2 handouts: slides, asst 1 solution

asst 1 due

EOLQs

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

Adversarial Search

Another Twist on Search

Adversarial Search

- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

- Shortest-path (M&C, vacuum, tile puzzle)
  - want least-cost path to goal at unkown depth
- Constraint satisfaction (map coloring, n-queens)
  - any goal that satisfies constraints (fixed depth)
- Combinatorial optimization (TSP, max-CSP)
  - want least-cost goal (fixed depth)
- Decisions with an adversary (chess, tic-tac-toe)
  - adversary might prevent path to best goal
  - want best assured outcome

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

Each *ply* corresponds to half a *move*.
Terminal states are labeled with value.
Can also bound depth and use a *static evaluation function* on non-terminal states.

Evaluation for Tic-tac-toe

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

A *3-length* is a complete row, column, or diagonal.

$$
\begin{aligned}
\text{value of position} &= \infty \text{ if win for me,} \\
\text{or} &= -\infty \text{ if a win for you,} \\
\text{otherwise} &= \#\ \text{3-lengths open for me} - \\
&\qquad \#\ \text{3-lengths open for you}
\end{aligned}
$$

Tic-tac-toe: two-ply search

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

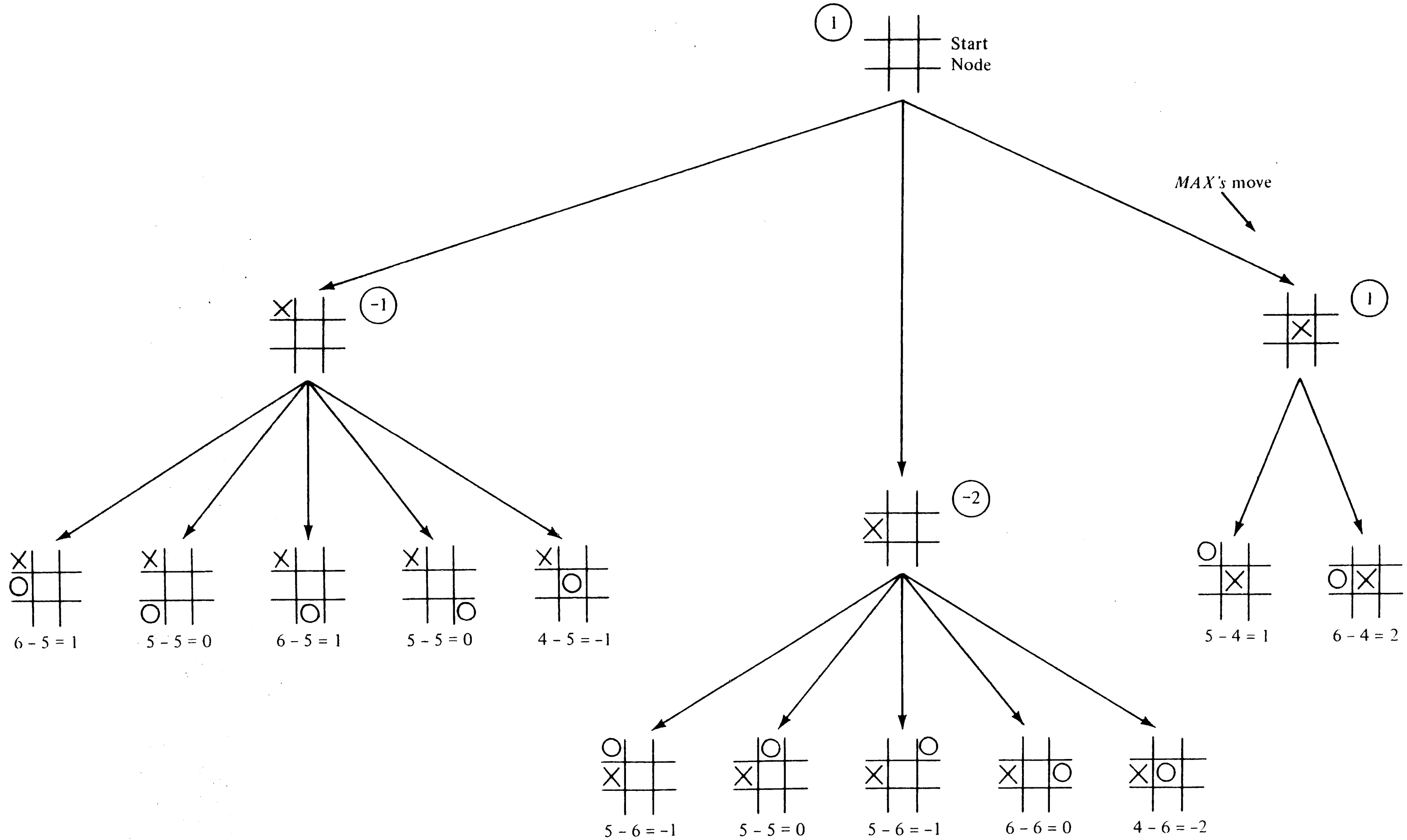

Fig. 3.8 Minimax applied to tic-tac-toe (stage 1).

Tic-tac-toe: second move

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

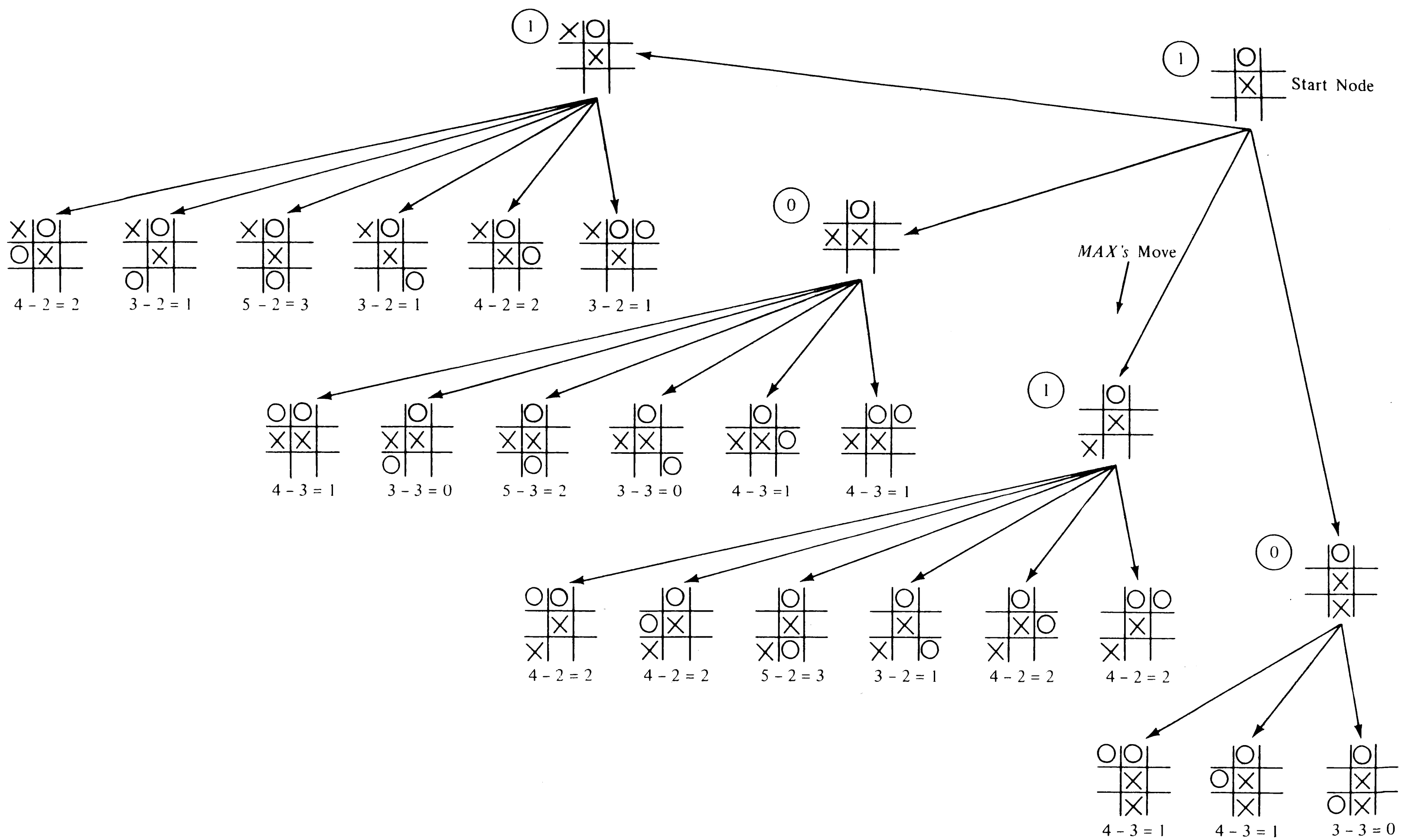

Fig. 3.9 Minimax applied to tic-tac-toe (stage 2).

Tic-tac-toe: third move

Adversarial Search

- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

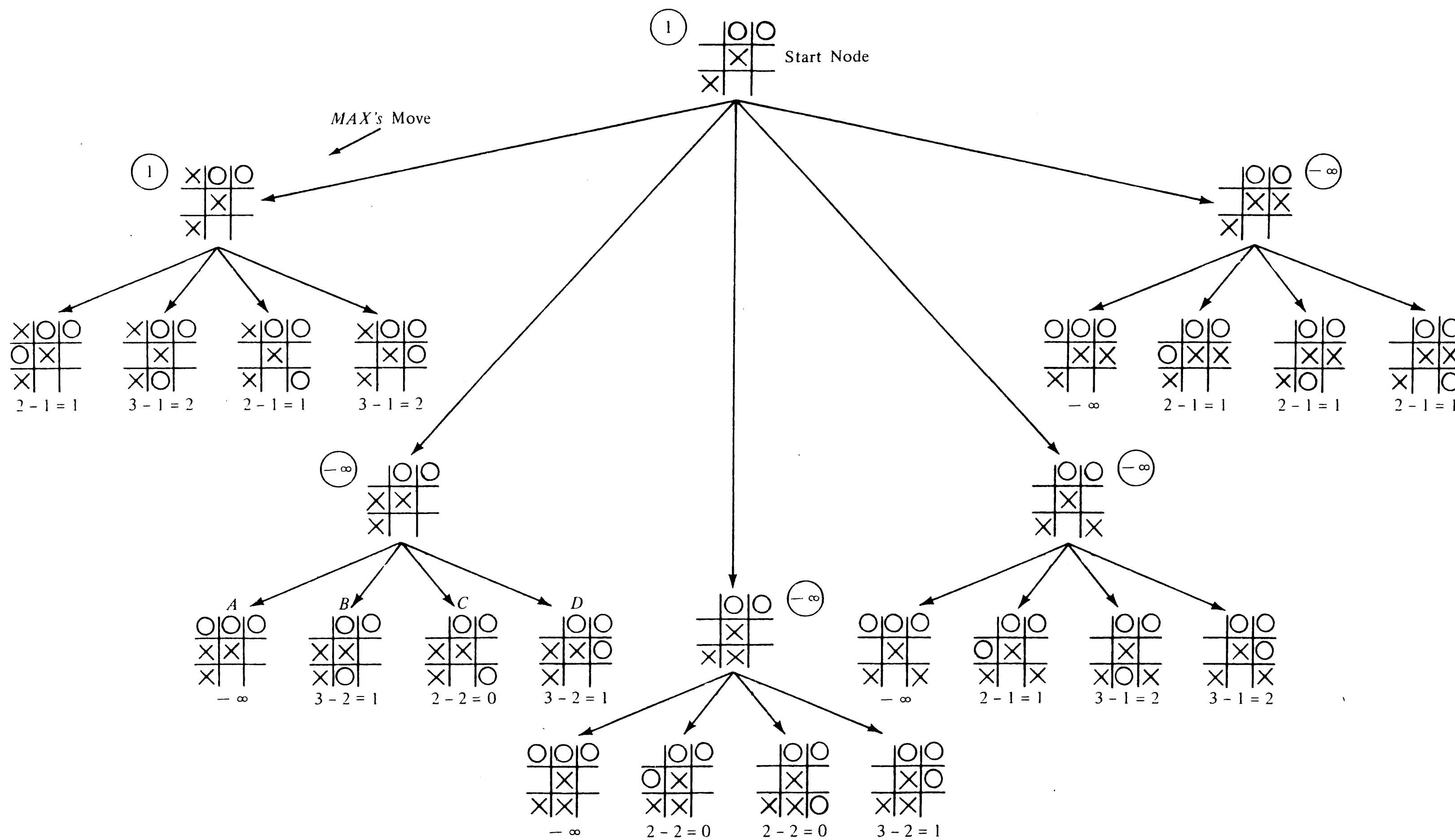

Fig. 3.10 Minimax applied to tic-tac-toe (stage 3).

Improving the Search

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

- partial expansion, SEF
- symmetry (‘transposition tables’)
- search more ply as we have time (De Groot figure)
- avoid unnecessary evaluations

Break

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

- asst 1 was due
- book
- asst 2 (theorem prover) going out on Wed. parse simple CFG.
- exams are during common exam time
- have web access? a clicker?

Which Values are Necessary?

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

α-β Pruning

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

α best outcome Max can force at previous decision on this path (init to $-\infty$)

β best outcome Min can force at previous decision on this path (init to ∞)

α and β values are copied down the tree (but not up).
Minmax values are passed up the tree, as usual.

α-β Pseudo-code

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs


```
Max-value (state, α, β):
    when depth-cutoff (state), return SEF(state)
    for each child of state
        α ← max(α, Min-value (child, α, β))
        when α ≥ β, return α
    return α

Min-value (state, α, β):
    when depth-cutoff (state), return SEF(state)
    for each child of state
        β ← min(β, Max-value (child, α, β))
        when β ≤ α, return β
    return β
```

α-

Adversarial Sear
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-c
- Why α-β?
- Progress
- EOLQs

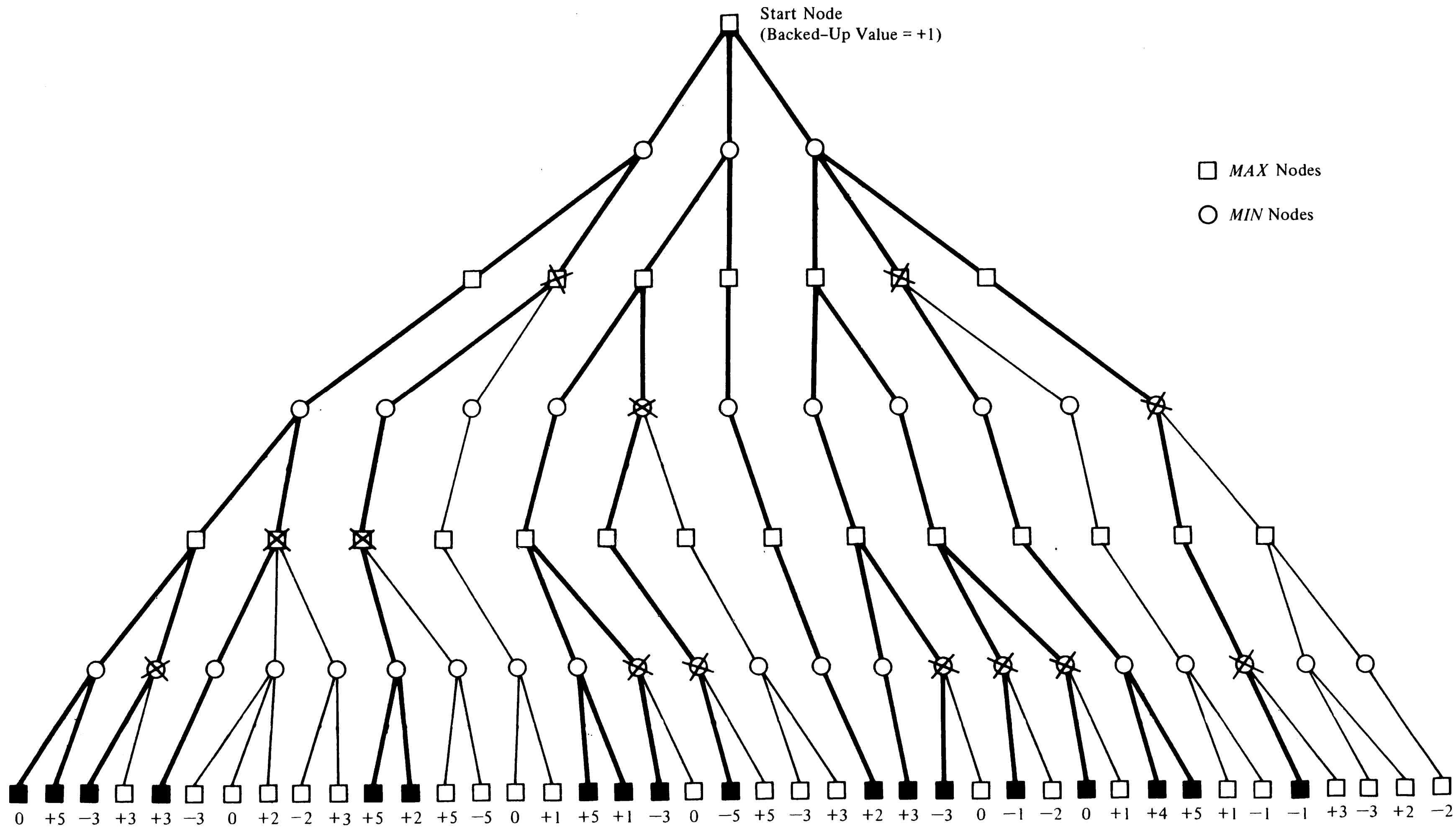

Fig. 3.12 An example illustrating the alpha-beta search procedure.

Why α-β?

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

Time complexity of α-β is about $O(b^{d/2})$

Progress on Games

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

Computers best: chess, checkers, Othello, backgammon, Scrabble

Computers competitive: bridge, crosswords, poker, small Go

Computers amateur: full Go

EOLQs

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- Break
- α-β Pruning
- α-β Pseudo-code
- Why α-β?
- Progress
- EOLQs

Please write down the most pressing question you have about the course material covered so far and put it in the box on your way out.

Thanks!