Lecture 13

CS 6380 Artificial Intelligence

Evaluation Functions, Cutting Off Search, and Move Ordering

Department of Computer Science and Engineering

Indian Institute of Technology Madras (IITM), India

Recap

Two Rules and You Can Trace Any Tree

- Cut when the branch you are exploring can no longer beat something the other player already refused to give up

Cut at a MAX node

stop as soon as v reaches beta or above

Cut at a MIN node

stop as soon as v falls to alpha or below

MAX updates alpha	and tests its running value against the beta it inherited from above
MIN updates beta	and tests its running value against the alpha it inherited from above
Never the other way	a node writing to the other player's bound is a bug, and a common one

Minimax Plus Four Lines

- Delete the four marked lines and plain minimax is back

```
MAX-VALUE(state, alpha, beta)

if TERMINAL-TEST(state) then
    return UTILITY(state)
v <- minus infinity
for each a in ACTIONS(state) do
    v <- MAX(v, MIN-VALUE(
           RESULT(s,a), alpha, beta))
    if v >= beta then return v
    alpha <- MAX(alpha, v)
return v
```

```
MIN-VALUE(state, alpha, beta)

if TERMINAL-TEST(state) then
    return UTILITY(state)
v <- plus infinity
for each a in ACTIONS(state) do
    v <- MIN(v, MAX-VALUE(
           RESULT(s,a), alpha, beta))
    if v <= alpha then return v
    beta <- MIN(beta, v)
return v
```

Check before you update test the cut condition first, then raise your own bound. Swapping the two is the classic bug

A cut returns a bound not always the true value of that node. The root value is still exact, which is all we needed

Where Marks Quietly Disappear

Pruning at the root

The root has no ancestor, so its beta stays at plus infinity forever and no cut can ever fire on a direct child of the root.

An equals sign on a pruned node

A node that cut early has a bound, not a value. Write at most 2 rather than equals 2, because markers look for exactly this.

Comparing the wrong way

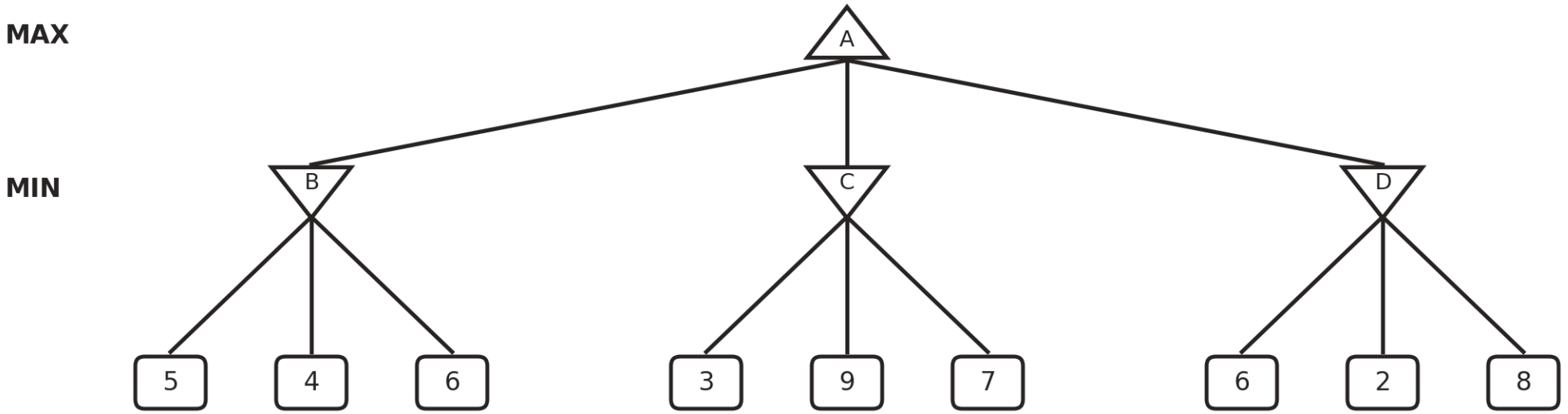
MIN compares against alpha and MAX compares against beta, always against the bound inherited from above. Your own bound can never fire.

Assuming a fixed leaf count

Reorder the children and a different number of leaves survives, so a question has to state the order before it can have an answer.

Question 1 From Last Class

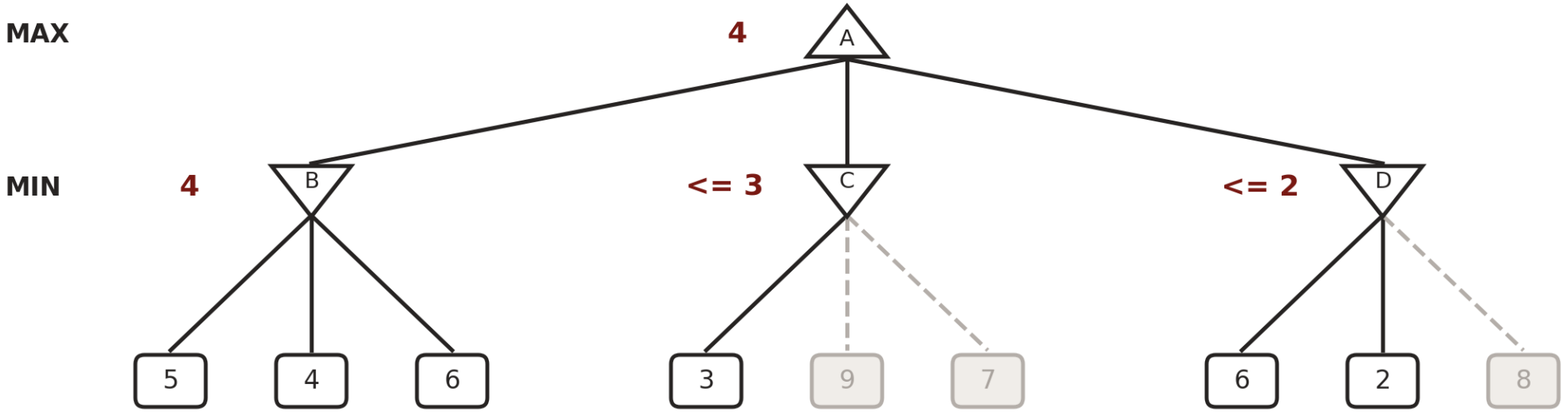
- Left to right. Value every node, mark every cut, and count the leaves you evaluated



A 2	**B** 3	**C** 4	**D** 6

Question 1, Solved

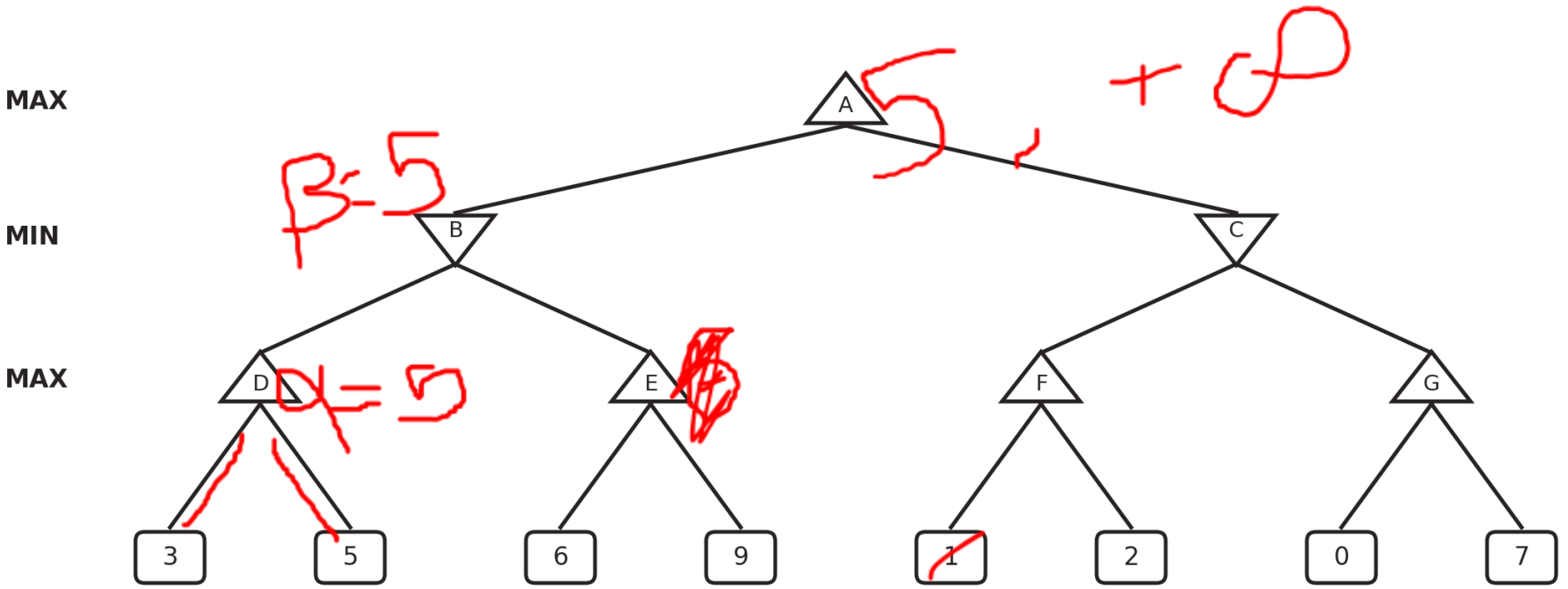
- B settles at 4 so alpha becomes 4, then C sees 3 and dies, then D survives a 6 before a 2 kills it
- The 9 under C is irrelevant, because MIN is choosing there and already holds a 3



six leaves evaluated, and the move is B

Question 2 From Last Class

- Three plies, so a MAX node can now cut on a beta inherited from above
- Predict first. Can a whole subtree disappear here, or only single leaves

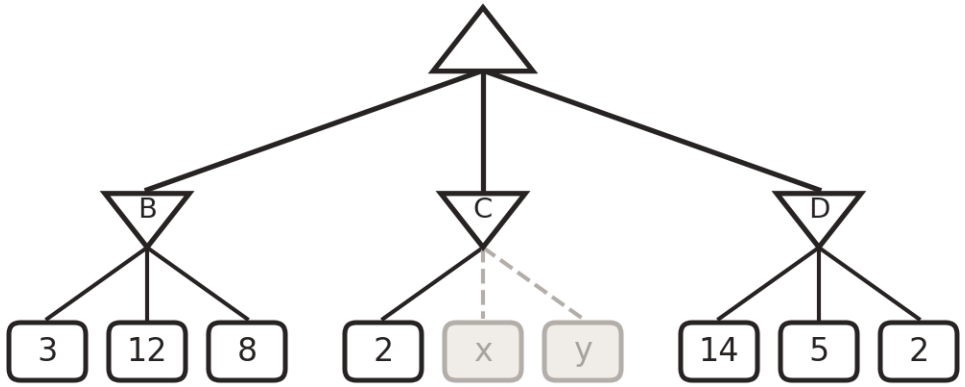


A	4 leaves	**B**	5 leaves	**C**	6 leaves	**D**	8 leaves

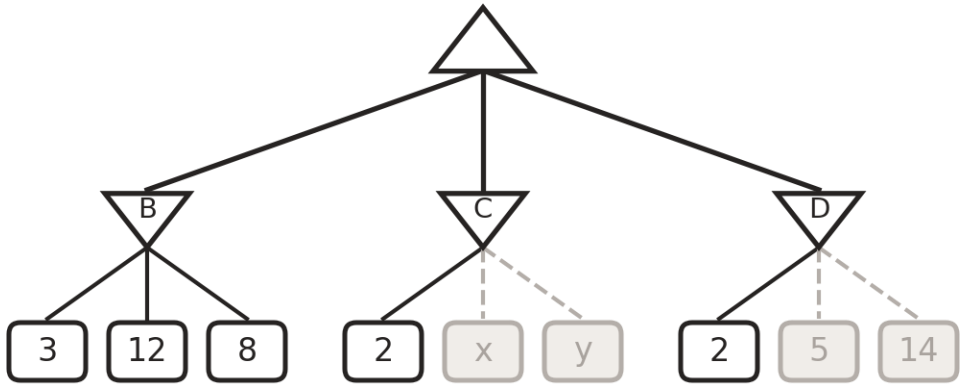
Ordering Matters

Same Tree, Different Order, Different Work

- Only the order of D's children changed. Same tree, same answer, two fewer leaves



worst case order, seven of nine leaves



D reordered, five of nine leaves

So can we just look at the best move first? Not perfectly. If we could, we would already know the answer and would not need to search at all.

What Good Ordering Is Actually Worth

- Halving the exponent is the same as taking the square root of the branching factor
- Chess stops behaving like a 35 wide tree and starts behaving like a 6 wide tree

Approach	Nodes examined	Effective branching for chess	Depth in the same time
Plain minimax	`b to the power m`	35	about 5 plies
Alpha beta, random order	`b to the power 3m/4`	about 14	about 7 plies
Alpha beta, perfect order	**`b to the power m/2`**	**6**	**about 10 plies**

Getting Good Ordering For Free

- Is it wasteful to search depth one, then two, then three, redoing the work each time? It is not

Killer move heuristic

A move that caused a cutoff a moment ago usually still works, because the threats on the board rarely change in a single ply. Keep a short list of recent offenders and try them first at the same depth.

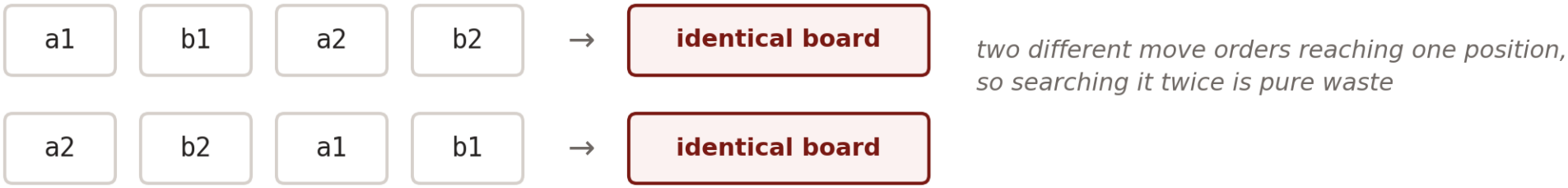
Iterative deepening, reused

Search one ply and record the best path, then search two plies but try that path first. Each pass hands the next a good ordering, and the shallow passes cost almost nothing.

The bottom level of an exponential tree dominates everything above it, so the earlier passes are nearly free.

The Same Position, Reached By Different Roads

- Hash every position you evaluate and read the answer back later, which in chess can double the reachable depth
- At a million nodes per second you cannot keep them all, so eviction is where the engineering lives



two different move orders reaching one position, so searching it twice is pure waste

With transpositions this is no longer a tree but a directed graph, which is exactly why the graph search fix applies.

Part Two, When You Cannot Reach the Leaves

We Have Been Quietly Lying This Whole Time

- Alpha beta prunes a great deal, but wherever it does explore, it goes all the way to a terminal state
- In chess that is fifty moves away and your budget is three minutes, so we stop early and guess instead

```
H-MINIMAX(s, d)  =
    EVAL(s)                                          if CUTOFF-TEST(s, d)
    max over a of H-MINIMAX(RESULT(s,a), d+1)        if PLAYER(s) = MAX
    min over a of H-MINIMAX(RESULT(s,a), d+1)        if PLAYER(s) = MIN
```

Shannon proposed exactly this in 1950, and every engine since has done it.

Two Substitutions, One Formula

What changed	UTILITY became EVAL and TERMINAL-TEST became CUTOFF-TEST. The recursion itself is untouched.
In your code	One line, and remember to increment depth on every recursive call, which is the part people forget.
Simplest cutoff	A fixed depth d chosen so the move arrives on time, returning true at genuine terminal states as well.
Better cutoff	Iterative deepening. When the clock runs out you play the move from the deepest search that finished.

Three Requirements For an Evaluation Function

- Chess has no dice, so where does a chance of winning come from? From your own ignorance
- The uncertainty is computational rather than informational. You stopped searching, so you do not know

Order terminal states right

Wins above draws above losses. Get this wrong and the program can misplay a position it can see all the way to the end of.

Be fast

The whole point was to search more positions. An evaluation clever enough to halve your node count has probably cost you more than it gained.

Track the chance of winning

Not beauty, not material for its own sake. For non terminal states the number should correlate with actually winning from here.

Features, Weights, and a Number at the End

- EVAL(s) is w1 f1(s) plus w2 f2(s) and so on, so count the features and add the contributions
- Nothing in the rules of chess says a bishop is worth three pawns. That came from human experience

Feature	Weight	White	Black	Contribution
Pawns	1	6	5	+1
Knights	3	1	2	−3
Bishops	3	2	1	+3
Rooks	5	2	2	0
Queens	9	1	1	0
King safety	0.5	good	exposed	+0.5
Total				**+1.5**

What the Number Actually Means

- Features sort positions into categories, and any category holds a mixture of wins, draws and losses

Win	72 percent of that category, with utility 1
Loss	20 percent of that category, with utility 0
Draw	8 percent of that category, with utility one half

```
(0.72 x 1)  +  (0.20 x 0)  +  (0.08 x 0.5)   =   0.76
```

Far too many categories exist to ever measure them all, so instead we add up a contribution per feature, and that addition assumes something false.

The Assumption Hiding in the Plus Signs

- Adding contributions assumes each feature is independent of the others, which is false in every real game

A bishop is worth 3	except in an open endgame with room to manoeuvre, where it is worth considerably more
Two bishops are worth 6	except that a pair covers both colour complexes, so it beats twice a single bishop
A rook is worth 5	except when it is trapped behind its own pawns and doing nothing at all

The fix is non linear terms. The deeper point is that where human experience does not exist, you have to learn the weights instead.

When Stopping Early Goes Wrong

- This is not a flaw in the evaluation function. Counting material is reasonable. The flaw is where we stopped

The material trap

The search reaches its depth limit, counts material, finds Black ahead by a knight and two pawns, and reports a probable win for Black. White then captures the queen for free, and the position was won for White the whole time.

Quiescence search

Only evaluate positions unlikely to swing wildly in value. If the position is not calm, keep searching, but expand only the violent moves. Follow the fireworks until they stop, then evaluate.

The Horizon Effect

- Black's bishop is trapped and falls in six plies, and Black searches eight plies deep

Ply 1 and 2	Black checks the king with a pawn, and the king is forced to capture it
Ply 3 and 4	Black checks again with a second pawn, and the king captures again
Ply 5 to 8	Four plies remain, which is not enough to reach the bishop capture
Black concludes	the bishop is safe and it only cost two pawns, which is excellent play

Two pawns thrown away to push an unavoidable loss just past the edge of what Black can see.

A singular extension follows a clearly best move past the depth limit, which mitigates this and never cures it.

Pruning On Purpose

- Every cut so far was safe. Forward pruning throws that guarantee away, and every strong engine does it
- Two of these three are gambles that pay off, and we know that only because somebody measured it

Beam search

Keep only the n best looking moves at each ply. Very human, and dangerous, since nothing stops you discarding the winning move.

ProbCut

Prune what is probably outside the window, using a shallow search and past statistics. It beat plain alpha beta 64 percent of the time at Othello.

Null move

Let the opponent move twice and see how bad it gets. A cheap lower bound that pushes chess effective branching under three.

Not Everything Needs Searching

- Retrograde analysis reverses the rules of chess, unmoving instead of moving, until every position is labelled

Openings, human expertise

It is absurd to weigh a billion positions in order to play the pawn to e4. Copy the opening books into a table. After about ten moves you leave the book and search resumes.

Endgames, machine expertise

Here the computer knows what no human does. It solves the endgame exhaustively and stores a policy, a best move for every position, including endings humans cannot describe.

3,494,568

KBNK positions, from 462 king placements times 62 times 61 times 2

262

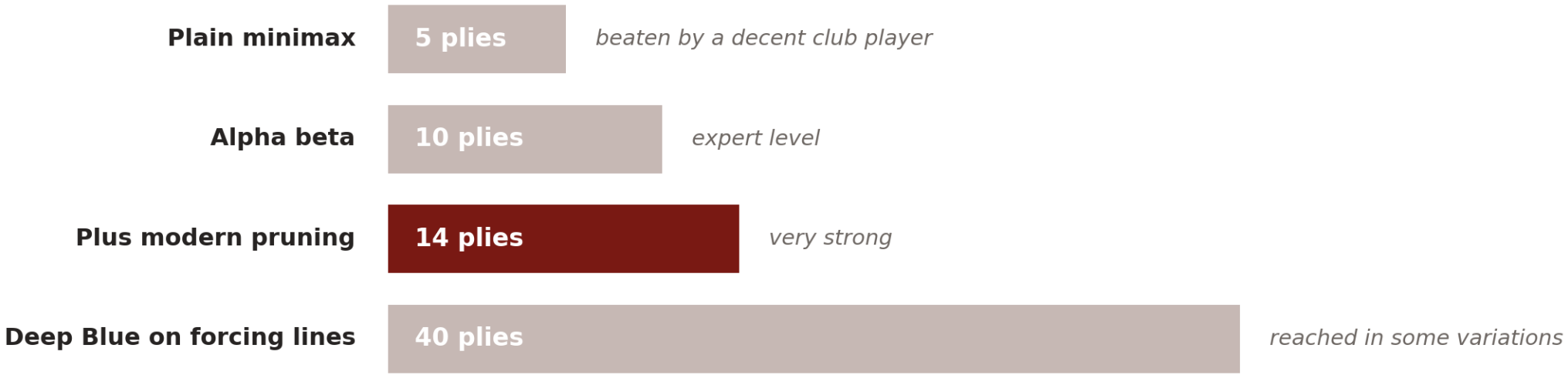
moves in a forced mate Stiller found, awkward under the fifty move rule

39 trillion

endgame positions stored by Chinook, which is why checkers is solved

What Each Technique Is Worth In Plies

- Search buys you depth, but grandmaster play also needs a tuned evaluation and large move databases



Part Three, Games With Dice

Now We Break Another Assumption

- You roll before you move, so White knows White's own options and has no idea what Black will be allowed to do
- Why is a double half as likely as any other roll? Because there is only one ordered way to roll it

36

ordered ways to roll two dice, all equally likely

21

distinct rolls, since a 6 then 5 matches a 5 then 6

1 in 36

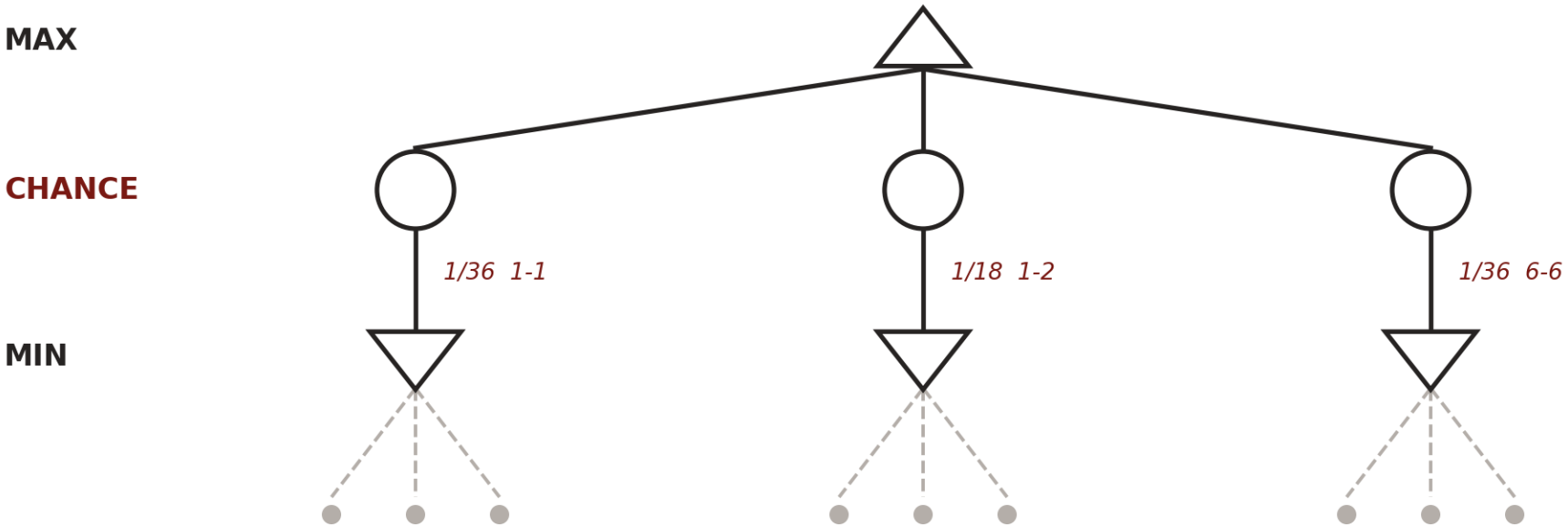
the probability of each of the six doubles

1 in 18

the probability of each of the other fifteen rolls

The Missing Layer, Chance Nodes

- Nobody chooses at a chance node, so it can take neither a max nor a min. It has to take an average



a real backgammon chance node has twenty one branches, not three