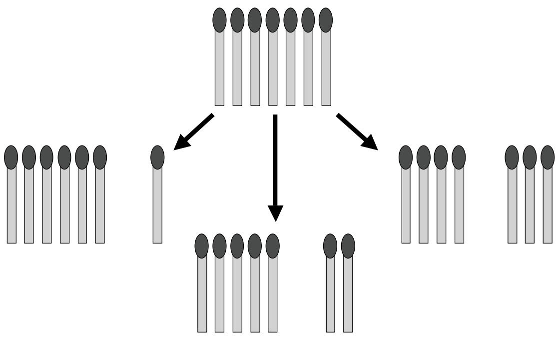
§4 Game Trees

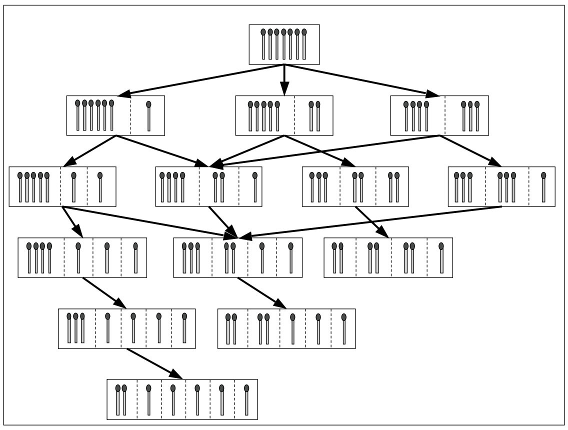
- perfect information games
  - no hidden information
- two-player, perfect information games
  - Noughts and Crosses
  - Chess
  - Go
- imperfect information games
  - Poker
  - Backgammon
  - Monopoly
- zero-sum property
  - one player's gain equals another player's loss

Game tree

- all possible plays of two-player, perfect information games can be represented with a game tree
  - nodes: positions (or states)
  - edges: moves
- players: MAX (has the first move) and MIN
- ply = the length of the path between two nodes
  - MAX has even plies counting from the root node
  - MIN has odd plies counting from the root node

Division Nim with seven matches





Problem statement

Given a node v in a game tree

find a winning strategy for MAX (or MIN) from v

or (equivalently)

show that MAX (or MIN) can force a win from v

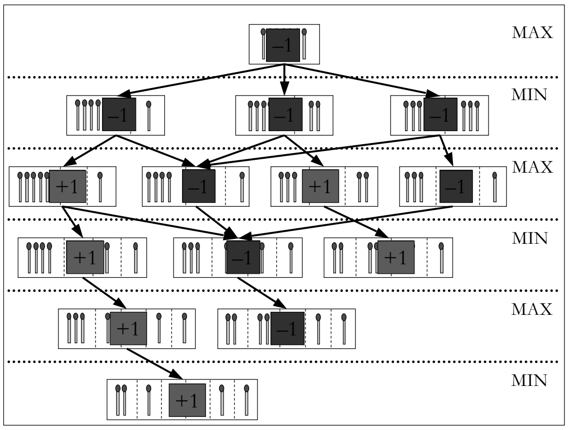
Minimax

- assumption: players are rational and try to win
- given a game tree, we know the outcome in the leaves
  - assign the leaves to win, draw, or loss (or a numeric value like +1, 0, –1) according to MAX's point of view
- at nodes one ply above the leaves, we choose the best outcome among the children (which are leaves)
  - MAX: win if possible; otherwise, draw if possible; else loss
  - MIN: loss if possible; otherwise, draw if possible; else win
- recurse through the nodes until in the root

Minimax rules

1. If the node is labelled to MAX, assign it to the maximum value of its children.
2. If the node is labelled to MIN, assign it to the minimum value of its children.

- MIN minimizes, MAX maximizes → minimax



Analysis

- simplifying assumptions
  - internal nodes have the same branching factor b
  - game tree is searched to a fixed depth d
- time consumption is proportional to the number of expanded nodes
  - 1 — root node (the initial ply)
  - b — nodes in the first ply
  - b^2 — nodes in the second ply
  - b^d — nodes in the dth ply
- overall running time $O(b^d)$

Rough estimates on running times when $d = 5$

- suppose expanding a node takes 1 ms
- branching factor b depends on the game
- Draughts ($b \approx 3$): $t = 0.243$ s
- Chess ($b \approx 30$): $t = 6\frac{3}{4}$ h
- Go ($b \approx 300$): $t = 77$ a
- alpha-beta pruning reduces b

Controlling the search depth

- usually the whole game tree is too large
  → limit the search depth
  → a partial game tree
  → partial minimax
- n-move look-ahead strategy
  - stop searching after n moves
  - make the internal nodes (i.e., frontier nodes) leaves
  - use an evaluation function to 'guess' the outcome

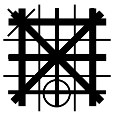
Evaluation function

- combination of numerical measurements $m_i(s, p)$ of the game state
  - single measurement: $m_i(s, p)$
  - difference measurement: $m_i(s, p) - m_j(s, q)$
  - ratio of measurements: $m_i(s, p) / m_j(s, q)$
- aggregate the measurements maintaining the zero-sum property

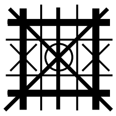
Example: Noughts and Crosses

- heuristic evaluation function e:
  - count the winning lines open to MAX
  - subtract the number of winning lines open to MIN
- forced wins
  - state is evaluated $+\infty$, if it is a forced win for MAX
  - state is evaluated $-\infty$, if it is forced win for MIN

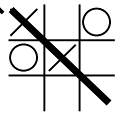
Examples of the evaluation



$e(\bullet) = 6 - 5 = 1$



$e(\bullet) = 4 - 5 = -1$



$e(\bullet) = +\infty$

Drawbacks of partial minimax

- horizon effect
  - heuristically promising path can lead to an unfavourable situation
  - staged search: extend the search on promising nodes
  - iterative deepening: increase n until out of memory or time
  - phase-related search: opening, midgame, end game
  - however, horizon effect cannot be totally eliminated
- bias
  - we want to have an estimate of minimax but get a minimax of estimates
  - distortion in the root: odd plies → win, even plies → loss

The deeper the better...?

- assumptions:
  - n-move look-ahead
  - branching factor b, depth d,
  - leaves with uniform random distribution
- minimax convergence theorem:
  - n increases → root value converges to $f(b, d)$
- last player theorem:
  - root values from odd and even plies not comparable
- minimax pathology theorem:
  - n increases → probability of selecting non-optimal move increases (← uniformity assumption!)

Alpha-beta pruning

- reduce the branching factor of nodes
- alpha value
  - associated with MAX nodes
  - represents the worst outcome MAX can achieve
  - can never decrease
- beta value
  - associated with MIN nodes
  - represents the worst outcome MIN can achieve
  - can never increase

Example

- in a MAX node, $\alpha = 4$
  - we know that MAX can make a move which will result at least the value 4
  - we can omit children whose value is less than or equal to 4
- in a MIN node, $\beta = 4$
  - we know that MIN can make a move which will result at most the value 4
  - we can omit children whose value is greater than or equal to 4

Rules of pruning

1. Prune below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors.
2. Prune below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN ancestors

Or, simply put: If $\alpha \geq \beta$, then prune below!

Best-case analysis

- omit the principal variation
- at depth $d - 1$ optimum pruning: each node expands one child at depth d
- at depth $d - 2$ no pruning: each node expands all children at depth $d - 1$
- at depth $d - 3$ optimum pruning
- at depth $d - 4$ no pruning, etc.
- total amount of expanded nodes: $\Omega(b^{d/2})$

Principal variation search

- alpha-beta range should be small
  - limit the range artificially → aspiration search
  - if search fails, revert to the original range
- if we find a move between α and β, assume we have found a principal variation node
  - search the rest of nodes the assuming they will not produce a good move
  - if the assumption fails, re-search the node
- works well if the principal variation node is likely to get selected first

Games of chance

- minimax trees assume determistic moves
  - what about indeterministic events like tossing a coin, casting a die or shuffling cards?
- chance nodes: *-minimax tree
- expectiminimax
  - if node v is labelled to CHANCE, multiply the probability of a child with its expectiminimax value and return the sum over all v's children
  - otherwise, act as in minimax