Why Study Games?

Games offer:

- Intellectual Engagement
- Abstraction
- Representability
- Performance Measure

- Not all games are suitable for AI research. We will restrict ourselves to **2 person perfect information** board games.

Game Playing as Search

- Playing a game involves searching for the **best move**.
- Board games clearly involve notions like **start state**, **goal state**, **operators**, etc. We can this usefully import problem solving techniques that we have already met.
- There are nevertheless important **differences** from standard search problems.

Special Characteristics of Game Playing Search

Main differences are **uncertainties** introduced by

- Presence of an **opponent**. One do not know what the opponent will do until he/she does it. Game playing programs must solve the contingency problem.

- **Complexity**. Most interesting games are simply **too complex to solve by exhaustive means**. Chess, for example, has an average branching factor of *35*. Uncertainty also arises from not having the resources to compute a move which is guaranteed to be the best.

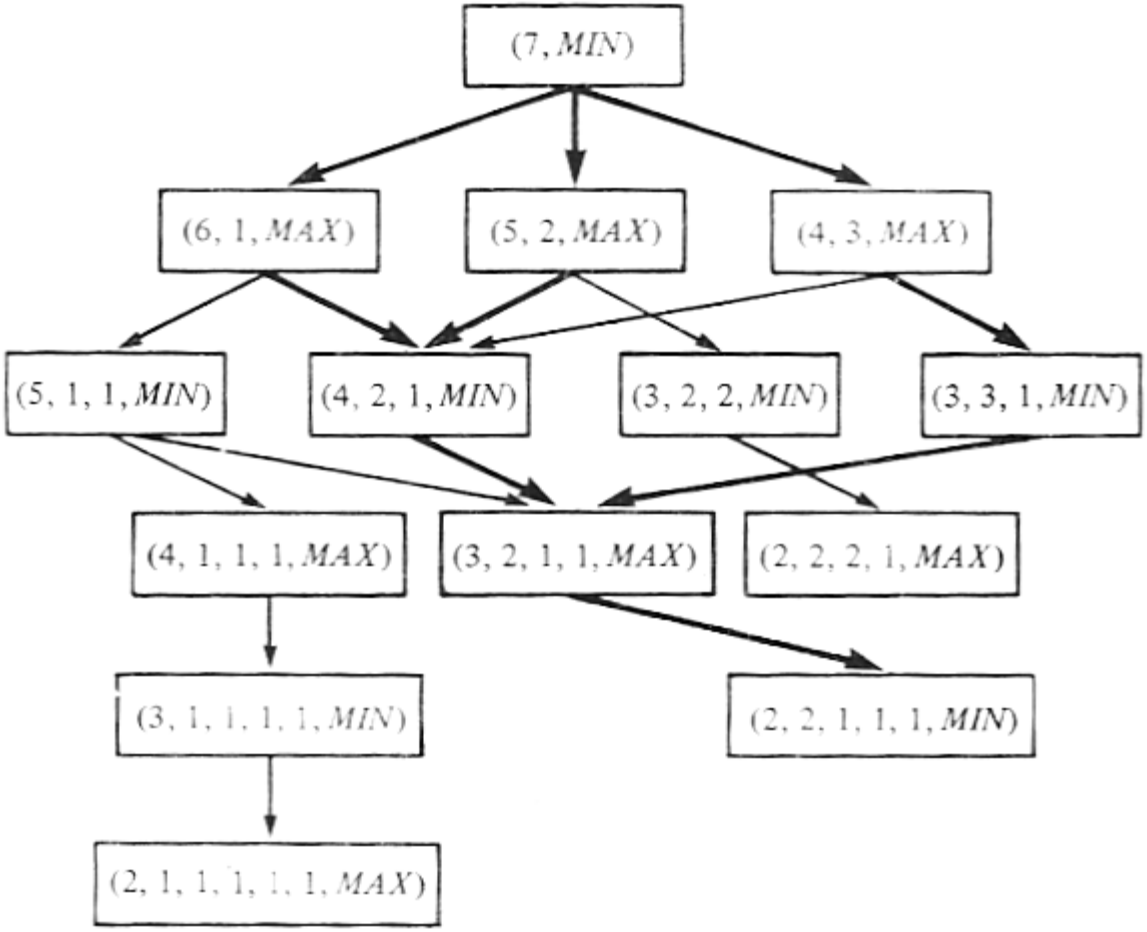


Fig. 3.7 A game graph for **Grundy's game**.

Ingredients of 2-Person Games

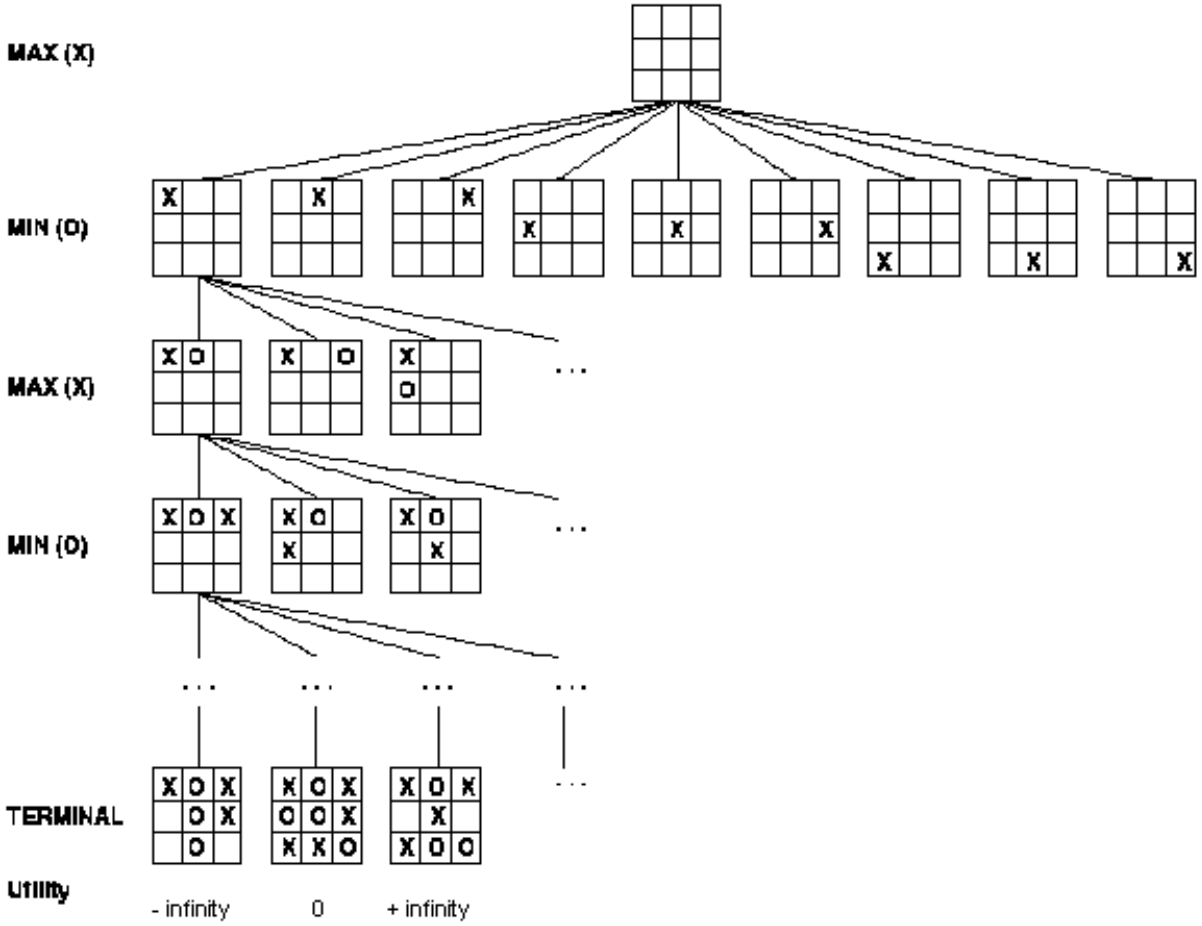
- **Players**: We call them Max and Min.
- **Initial State**: Includes board position and whose turn it is.
- **Operators**: These correspond to legal moves.
- **Terminal Test**: A test applied to a board position which determines whether the game is over. In chess, for example, this would be a checkmate or stalemate situation.
- **Utility Function**: A function which assigns a **numeric value** to a **terminal state**. For example, in chess the outcome is win (+1), lose (-1) or draw (0). Note that by convention, we always measure utility relative to Max.

Normal and Game Search Problem

- **Normal search problem**: **Max** searches for a sequence of moves yielding a winning position and then makes the first move in the sequence.

- **Game search problem**: Clearly, this is not feasible in in a game situation where **Min**'s moves must be taken into consideration. **Max** must devise a strategy which leads to a winning position no matter what moves Min makes.

Game Tree for Tic Tac Toe



A Perfect Decision Strategy Based on Minimaxing

1. Generate the whole game tree.

2. Apply the utility function to leaf nodes to get their values.

3. Use the utility of nodes at level n to derive the utility of nodes at level n-1.

4. Continue backing up values towards the root (one layer at a time).

5. Eventually the backed up values reach the top of the tree, at which point Max chooses the move that yields the highest value. This is called the minimax decision because it maximises the utility for Max on the assumption that Min will play perfectly to minimise it.

The Need for Imperfect Decision

- **Problem**: Minimax assumes the program has **time** to search to the terminal nodes
- **Solution**: **Cut off search earlier** and apply a **heuristic evaluation function** to the leaves.

Assigning Utilities: Evaluation Functions

For Tic-Tac-Toe: Let p be a board state and e(p) the evaluation function.

e(p) is:

1. +∞ if p is a winning position for Max
2. - ∞ if p is a winning position for Min
3. if p is neither then (number of complete rows, columns & diagnols open for Max) – (number of complete rows, columns & diagnols open for Min)

- In general weighted linear **evaluation function**
  $e = w_1 f_1 + w_2 f_2 + ... + w_n f_n$
  where f is some feature and w is a weighting that biases the contribution of that feature to the final score.

Tic-tac-toe using minimax

Look-ahead: 2 moves

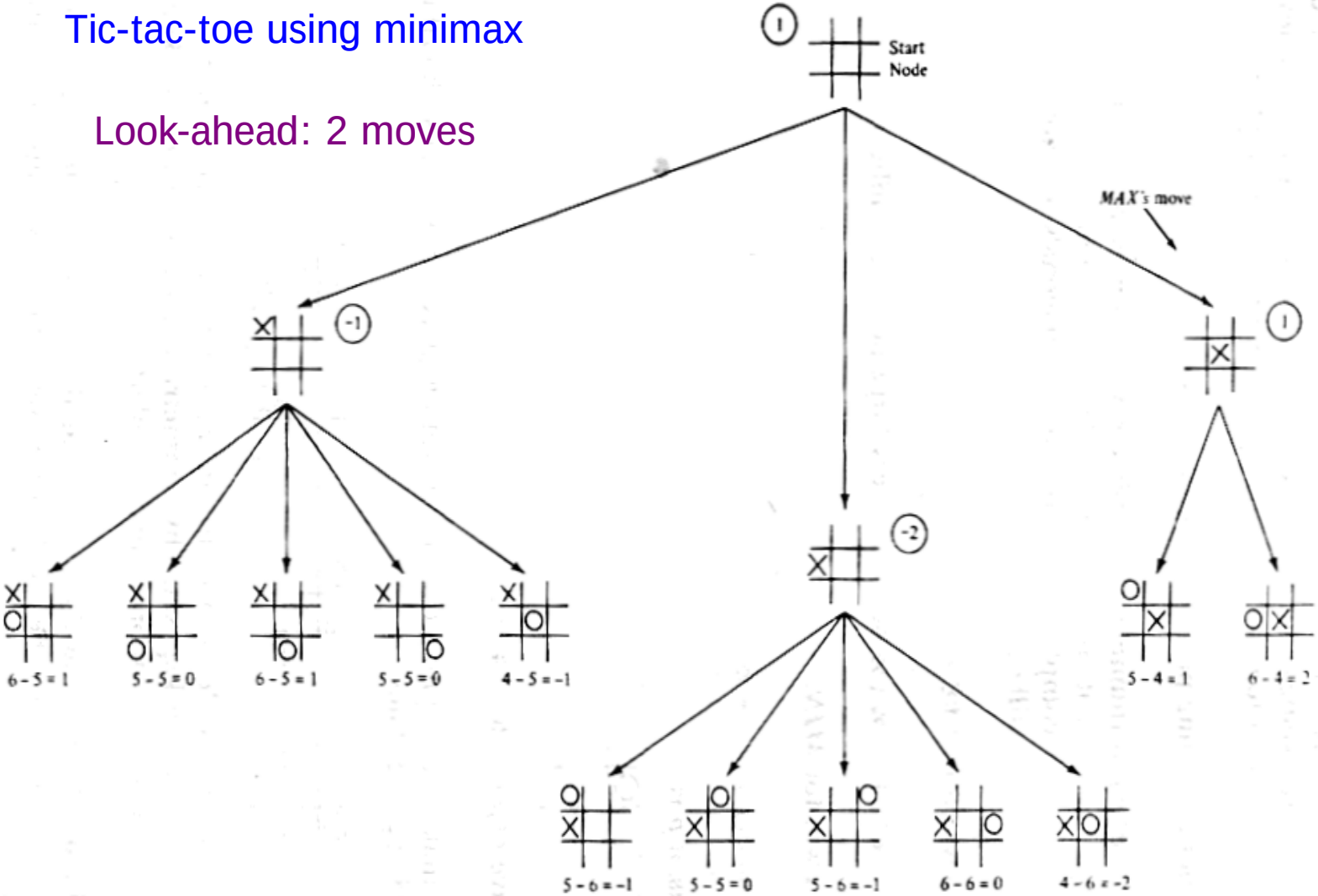


Fig. 3.8 Minimax applied to tic-tac-toe (stage 1).

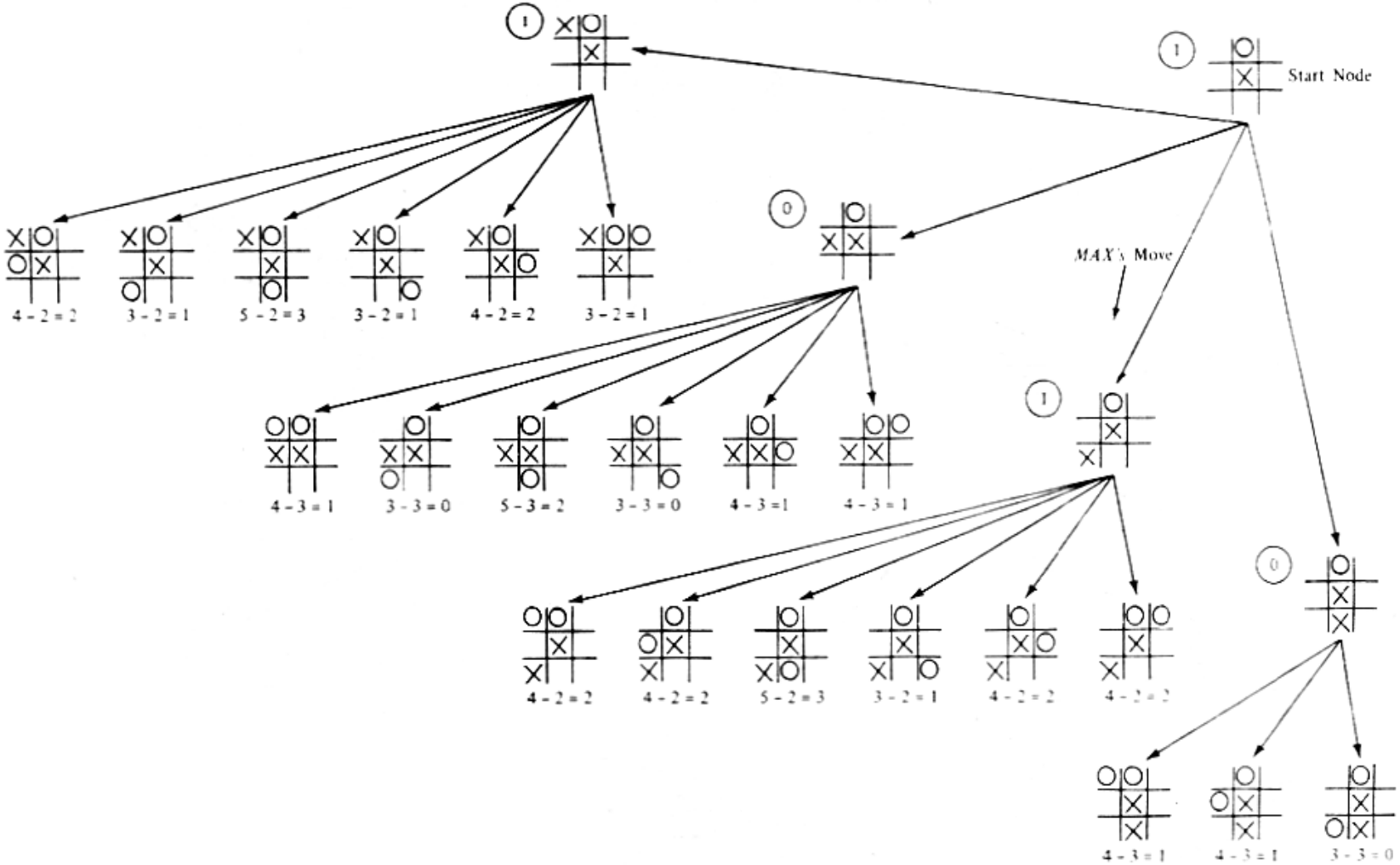


Fig. 3.9 Minimax applied to tic-tac-toe (stage 2).

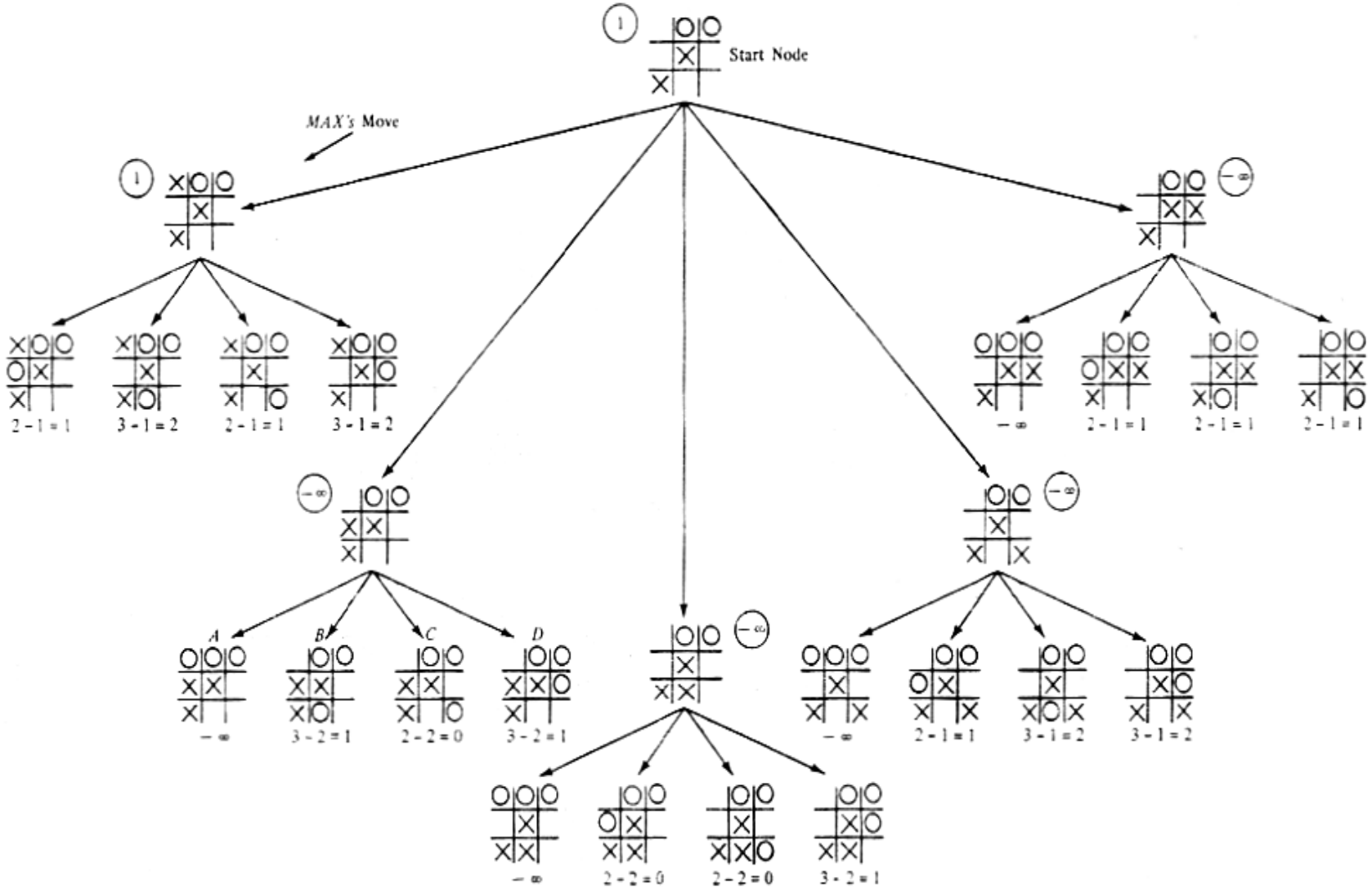


Fig. 3.10 Minimax applied to tic-tac-toe (stage 3).

Alpha-Beta Pruning

- **Alpha-beta pruning** is a technique that enables the Minimax algorithm **to ignore branches** that will **not contribute further** to the outcome of the search.

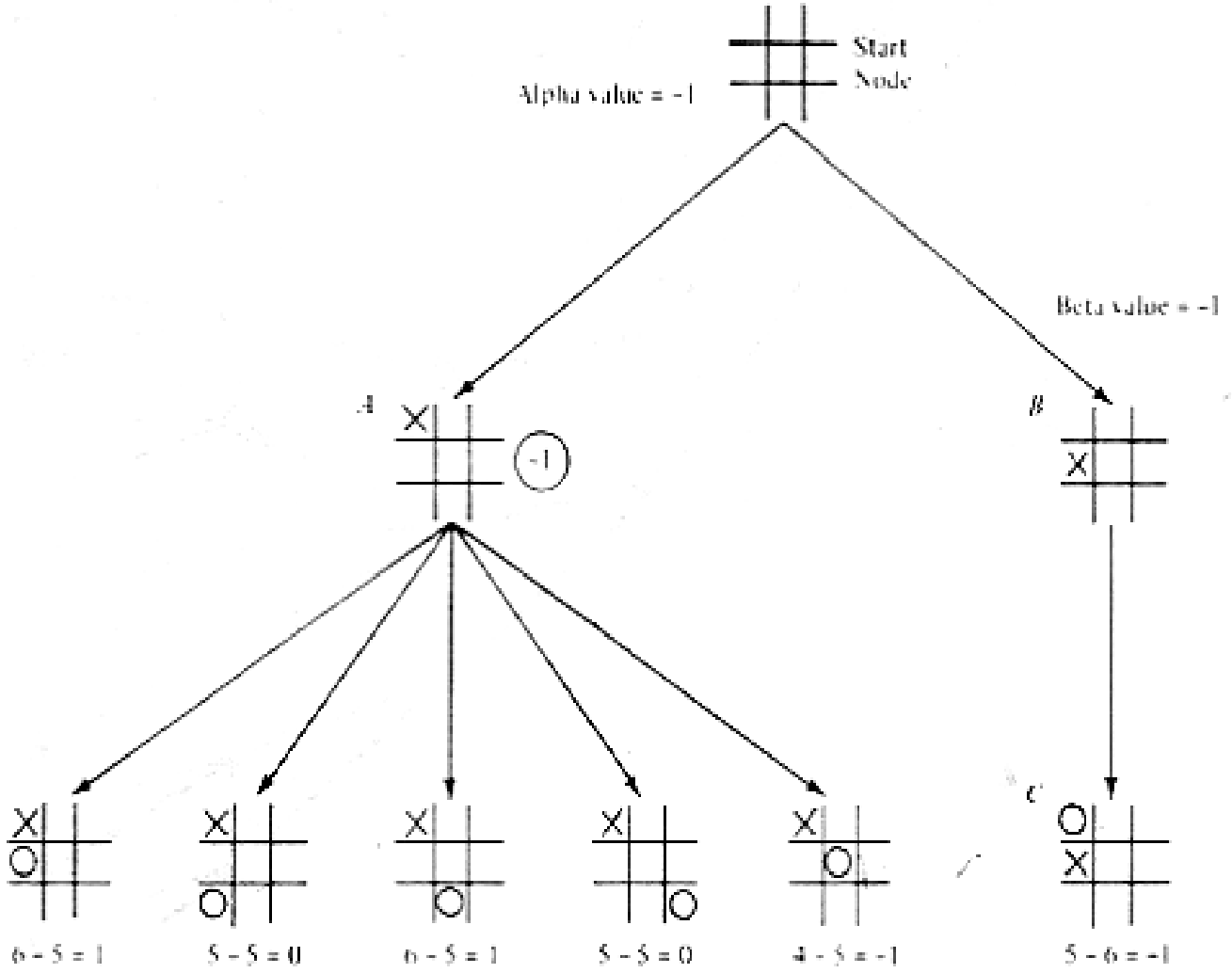


Fig. 3.11 Part of the first stage tic-tac-toe tree.

Suppose we do DFS search to do lookahead of 2 on the tic-tac-toe game. Suppose DFS has been done on A and all of its successors, but before B is Explored. A is give the value of -1. Now we know that the value propagated to the start node has a lower bound of -1. This is the alplha value for the start node.

Now assume B as`well as its first successor C has been visited. C has been assigned -1. This means the upper bound on the value propagated to B will be -1. This is the beta value for node B.

Observe that the eventual propagated value that will be assigned to B will never exceed -1. So none of the other children of B will have any impact on the alpha value of the start node.

Alpha-beta cutoff

- The alpha value of Max nodes can never decrease
- The beta values of Min nodes can never increase.
- So search can be discontinued below any Min node having a beta value <= alpha value of any of its Max node ancestors. – alpha cutoff
- Search can be discontinued below any Max node having an alpha value >= beta value of any of its Min node ancestors. – beta cutoff

Setting Alpha/Beta Values

- The alpha value of a Max node is set equal to the current largest final backed-up of its successors
- The beta value of a Min node is set equal to the current smallest final backed-up value of its successors.

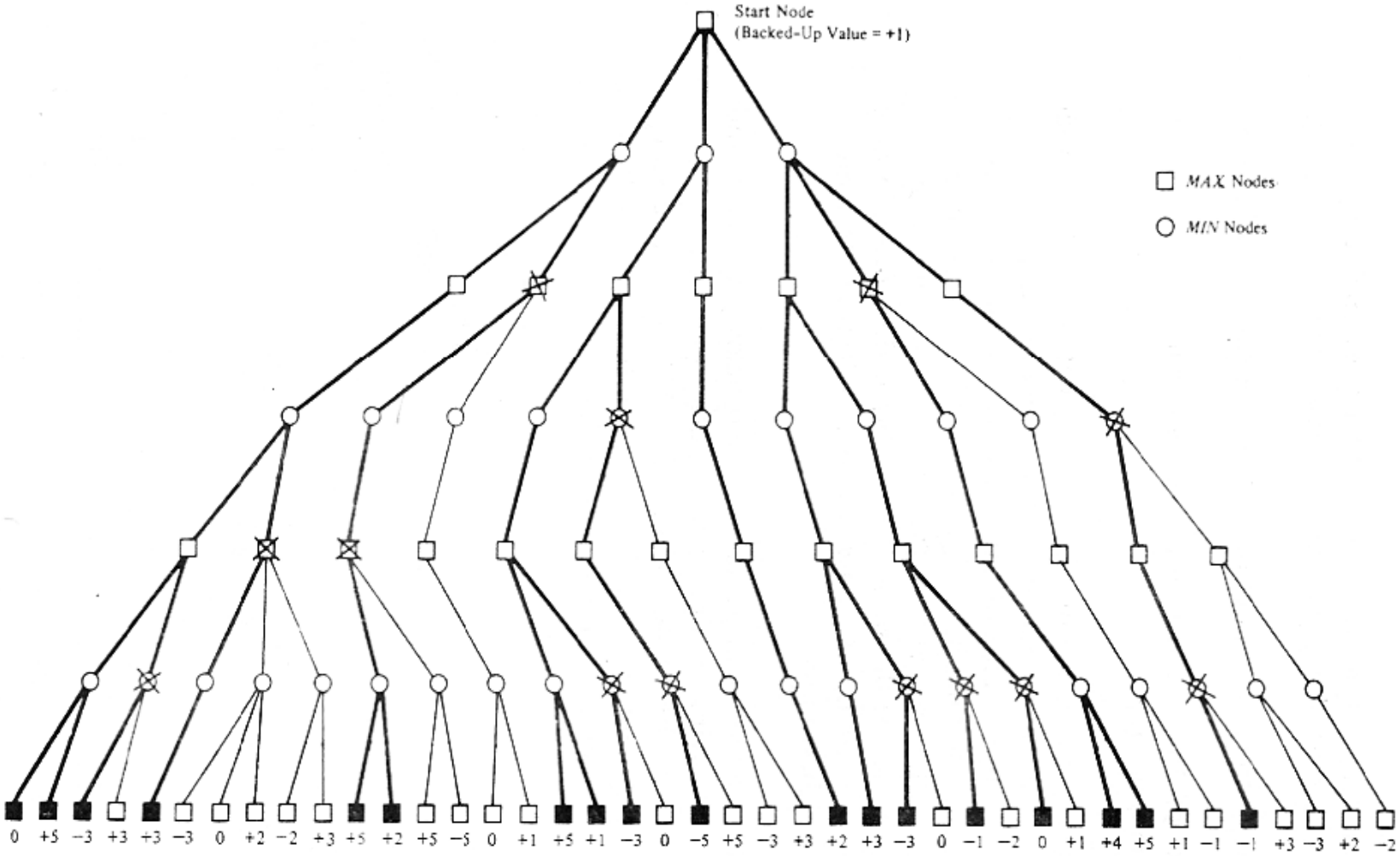


Fig. 3.12 An example illustrating the alpha-beta search procedure.