Artificial Intelligence

Topic 5

Game playing

- ◇ broadening our world view — dealing with incompleteness
- ◇ why play games?
- ◇ perfect decisions — the MINIMAX algorithm
- ◇ dealing with resource limits
  - — evaluation functions
  - — cutting off search
- ◇ alpha-beta pruning
- ◇ game-playing agents in action

Reading: Russell and Norvig, Chapter 5

1. Broadening our world view

We have assumed we are dealing with world descriptions that are:

complete — all necessary information about the problem is available to the search algorithm

deterministic — effects of actions are uniquely determined

Real-world problems are rarely complete and deterministic. . .

Sources of Incompleteness

sensor limitations — not possible to gather enough information about the world to completely know its state — includes the future!

intractability — full state description is too large to store, or search tree too large to compute

Sources of (Effective) Nondeterminism

- humans, the weather, stress fractures, dice, . . .

Aside. . .

Debate: incompleteness $\leftrightarrow$ nondeterminism

contingency planning

- build all possibilities into the plan
- may make the tree very large
- can only guarantee a solution if the number of contingencies is finite and tractable

interleaving or adaptive planning

- alternate between planning, acting, and sensing
- requires extra work during execution — planning cannot be done in advance (or "off-line")

strategy learning

- learn, from looking at examples, strategies that can be applied in any situation
- must decide on parameterisation, how to evaluate states, how many examples to use, . . . black art??

2. Why Play Games?

- abstraction of real world
- well-defined, clear state descriptions
- limited operations, clearly defined consequences

but!

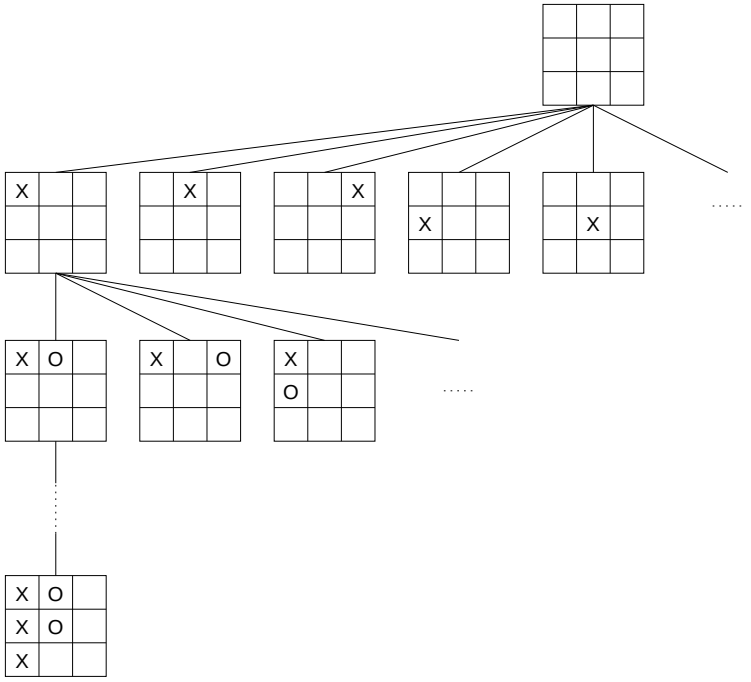
- provide a mechanism for investigating many of the real-world issues outlined above

  $\Rightarrow$ more like the real world than examples so far

Added twist — the domain contains hostile agents (also making it like the real world. . . ?)

Tractable Problem with Complete Information

Noughts and crosses (tic-tac-toe) for control freaks — you get to choose moves for both players!



Stop when you get to a goal state.

- What uninformed search would you select? How many states visited?
- What would be an appropriate heuristic for an informed search? How many states visited?

Tractable Contingency Problem

Noughts and crosses — allow for all the oponents moves. (Opponent is non-deterministic.)

How many states?

Intractable Contingency Problem

Chess

- average branching factor 35, approx 50 operations
  $\Rightarrow$ search tree has about 35^{100} nodes (although only about 10^{40} different legal positions)!
- cannot solve by brute force, must use other approaches, eg.
  - interleave time- (or space-) limited search with moves
    $\Rightarrow$ this section
    - algorithm for perfect play (Von Neumann, 1944)
    - finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
    - pruning to reduce costs (McCarthy, 1956)
  - learn strategies that determine what to do based on some aspects of the current position
    $\Rightarrow$ later in the course

3. Perfect Decisions — MINIMAX Algorithm

Perfect play for deterministic, perfect-information games

- two players, MAX and MIN, both try to win
- MAX moves first

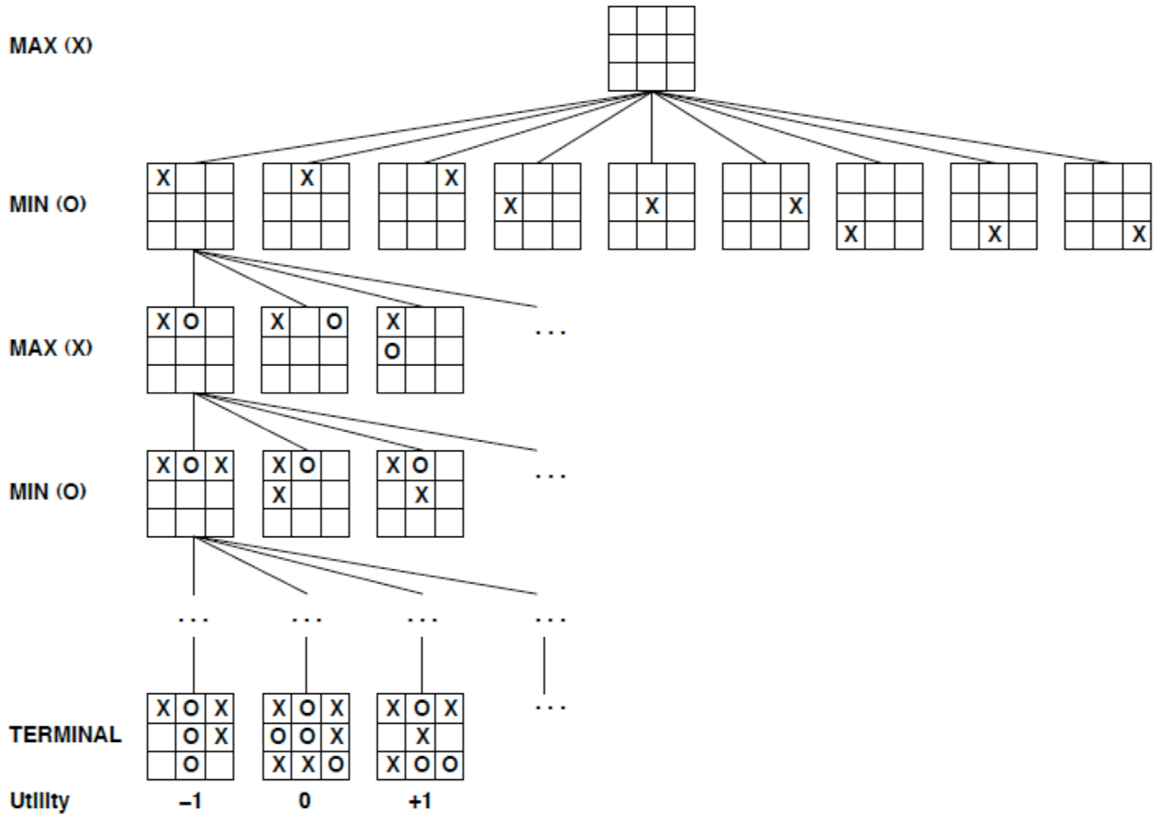
  $\Rightarrow$ can MAX find a strategy that *always wins?*

Define a game as a kind of search problem with:

- initial state
- set of legal moves (operators)
- terminal test — is the game over?
- utility function — how good is the outcome for each player?

eg. Tic-tac-toe — can MAX choose a move that always results in a terminal state with a utility of $+1$?

3. Perfect Decisions — Minimax Algorithm

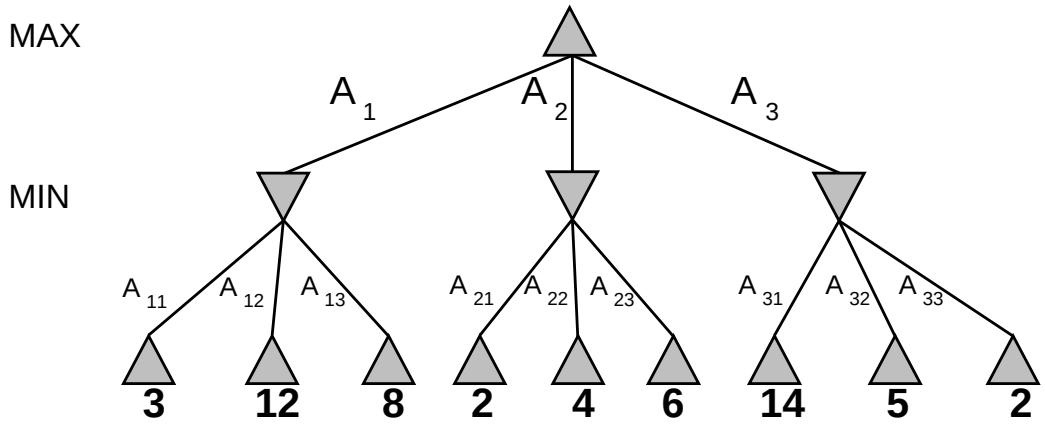


Even for this simple game the search tree is large.

Try an even simpler game...

3. Perfect Decisions — MINIMAX Algorithm

eg. Two-ply (made-up game)



(one move deep, two *ply*)

- MAX's aim — maximise utility of terminal state
- MIN's aim — minimise it
- what is MAX's optimal strategy, *assuming* MIN makes the best possible moves?

3. Perfect Decisions — Minimax Algorithm

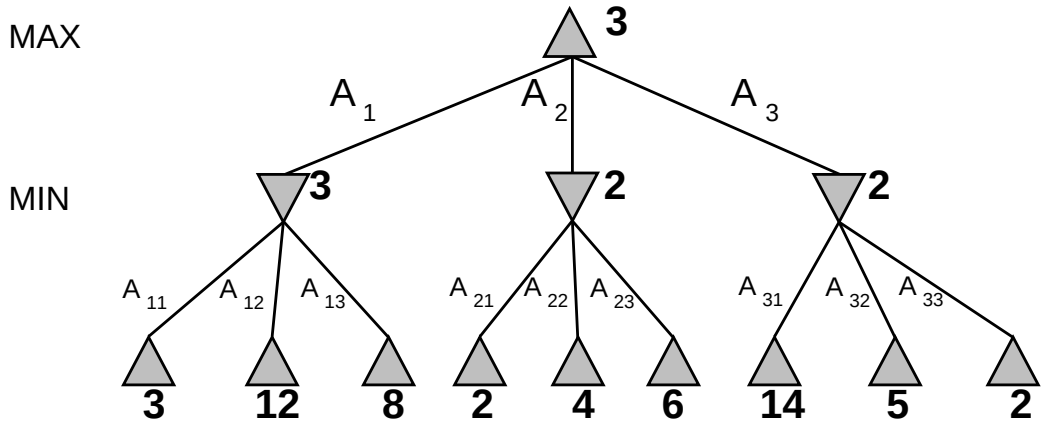
function MINIMAX-DECISION(*game*) **returns** *an operator*

  for each *op* **in** OPERATORS[*game*] **do**
    VALUE[*op*] ← MINIMAX-VALUE(APPLY(*op*, *game*), *game*)
  end
  return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state*, *game*) **returns** *a utility value*

  if TERMINAL-TEST[*game*](*state*) **then**
    return UTILITY[*game*](*state*)
  else if MAX is to move in *state* **then**
    return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)



3. Perfect Decisions — Minimax Algorithm

Complete Yes, if tree is finite (chess has specific rules for this)
Optimal Yes, against an optimal opponent. Otherwise??
Time complexity $O(b^m)$
Space complexity $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$\Rightarrow$ exact solution completely infeasible

Resource limits

Usually time: suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow \underline{10^6}$ nodes per move

Standard approach:

- *cutoff test*
  e.g., depth limit (perhaps add *quiescence search*)
- *evaluation function*
  = estimated desirability of position

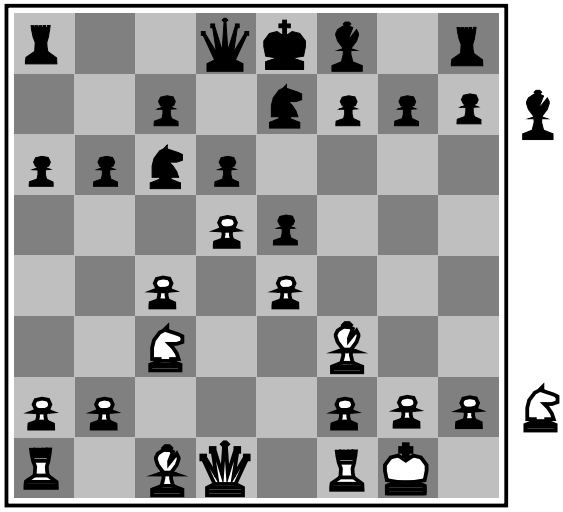
4. Evaluation functions

Instead of stopping at terminal states and using utility function, cut off search and use a heuristic *evaluation function*.

Chess players have been doing this for years...

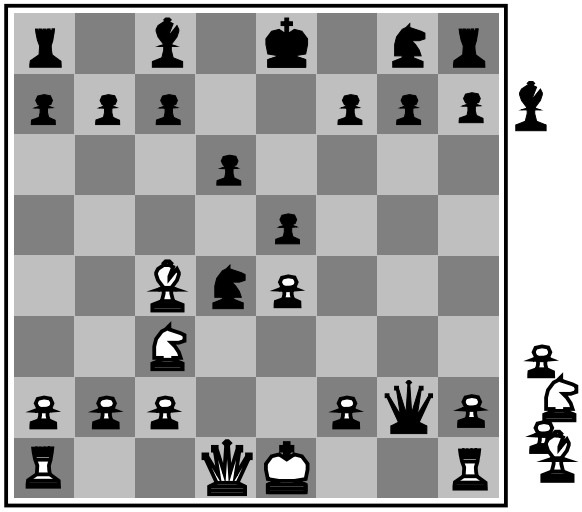
simple — 1 for pawn, 3 for knight/bishop, 5 for rook, etc

more involved — centre pawns, rooks on open files, etc



Black to move

White slightly better



White to move

Black winning

Can be expressed as *linear weighted sum* of *features*

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

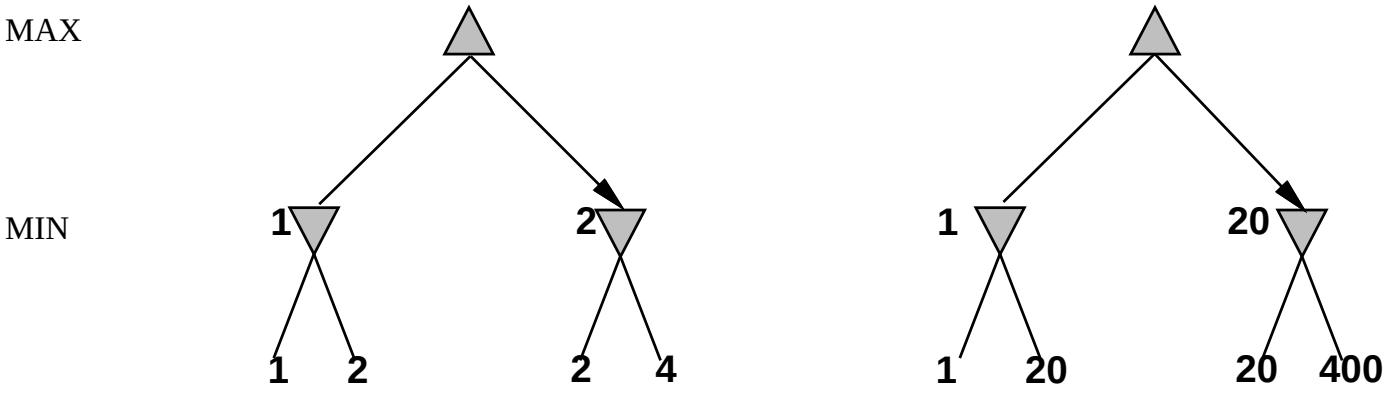
e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens)

4.1 Quality of evalation functions

Success of program depends *critically* on quality of evalutation function.

- agree with utility function on terminal states
- time efficient
- reflect chances of winning

Note: Exact values don't matter



Behaviour is preserved under any *monotonic* transformation of Eval

Only the order matters:

payoff acts as an *ordinal utility* function

5. Cutting off search

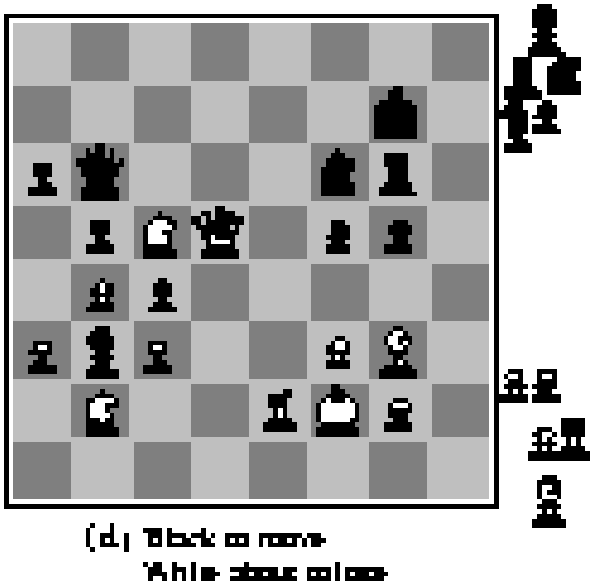
Options. . .

- fixed depth limit
- iterative deepening (fixed time limit) — more robust

Problem — inaccuracies of evaluation function can have disastrous consequences.

5.1 Non-quiescence problem

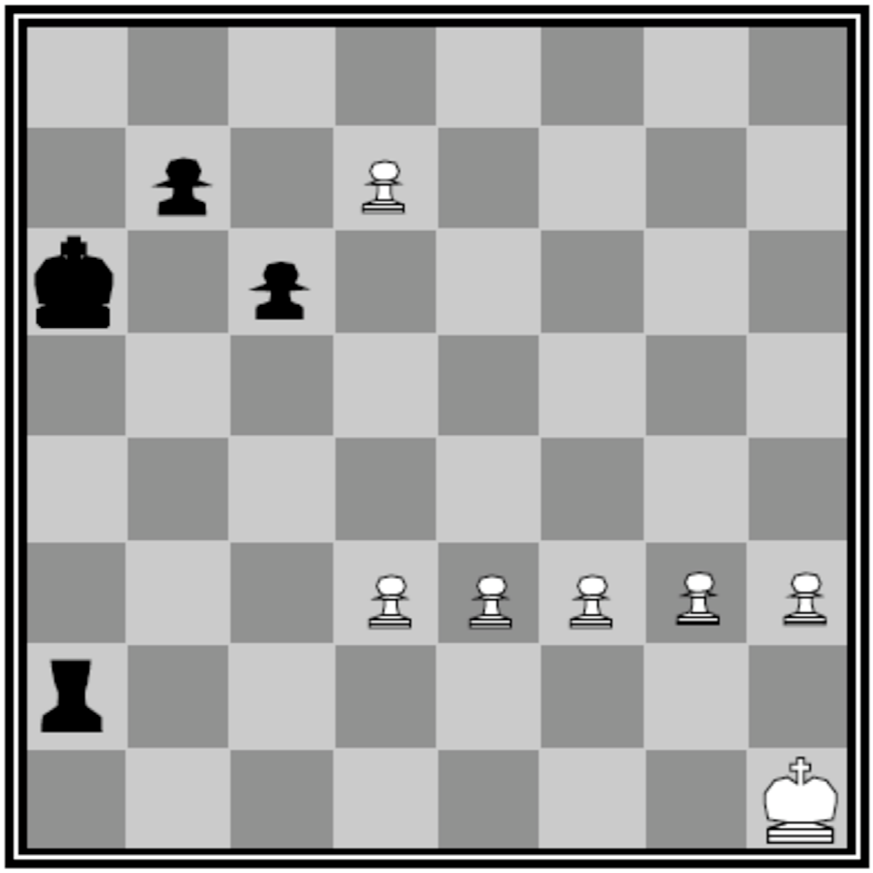
Consider chess evaluation function based on material advantage. White's depth limited search stops here...



Looks like a win to white — actually a win to black.

Want to stop search and apply evaluation function in positions that are *quiescent*. May perform *quiescence search* in some situations — eg. after capture.

5.2 Horizon problem



Black to move

Win for white, but black may be able to chase king for extent of its depth-limited search, so does not see this. Queening move is "pushed over the horizon".

No general solution.

6. Alpha-beta pruning

Consider MINIMAX with reasonable evaluation function and quiescent cut-off. Will it work in practice?

Assume can search approx 5000 positions per second. Allowed approx 150 seconds per move. Order of 10^6 positions per move.

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

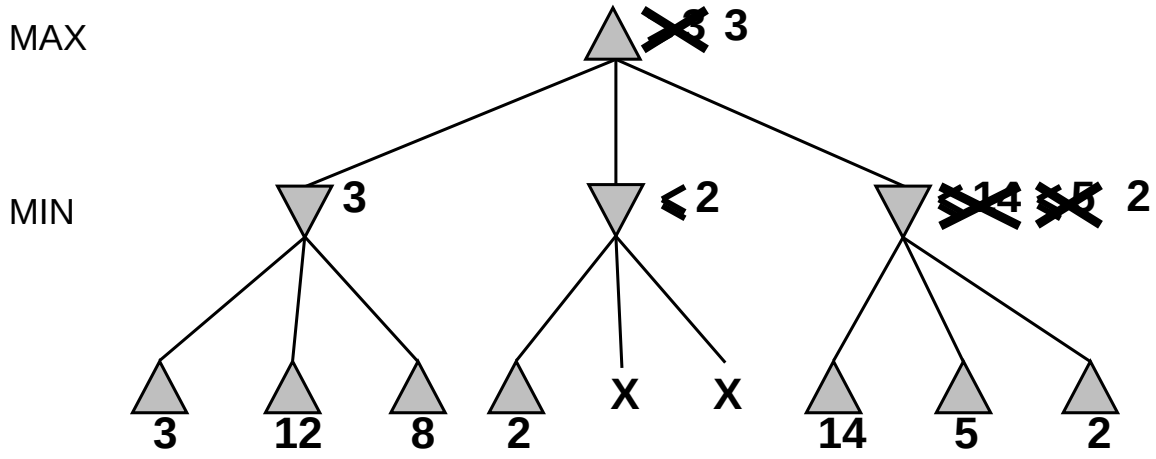
4-ply lookahead is a hopeless chess player!

4-ply $\approx$ human novice
8-ply $\approx$ typical PC, human master
12-ply $\approx$ Deep Blue, Kasparov

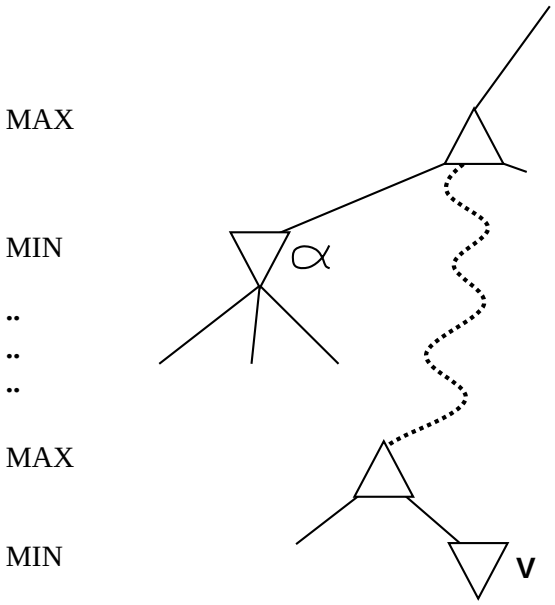
But do we need to search all those positions? Can we eliminate some before we get there — *prune* the search tree?

One method is *alpha-beta* pruning...

6.1 α–β pruning example



6.2 Why is it called α–β?



α is the best value (to MAX) found so far off the current path

If V is worse than α, MAX will avoid it $\Rightarrow$ prune that branch

Define β similarly for MIN

6.3 The α–β algorithm

Basically MINIMAX + keep track of α, β + prune

function MAX-VALUE(*state*, *game*, α, β) **returns** the minimax value of *state*

inputs: *state*, current state in game
game, game description
α, the best score for MAX along the path to *state*
β, the best score for MIN along the path to *state*

if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
for each *s* **in** SUCCESSORS(*state*) **do**
$\alpha \leftarrow$ MAX(α, MIN-VALUE(*s*, *game*, α, β))
if $\alpha \geq \beta$ **then return** β
end
return α

---

function MIN-VALUE(*state*, *game*, α, β) **returns** the minimax value of *state*

if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
for each *s* **in** SUCCESSORS(*state*) **do**
$\beta \leftarrow$ MIN(β, MAX-VALUE(*s*, *game*, α, β))
if $\beta \leq \alpha$ **then return** α
end
return β

6.4 Properties of α–β

Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
$\Rightarrow$ *doubles* depth of search
$\Rightarrow$ can easily reach depth 8 and play good chess

Perfect ordering is unknown, but a simple ordering (captures first, then threats, then forward moves, then backward moves) gets fairly close.

Can we learn appropriate orderings? $\Rightarrow$ *speedup learning*

(Note complexity results assume idealized tree model:

- nodes have same branching factor b
- all paths reach depth limit d
- leaf evaluations randomly distributed

Ultimately resort to empirical tests.)

7. Game-playing agents in practice

Games that don't include chance

Checkers: Chinook became world champion in 1994 after 40-year-reign of human world champion Marion Tinsley (who retired due to poor health). Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match (not a World Championship) in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

7. Game-playing agents in practice

Games that include an element of chance

Dice rolls increase b: 21 possible rolls with 2 dice

Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

As depth increases, probability of reaching a given node shrinks
$\Rightarrow$ value of lookahead is diminished

α–β pruning is much less effective

TDGammon uses depth-2 search + very good Eval
$\approx$ world-champion level

8. Summary

Games are fun to work on! (and can be addictive)

They illustrate several important points about AI

♢ problems raised by
  — incomplete knowledge
  — resource limits

♢ perfection is unattainable ⇒ must approximate

Games are to AI as grand prix racing is to automobile design

The End