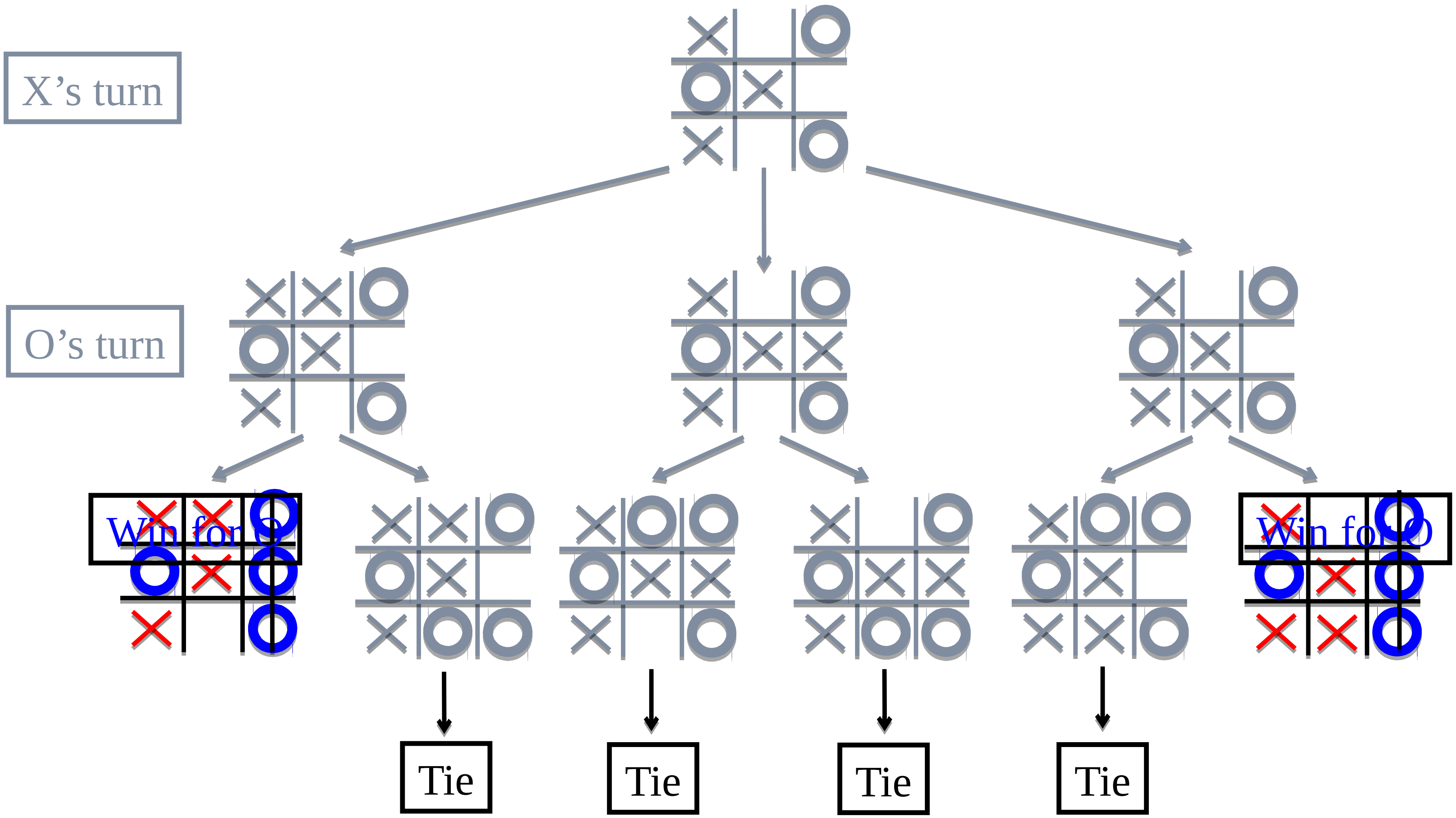
3x3 Tic-Tac-Toe
optimal play



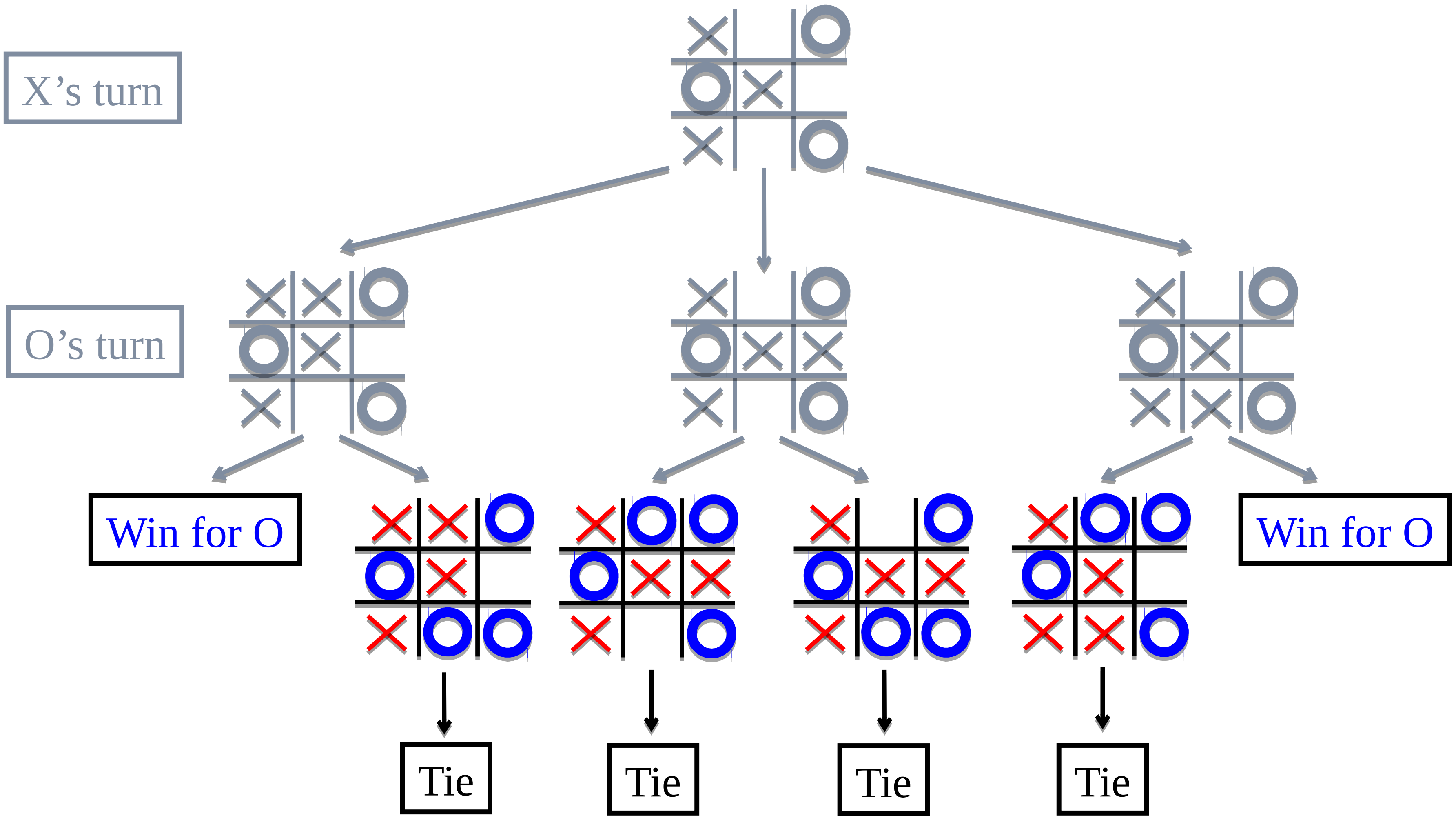
Look-ahead based Tic-Tac-Toe



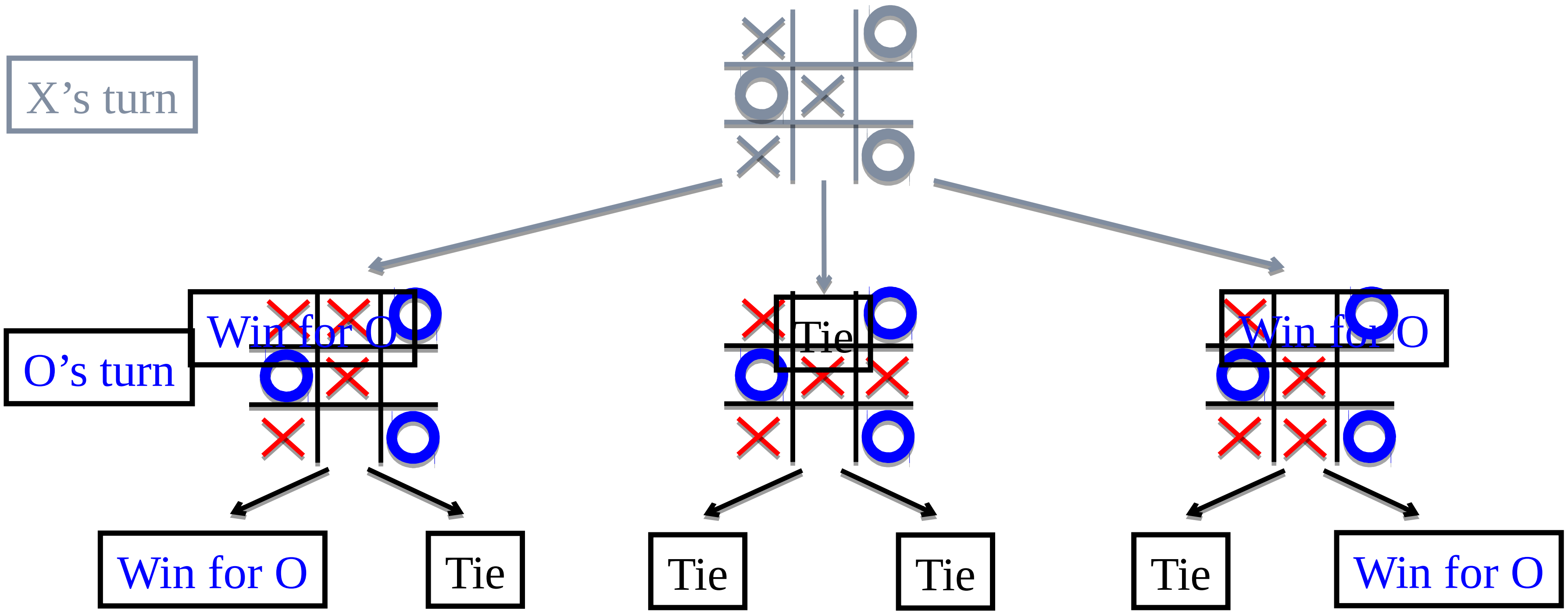
Look-ahead based Tic-Tac-Toe



Look-ahead based Tic-Tac-Toe



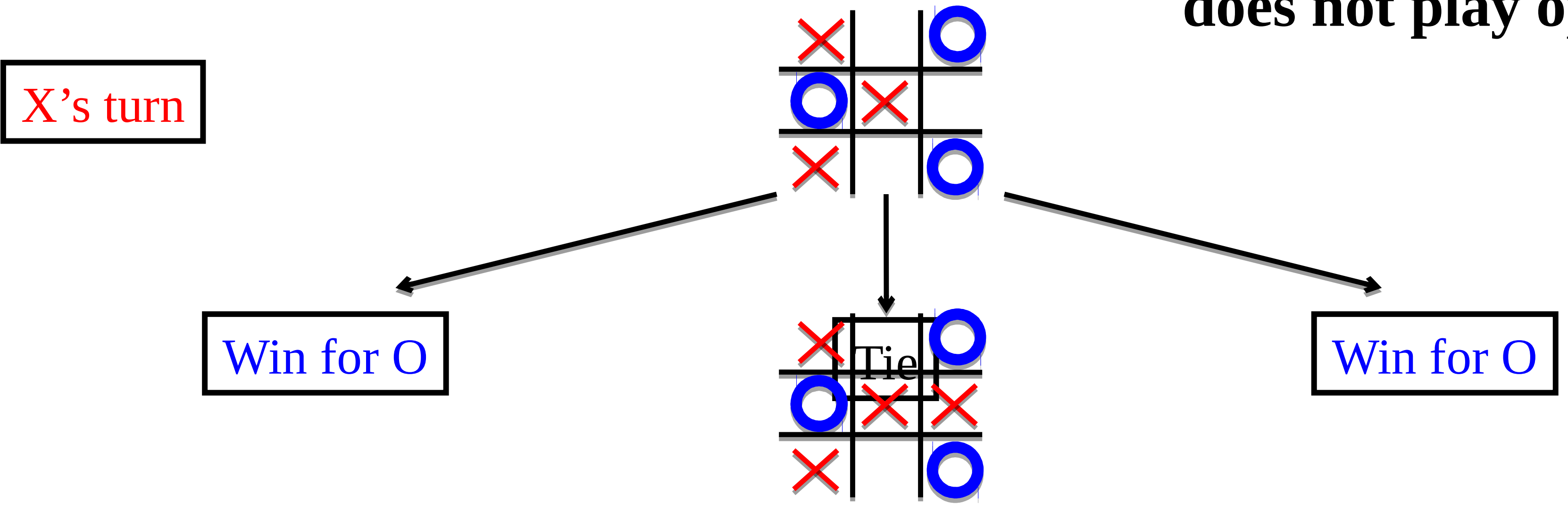
Look-ahead based Tic-Tac-Toe



Each board in game tree gets *unique* game tree value (utility; -1/0/+1) under optimal rational play. (Convince yourself.)

E.g. 0 for top board.

What if our opponent does not play optimally?



Approach: Look first at bottom tree. Label bottom-most boards.

Then label boards one level up, according result of **best possible move.**

… and so on. Moving up layer by layer.

Termed the Minimax Algorithm

– Implemented as a depth-first search

Aside: Game tree learning

Can (in principle) store all board values in large table. 3^19 = 19,683 for tic-tac-toe.

Can use table to try to train classifier to predict "win", "loss", or "draw."

Issue: For real games, one can only look at tiny, tiny fragment of table.

Reinforcement learning builds on this idea.

See eg Irvine Machine Learning archive.
*archive.ics.uci.edu/ml/**datasets/Tic-Tac-Toe**+Endgame*

Look-ahead based Chess

But there's a catch…

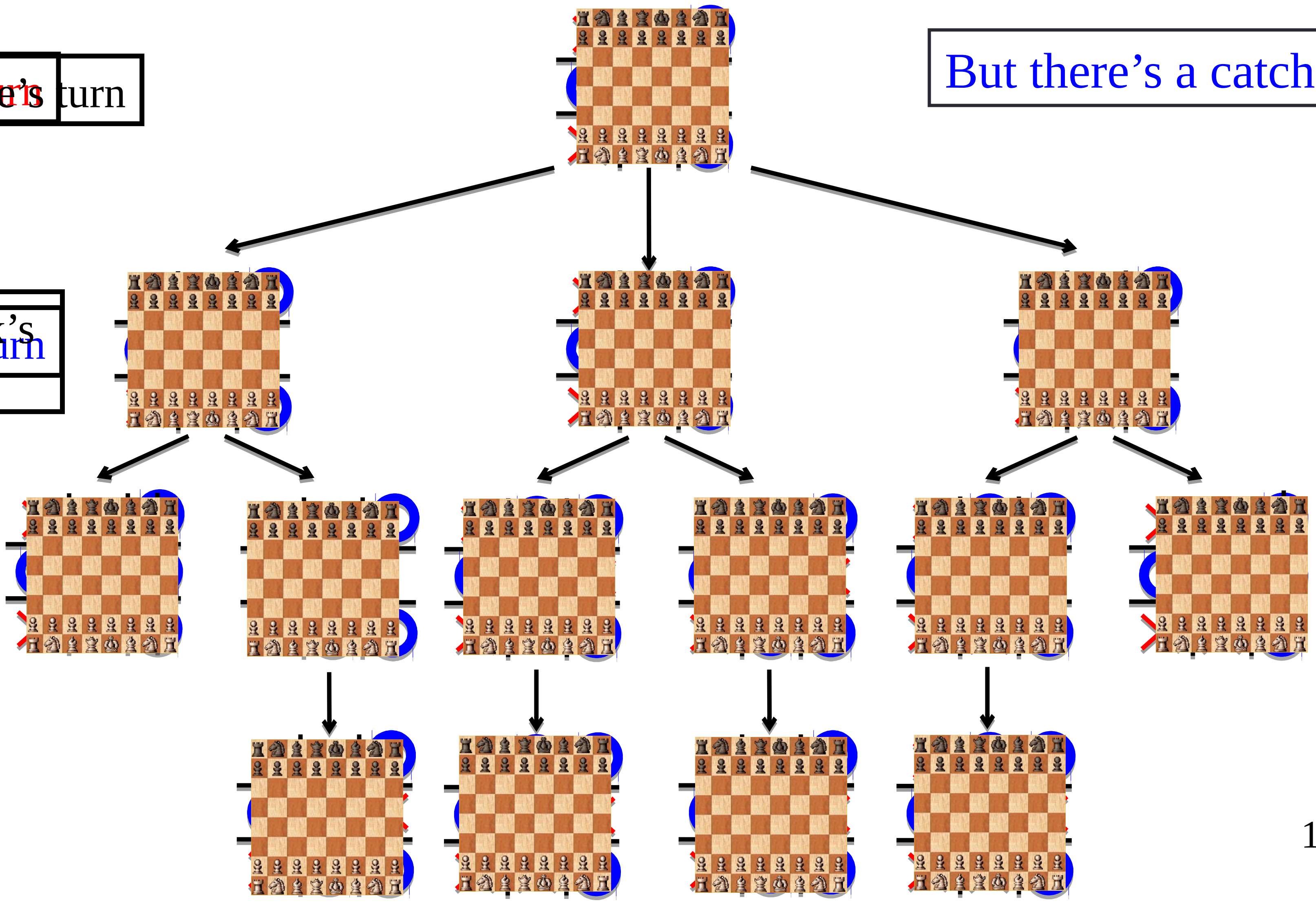
X's turn

White's turn

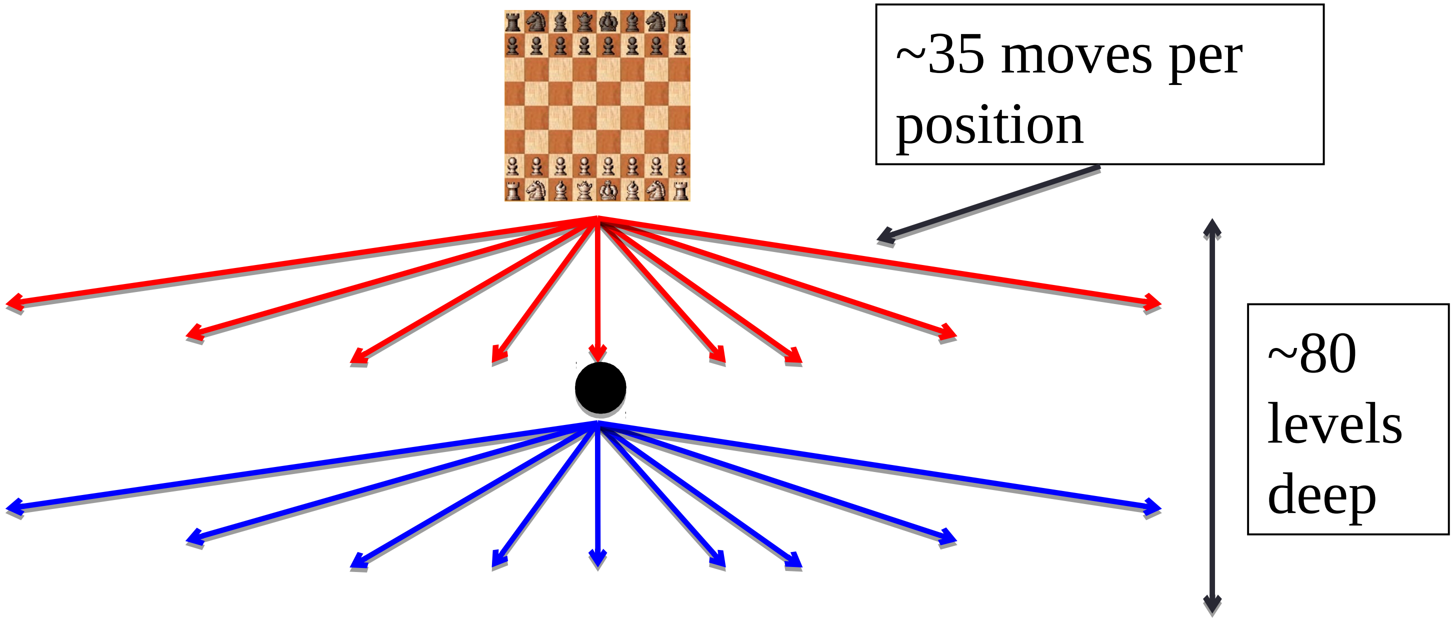
O's turn

Black's turn

X



How big is this tree?



Approx. 10^120 > Number of atoms in the observable universe (10^80)

We can really only search a **tiny, miniscule faction** of this tree!

Around 60 x 10^9 nodes for 5 minute move. **Approx. 1 / 10^70 fraction.**

What's the work-around?

Don't search to the very end

- Go down 10-12 levels (still deeper than most humans)
- But now what?
- Compute *an estimate of the position's value*
  - This heuristic function is typically designed by a domain expert

Consider a game tree with leaf utilities (final boards) +1 / 0 / -1 (or +inf / 0 –inf). What are the utilities of intermediate boards in the game tree?

+1 / 0 / -1
(or +inf / 0 / -inf)

The board heuristics is trying to *estimate* these values from a quick calculation on the board. Eg, considering material won/loss on chess board or regions captures in GO. Heuristic value of e.g. +0.9, suggests true value may be +1.

What is a problem for the board heuristics (or evaluation functions) at the beginning of the game?

(Consider a heuristics that looks at lost and captured pieces.)

What will the heuristic values be near the top?

Close to 0! Not much has happened yet….

Other issue: children of any node are mostly quite similar. Gives almost identical heuristic board values. Little or no information about the right move.

Solution: Look ahead. I.e., build search tree several levels deep (hopefully 10 or more levels). Boards at bottom of tree more diverse. ***Use minimax search to determine value of starting board, assuming optimal play for both players.***

Intriguing aside: What is the formal computational complexity of chess? Use Big-O notation.

KASPAROV VERSUS DEEP BLUE

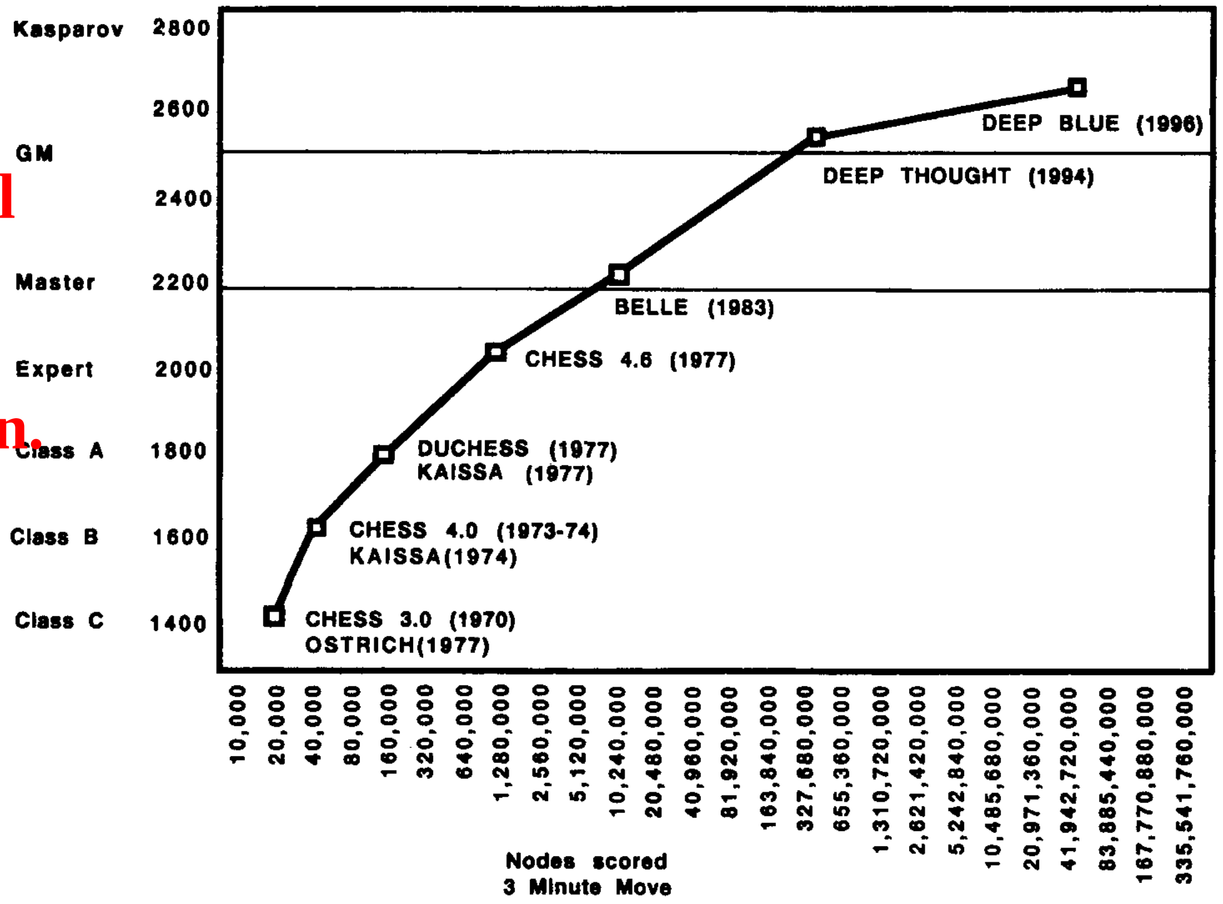


Figure 6.23. Relationship between the level of play by chess programs

IBM knew this when they "acquired" the Deep Thought team. They could predict what it would take to beat Kasparov.

Will deeper search give stronger play? Always? And why?

Very counterintuitive: there are "artificial games" where searching deeper leads to worse play! (Nau and Pearl 1980) Not in natural games! Game tree anomaly.

Heuristic board eval value is sometimes informally referred to as the "chance of winning" from that position.

That's a bit odd, because in a deterministic game with perfect information and optimal play, there is no "chance" at all! Each board has a fixed utility:
-1, 0, or +1 (a loss, draw, or a win). (result from game theory)

Still, "chance of winning" is an informally useful notion. But, remember, no clear semantics to heuristic values.

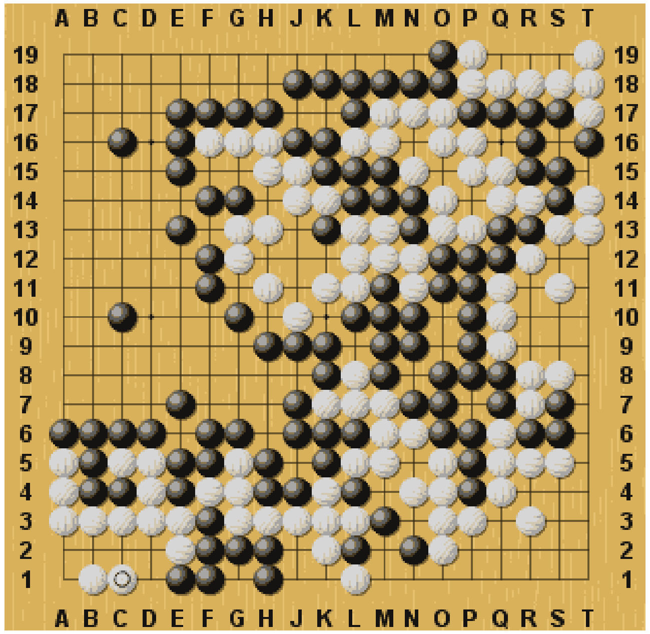
What if board eval gives true board utility? How much search is needed to make a move?
We'll see that using machine learning and "self play," we can get close for backgammon.

Limitations?

Two important factors for success:

- Deep look ahead
- Good heuristic function

Are there games where this is not feasible?



Limitations?

Two important factors for success:

- ~~Deep look ahead~~
- Good heuristic function

Are there games where this is not feasible?

Looking 14 levels ahead in Chess ≈ Looking 4 levels ahead in Go

Limitations?

Two important factors for success:

- ~~Deep look ahead~~
- ~~Good heuristic function~~

Are there games where this is not feasible?

Looking 14 levels ahead in Chess ≈ Looking 4 levels ahead in Go

Moves have extremely delayed effects

Limitations?

Two important factors for success:

- ~~Deep look ahead~~
- ~~Good heuristic function~~

Are there games where this is not feasible?

Looking 14 levels ahead in Chess ≈ Looking 4 levels ahead in Go

Moves have extremely delayed effects

Minimax players for GO were very weak until 2007…but then play at master level. Now, AlphaGo world champion.

Limitations?

Two important factors for success:

- ~~Deep look ahead~~
- ~~Good heuristic function~~

Are there games where this is not feasible?

Looking 14 levels ahead in Chess ≈ Looking 4 levels ahead in Go

Moves have extremely delayed effects

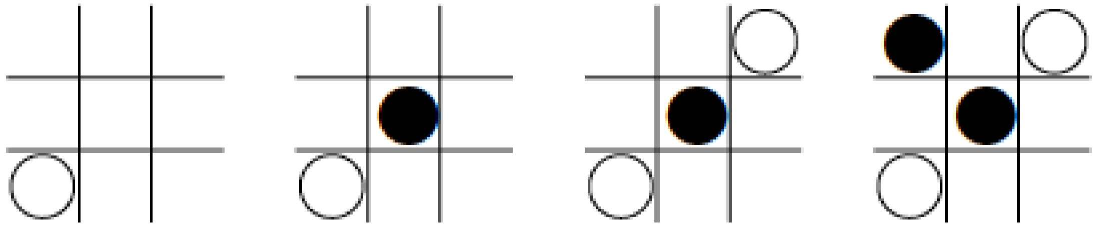
New sampling based search method:
Upper Confidence bounds applied to Trees (UCT)

Well… Why not use a strategy / knowledge, as humans do?

Consider for Tic-Tac-Toe:

1. If there is a winning move, make it.
2. If the opponent can win at a square by his next move, play that move.
3. Taking the central square is more important than taking other squares.
4. Taking corner squares is more important than taking squares on the edges.

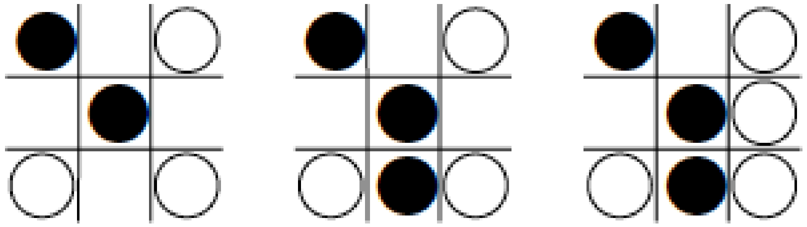
Rule 3 Rule 4



Oops!!

Consider Black uses the strategy…

Rule 2



So, although one can capture strategic knowledge of many games in high-level rules (at least to some extent), in practice any interesting game will revolve *precisely around the exceptions to those rules!*

Issue has been studied for decades but research keeps coming back to game tree search (or most recently, game tree sampling).

Currently only one exception: reinforcement learning for backgammon.

> **(discussed later)**
>
> **A very strong board evaluation function was learned in self-play.**
>
> **Represented as a neural net.**
>
> **Almost no search remained.**

Formal definition of a game:

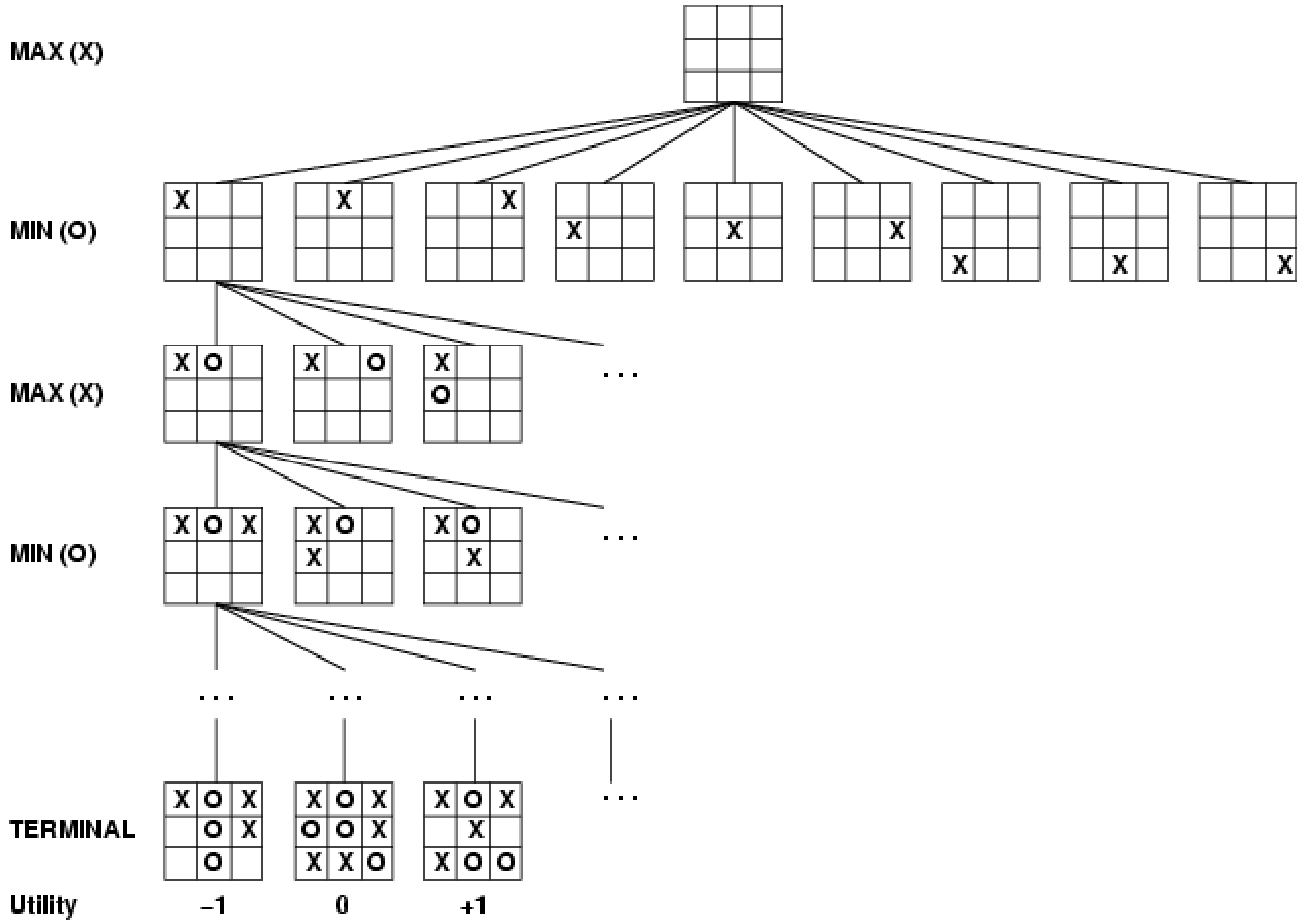
- **Initial state**
- **Successor function: returns list of *(move, state)* pairs**
- **Terminal test: determines when game over**

  Terminal states: states where game ends
- **Utility function (objective function or payoff function): gives numeric value for terminal states**

We will consider games with 2 players (**Max and Min**)

Max moves first.

Game Tree Example: Tic-Tac-Toe



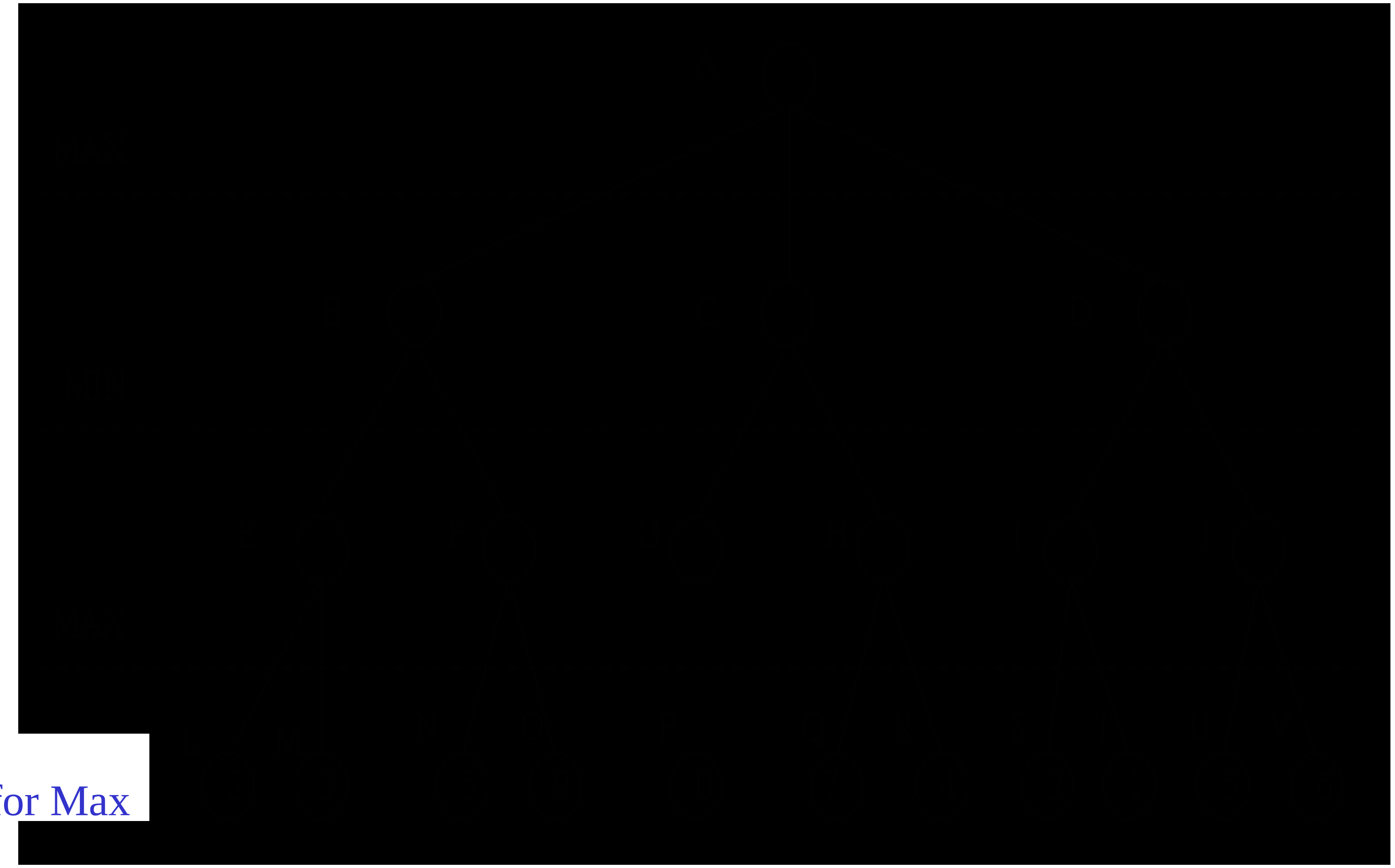
Tree from Max’s perspective

Minimax Algorithm

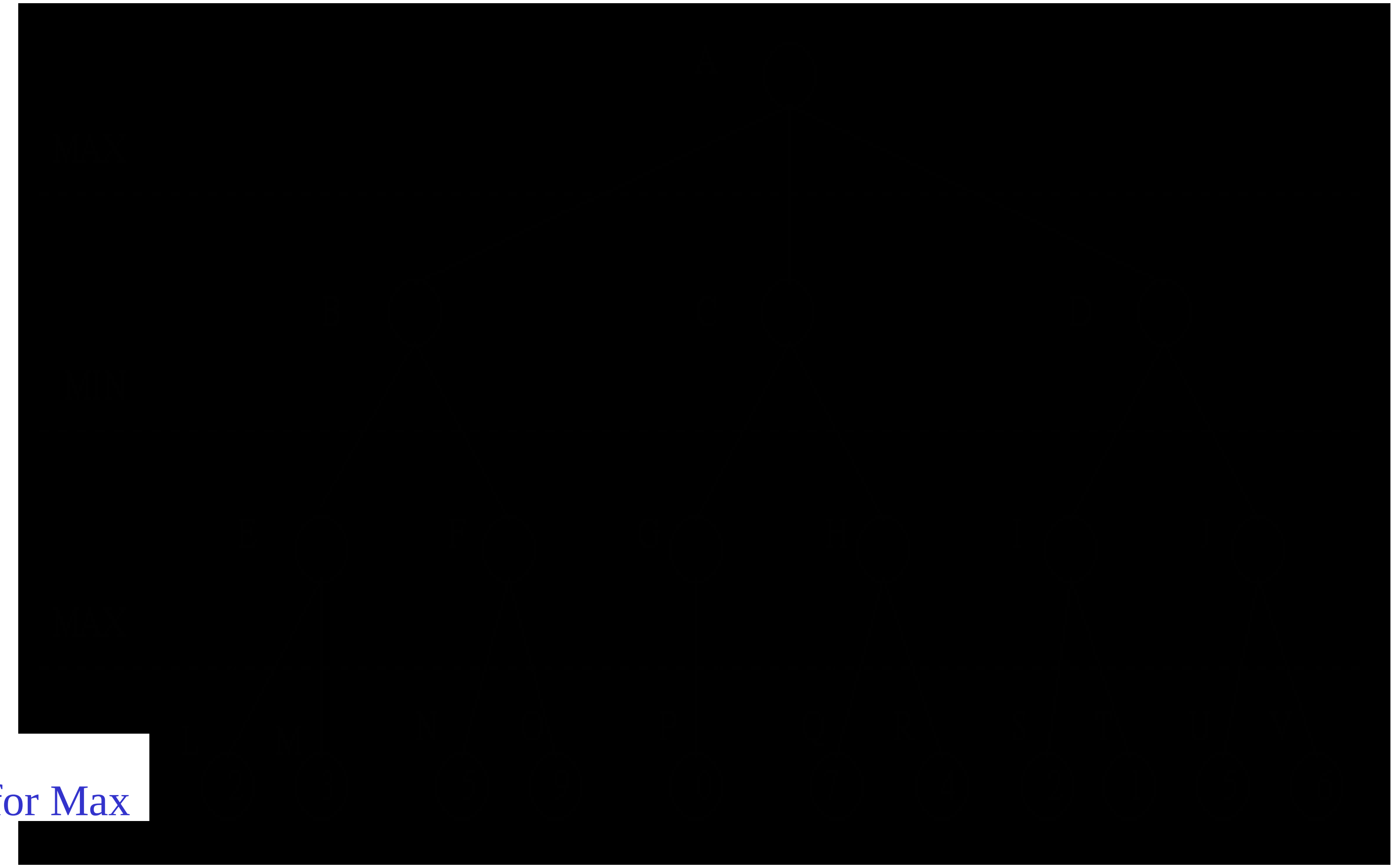
Minimax algorithm

- **Perfect play for deterministic, 2-player game**
- **Max tries to maximize its score**
- **Min tries to minimize Max's score (Min)**
- **Goal: Max to move to position with highest minimax value**
  - **Identify best achievable payoff against best play**

Minimax Algorithm



Minimax Algorithm (cont’d)



Payoff for Max

Minimax Algorithm (cont’d)

Payoff for Max

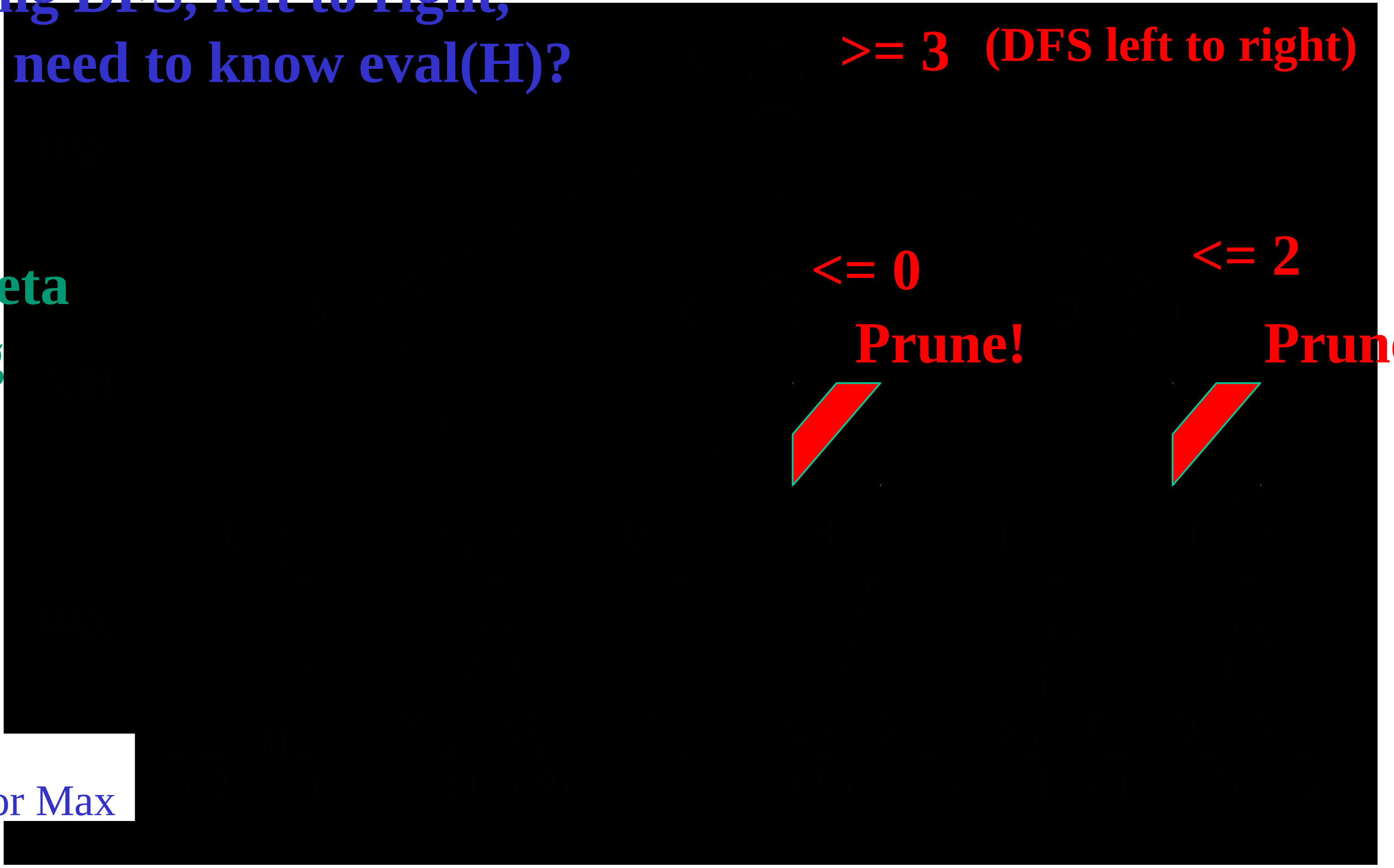
Minimax Algorithm

What if
payoff(Q) = 100
payoff(R) = 200

Do DFS. Real games:
use iterative deepening.
(gives "anytime" approach.)

Starting DFS, left to right,
do we need to know eval(H)?

alpha-beta
pruning



Payoff for Max

here

Properties of minimax algorithm:

Complete? Yes (if tree is finite)

Optimal? Yes (against an optimal opponent)

Time complexity? O(b^m)

Space complexity? O(bm) (depth-first exploration, if it generates all successors at once)

For chess, b ≈ 35, m ≈ 80 for "reasonable" games
✉ exact solution completely infeasible

m – maximum depth of the tree; b – legal moves

Minimax Algorithm

Limitations

- **Generally not feasible to traverse entire tree**
- **Time limitations**

Key Improvements

- **Use evaluation function instead of utility (discussed earlier)**
  - **Evaluation function provides estimate of utility at given position**

- **Alpha/beta pruning**

α-β Pruning

Can we improve search by reducing the size of the game tree to be examined?

Yes! Using alpha-beta pruning

Principle

- **If a move is determined worse than another move already examined, then there is no need for further examination of the node.**

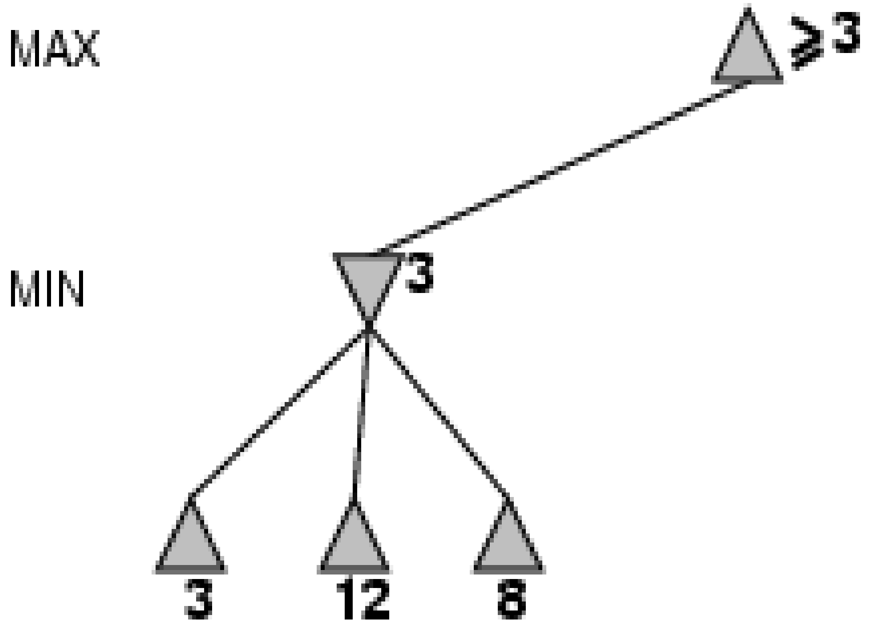
Analysis shows that will be able to search almost twice as deep.

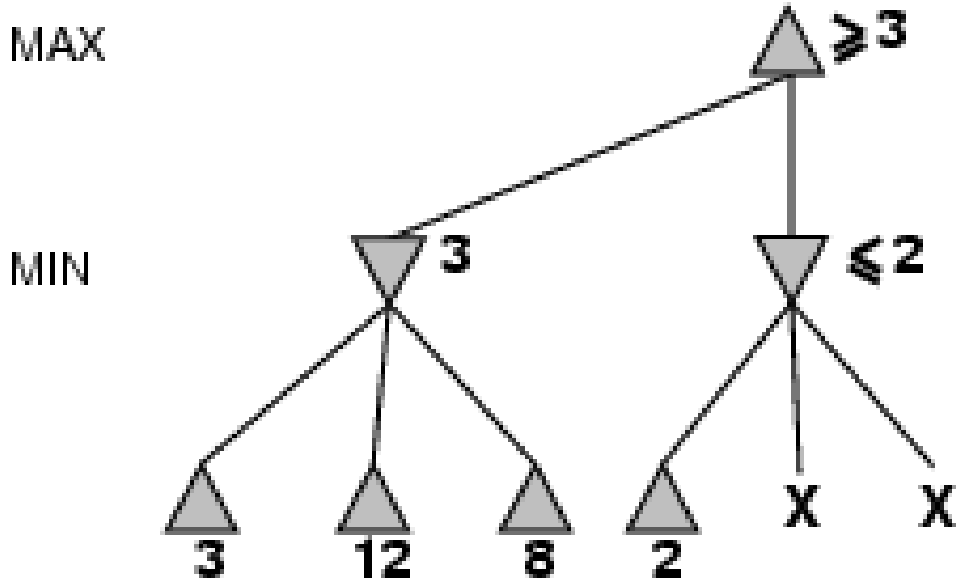
Really is what makes game tree search practically feasible.

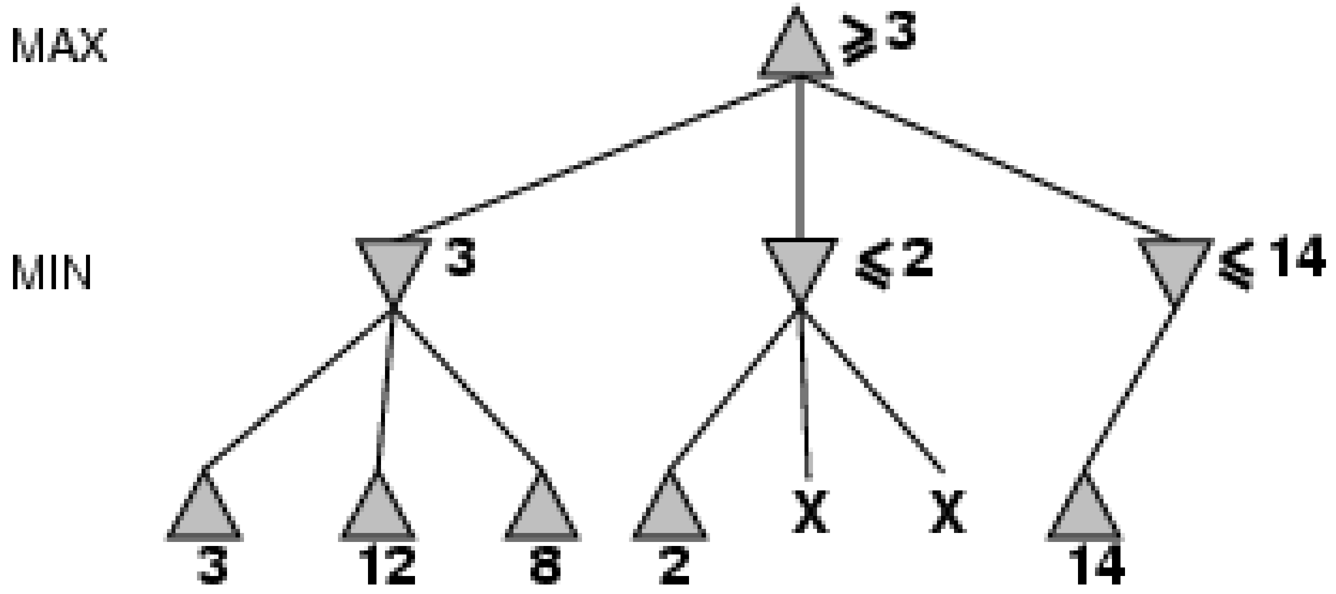
E.g. Deep Blue 14 plies using alpha-beta pruning.

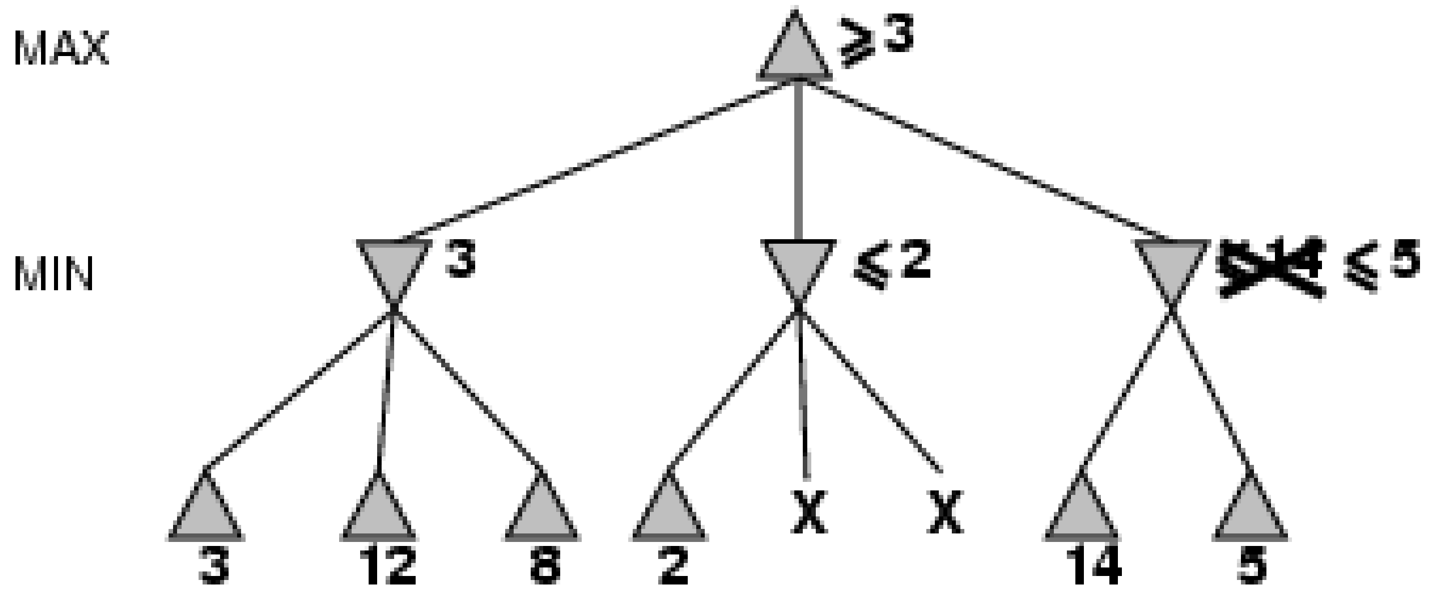
Otherwise only 7 or 8 (weak chess player). (plie = half move / one player)

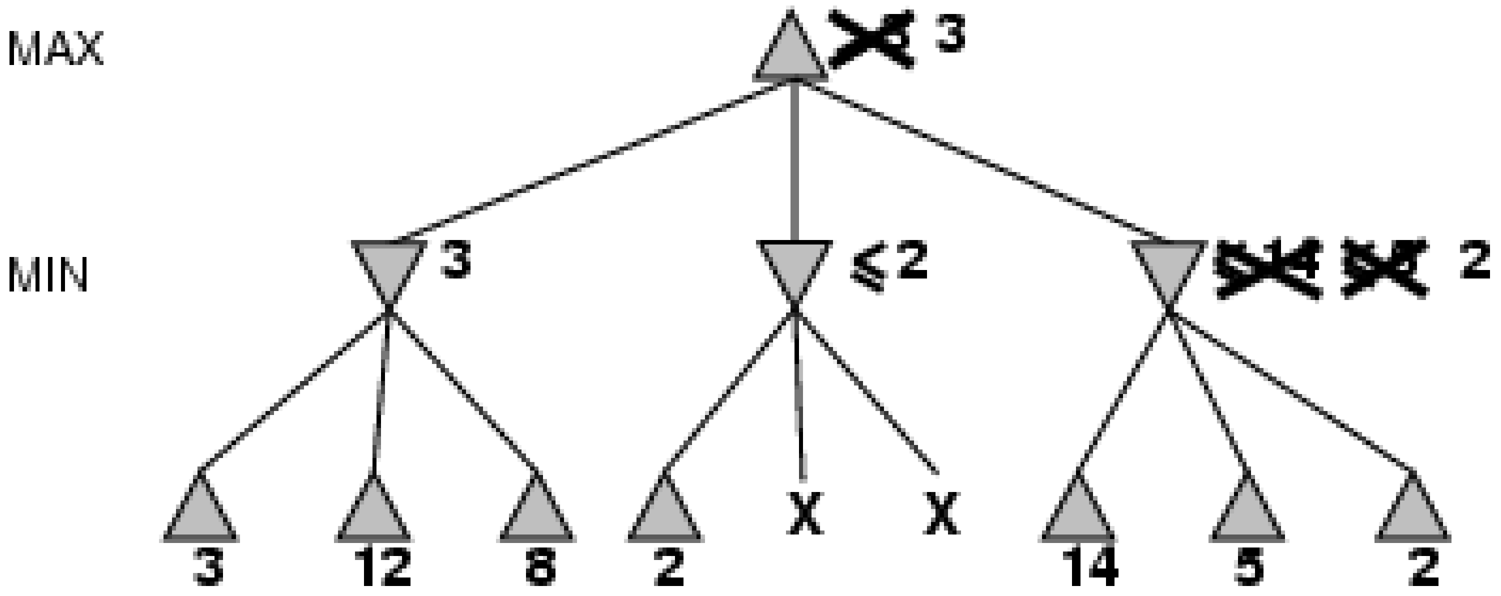
α-β Pruning Example




MAX
MIN
≥3
3
≤2
3
12
8
2
X
X


MAX
≥ 3
MIN
3
≤ 2
≤ 14
3
12
8
2
X
X
14


MAX
≥3
MIN
3
≤2
~~≤14~~ ≤5
3
12
8
2
X
X
14
5



Note: order children matters!

What gives best pruning?

Visit most promising (from min/max perspective) first.

Alpha-Beta Pruning

Rules:

- **α is the best (highest) found so far along the path for Max**
- **β is the best (lowest) found so far along the path for Min**
- **Search below a MIN node may be alpha-pruned if its β <= α of some MAX ancestor**
- **Search below a MAX node may be beta-pruned if its α >= β of some MIN ancestor.**

See also fig. 5.5 R&N.

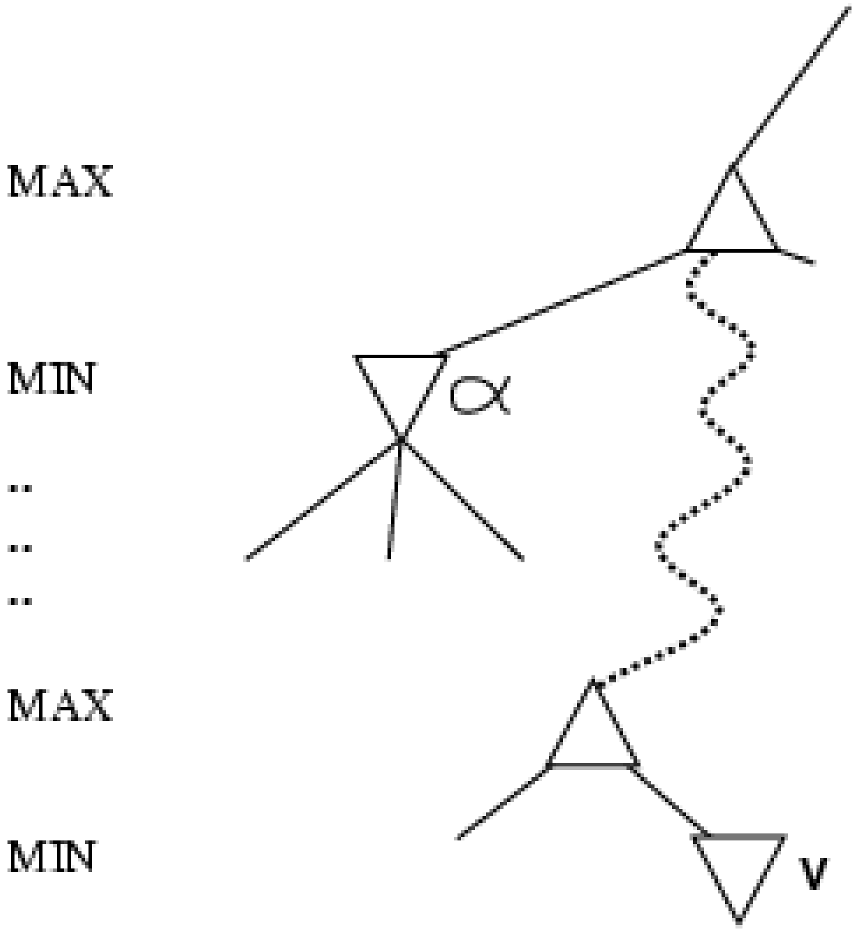
More abstractly

α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*

If *v* is worse than α, *max* will avoid it

- prune that branch

Define β similarly for *min*



Properties of α-β Prune

Pruning does not affect final result

Good move ordering improves effectiveness of pruning b(e.g., chess, try captures first, then threats, froward moves, then backward moves…)

With "perfect ordering," time complexity = O($b^{m/2}$)

- ✉ doubles depth of search that alpha-beta pruning can explore

Example of the value of reasoning about which computations are relevant (a form of metareasoning)

A few quick approx. numbers for Chess:

b = 35

200M nodes / second ===> 5 mins = 60 B nodes in search tree

(2 M nodes / sec. software only, fast PC ===> 600 M nodes in tree)

35^7 = 64 B

35^5 = 52 M

So, basic minimax: around 7 plies deep. (5 plies)

With, alpha-beta 35^(14/2) = 64 B. Therefore, 14 plies deep. (10 plies)

Aside:
4-ply ≈ human novice
8-ply / 10-ply ≈ typical PC, human master
14-ply ≈ Deep Blue, Kasparov (+ depth 25 for "selective extensions") / 7 moves by each player.

Resource limits

Can't go to all the way to the "bottom:"

evaluation function
= estimated desirability of position

cutoff test:
e.g., depth limit
(Use Iterative Deepening)

What is the problem with that?

Horizon effect.

"Unstable positions:"
Search deeper.
Selective extensions.
E.g. exchange of several pieces in a row.

→ **add quiescence search:**
→ **quiescent position: position where next move unlikely to cause large change in players' positions**

Evaluation Function

- Performed at search cutoff point
- Must have same terminal/goal states as utility function
- Tradeoff between accuracy and time → reasonable complexity
- Accurate
  - Performance of game-playing system dependent on accuracy/goodness of evaluation
  - Evaluation of nonterminal states strongly correlated with actual chances of winning

Evaluation functions

For chess, typically linear weighted sum of features

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

e.g., $w_1 = 1$ with

$f_1(s)$ = (number of white pawns) – (number of black pawns), etc.

Key challenge – find a good evaluation features:

- **Not just material! (as used by novice)**
- **Isolated pawns are bad.**
- **How well protected is your king?**
- **How much maneuverability to you have?**
- **Do you control the center of the board?**
- **Strategies change as the game proceeds**

Features are a form of chess knowledge. Hand-coded in eval function.

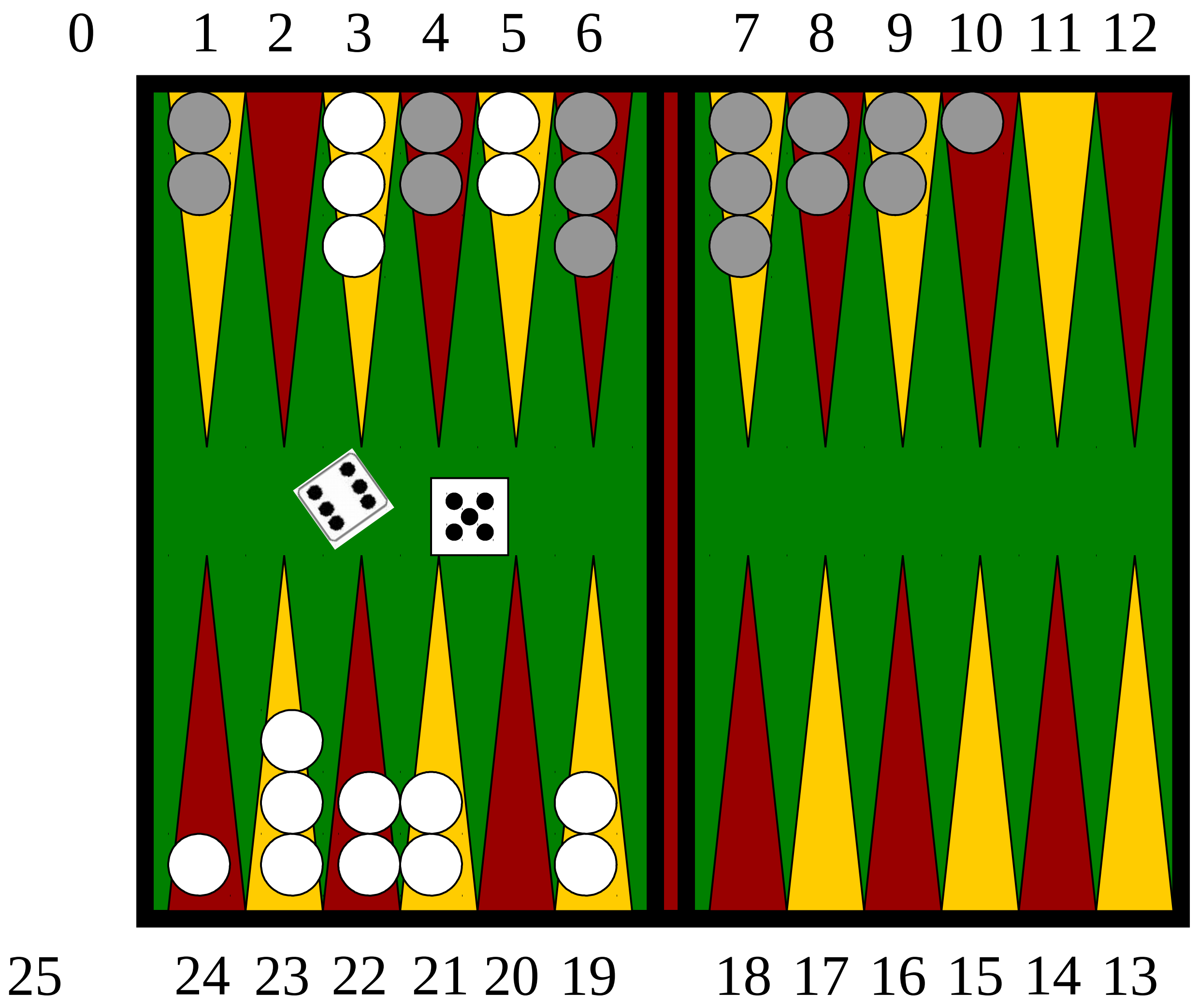
Knowledge tightly integrated in search.

Feature weights: can be automatically tuned (“learned”). **How?**

Standard issue in machine learning:

Features, generally hand-coded; weights tuned automatically.

When Chance is involved:
Backgammon Board



Expectiminimax

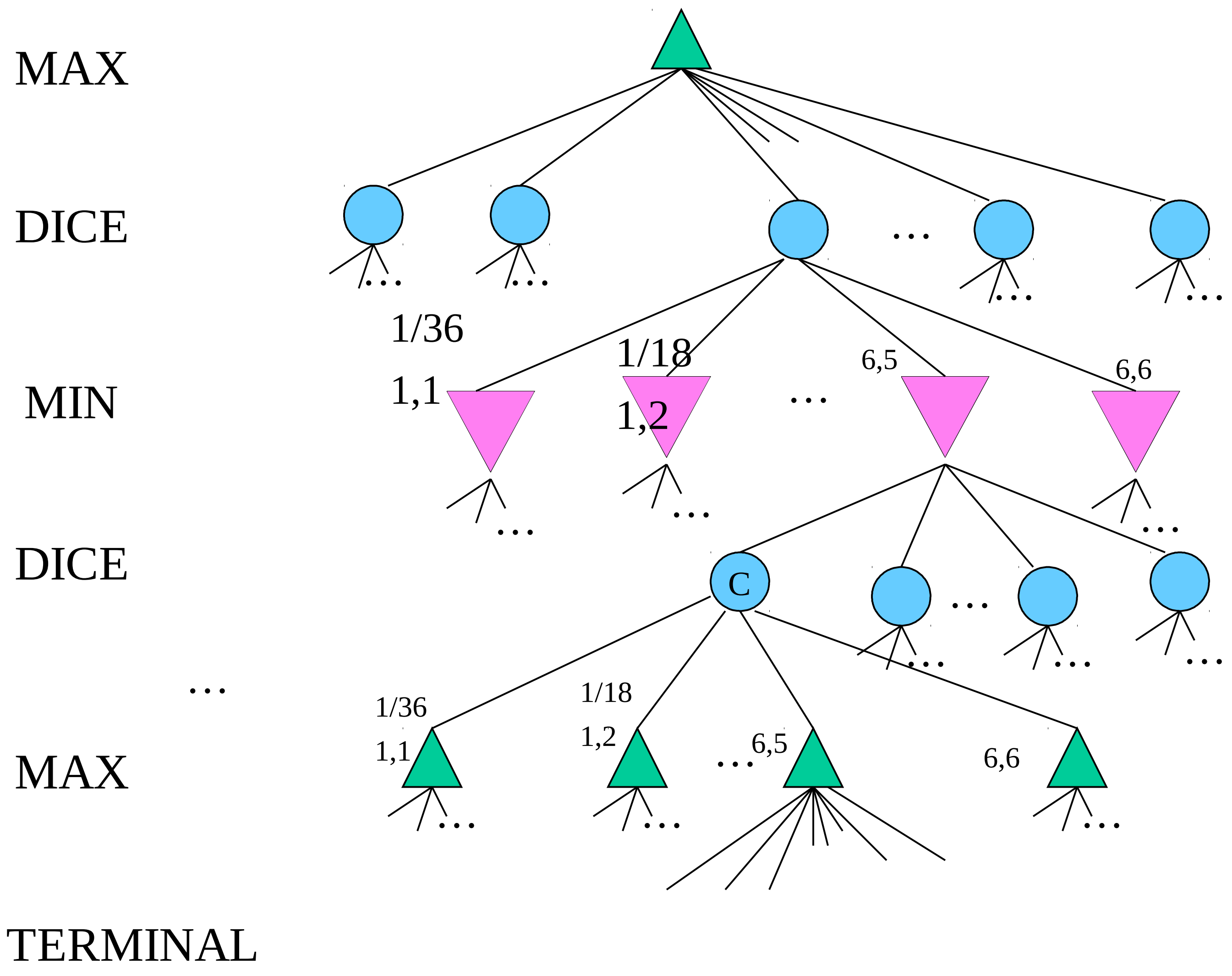
Generalization of minimax for games with chance nodes

Examples: Backgammon, bridge

Calculates expected value where probability is taken over all possible dice rolls/chance events

- **Max and Min nodes determined as before**
- **Chance nodes evaluated as weighted average**

Game Tree for Backgammon

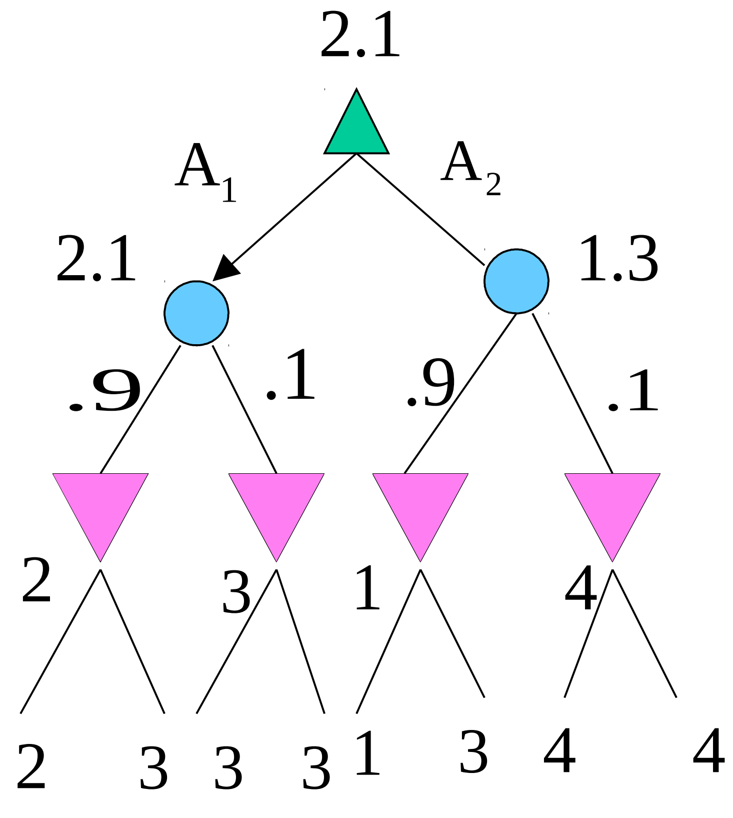


Expectiminimax

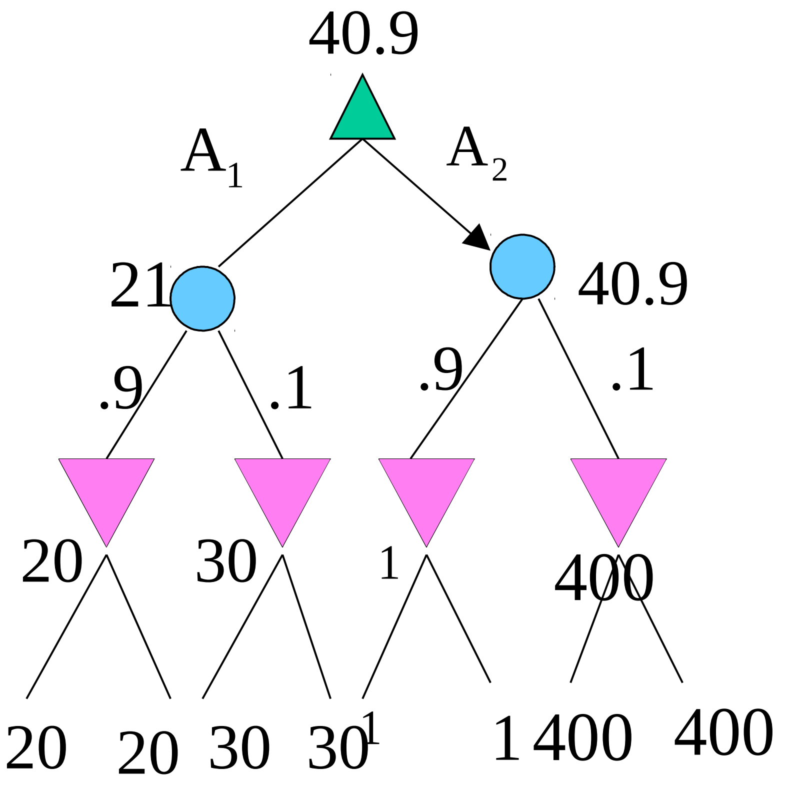
Expectiminimax(n) =

Utility(n)	for n, a terminal state
$max_{s \in Succ(n)}\ \text{expectiminimax}(\boldsymbol{s})$	for n, a Max node
$min_{s \in Succ(n)}\ \text{expectiminimax}(\boldsymbol{s})$	for n, a Min node
$\Sigma_{\boldsymbol{s} \in \boldsymbol{Succ}(\boldsymbol{n})} \boldsymbol{P}(\boldsymbol{s}) * \text{expectiminimax}(\boldsymbol{s})$	for n, a chance node

Expectiminimax



.9 * 2 + .1 * 3 = 2.1



Small chance at high payoff wins.
But, not necessarily the best thing to do!

Summary

--- game tree search

--- minimax

--- optimality under rational play

--- alpha-beta pruning

--- board evaluation function (utility) / weighted sum of features and tuning

--- expectiminimax