Game Playing

Chapter 6

Additional references for the slides:

Luger's AI book (2005).

Robert Wilensky's CS188 slides:
www.cs.berkeley.edu/~wilensky/cs188/lectures/index.html

Tim Huang's slides for the game of Go.

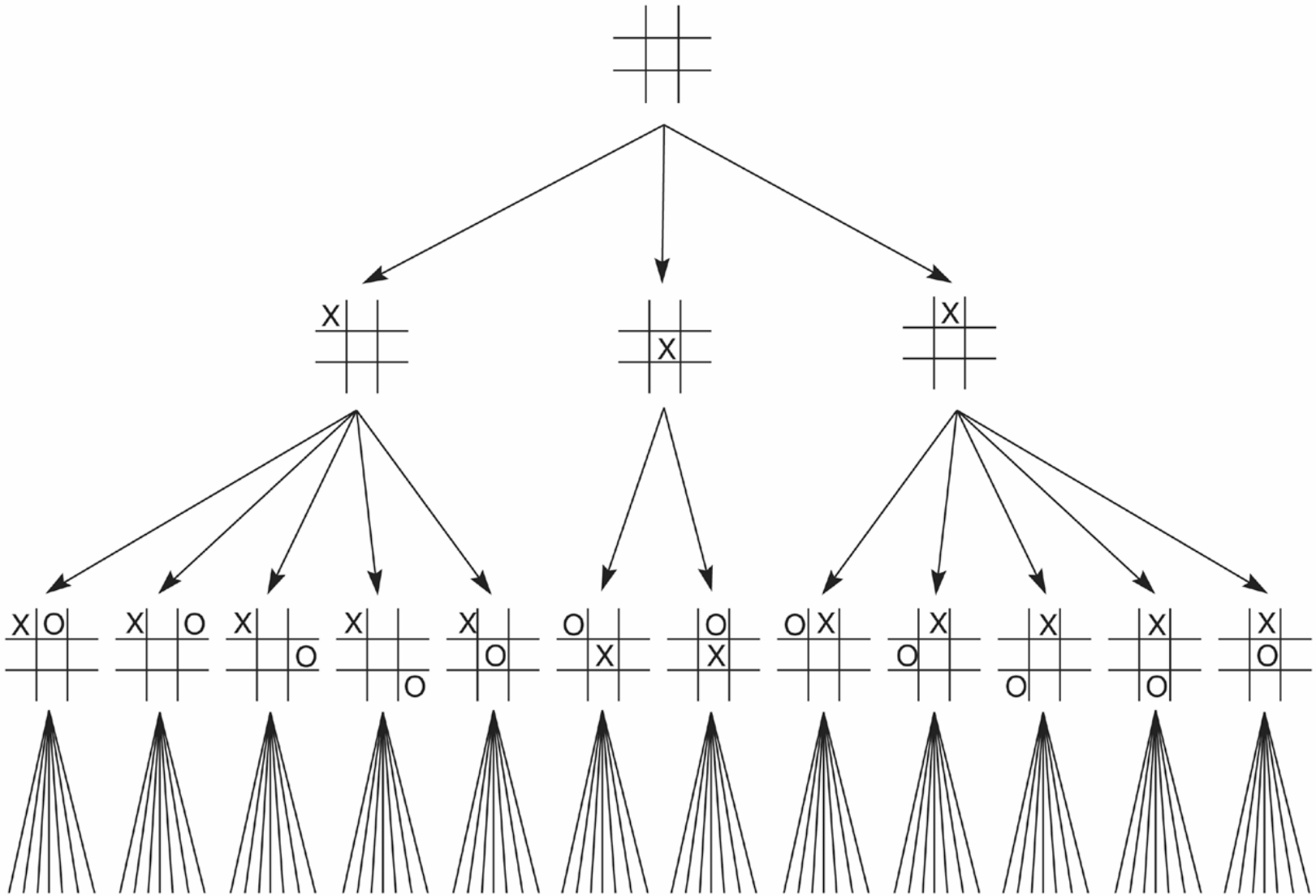
Game playing

Games have always been an important application area for heuristic algorithms. The games that we will look at in this course will be two-person board games such as Tic-tac-toe, Chess, or Go.

Types of Games

	Deterministic	Chance
Perfect information	Chess, Checkers Go, Othello	Backgammon Monopoly
Imperfect Information	Battleships Blind tictactoe	Bridge, Poker, Scrabble, Nuclear War

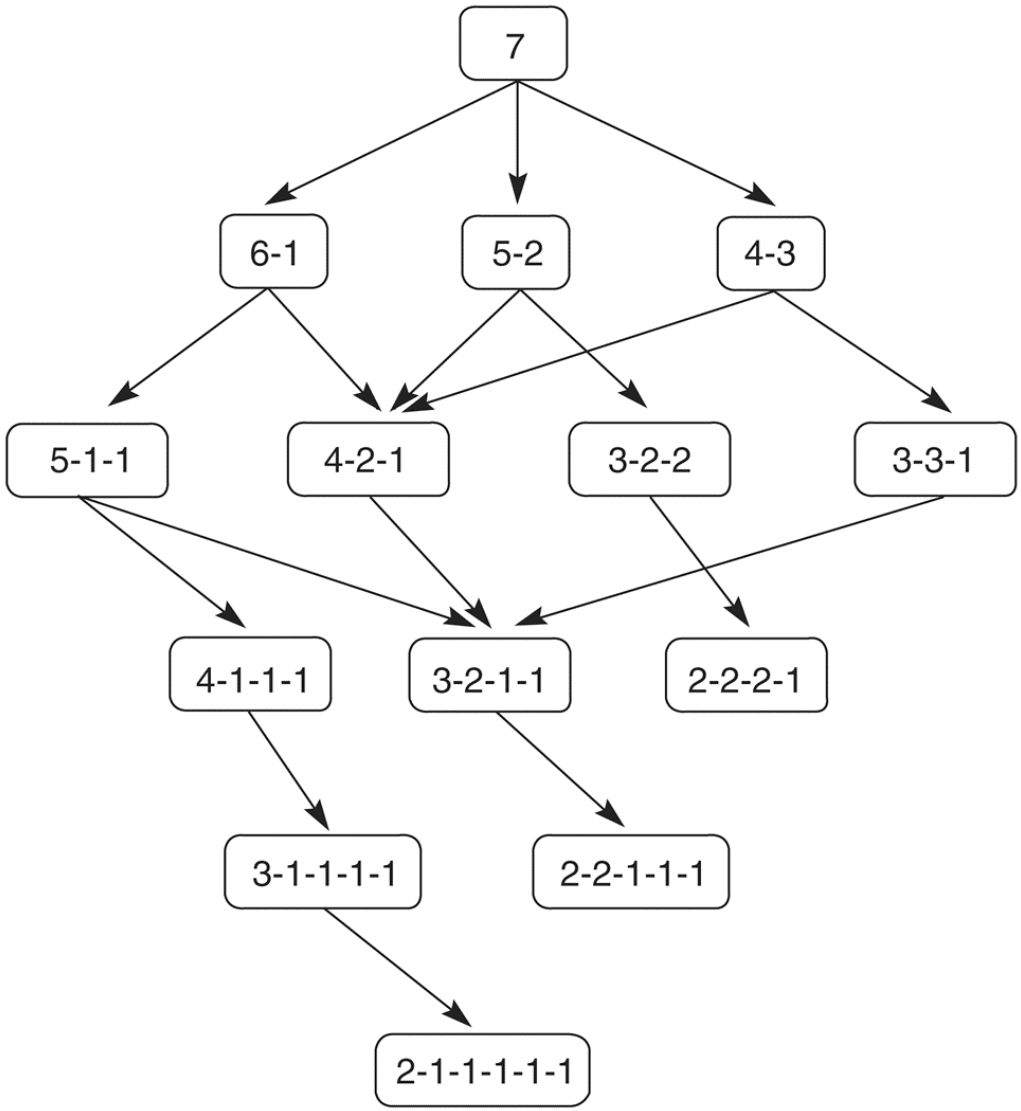
Tic-tac-toe state space reduced by symmetry (2-player, deterministic, turns)



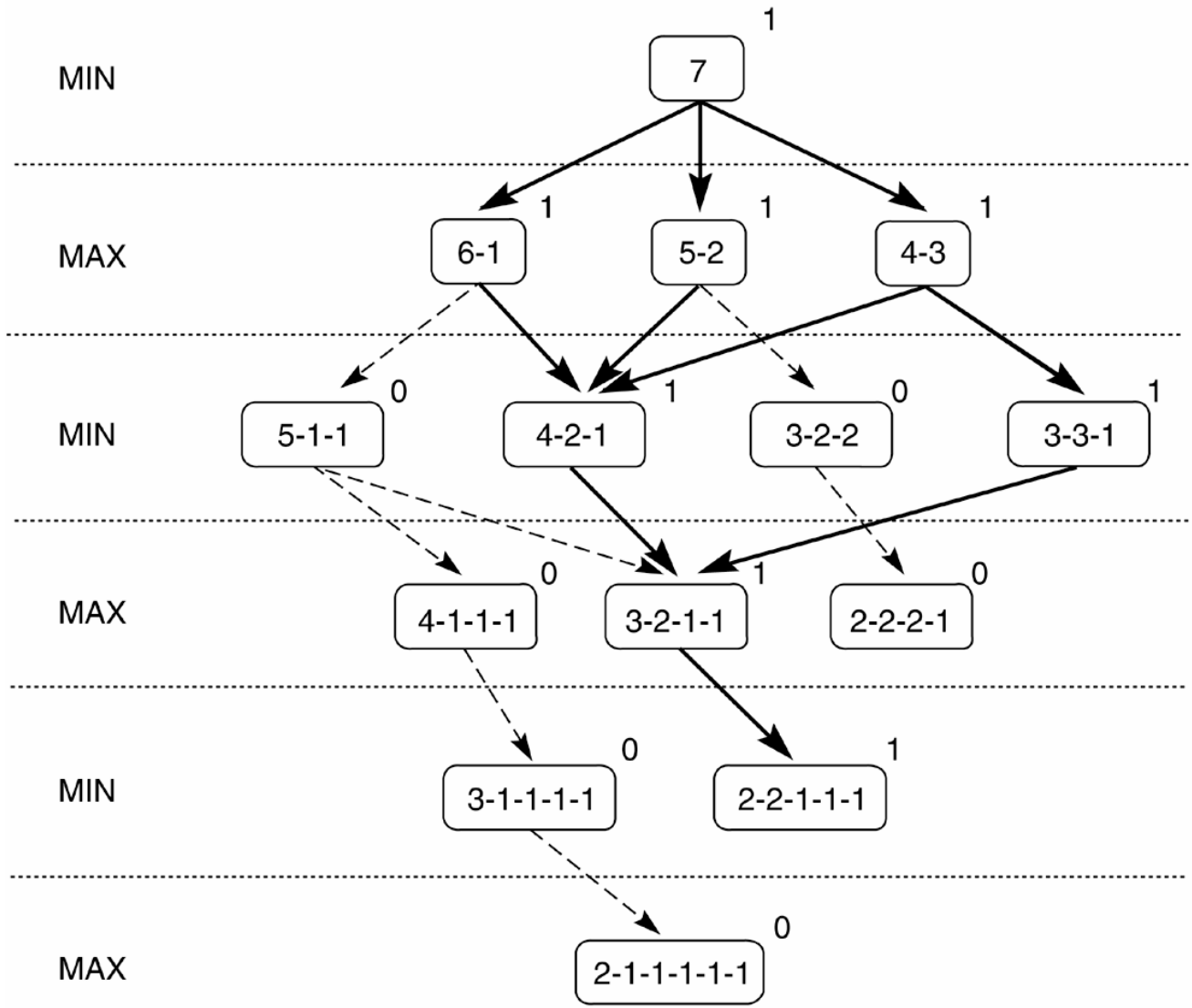
A variant of the game nim

- A number of tokens are placed on a table between the two opponents
- A move consists of dividing a pile of tokens into two nonempty piles of different sizes
- For example, 6 tokens can be divided into piles of 5 and 1 or 4 and 2, but not 3 and 3
- The first player who can no longer make a move loses the game
- For a reasonable number of tokens, the state space can be exhaustively searched

State space for a variant of nim



Exhaustive minimax for the game of nim



Two people games

- One of the earliest AI applications
- Several programs that compete with the best human players:
  - Checkers: beat the human world champion
  - Chess: beat the human world champion
  - Backgammon: at the level of the top handful of humans
  - Go: no competitive programs (? In 2008)
  - Othello: good programs
  - Hex: good programs

Search techniques for 2-person games

- The search tree is slightly different: It is a *two-ply tree* where levels alternate between players
- Canonically, the first level is “us” or the player whom we want to win.
- Each final position is assigned a payoff:
  - win (say, 1)
  - lose (say, -1)
  - draw (say, 0)
- We would like to maximize the payoff for the first player, hence the names MAX & MINIMAX

The search algorithm

- The root of the tree is the current board position, it is MAX's turn to play
- MAX generates the tree as much as it can, and picks the best move assuming that Min will also choose the moves for herself.
- This is the ***Minimax algorithm*** which was invented by Von Neumann and Morgenstern in 1944, as part of game theory.
- The same problem with other search trees: the tree grows very quickly, exhaustive search is usually impossible.

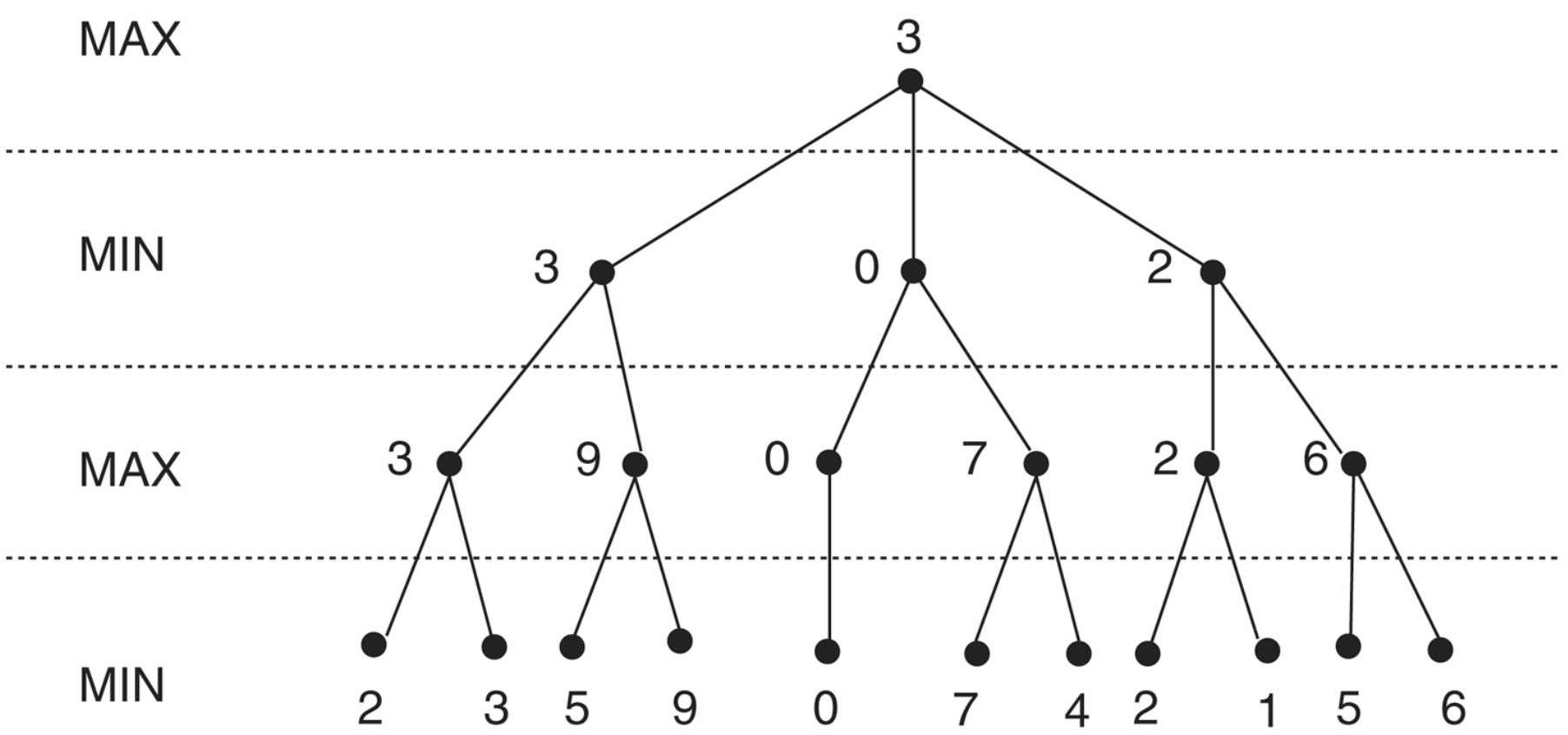
Minimax

Perfect play for deterministic, perfect information games

Idea: choose to move to the position with the highest mimimax value

Best achievable payoff against best play

Minimax applied to a hypothetical state space (Luger Fig. 4.15)



Minimax algorithm

Function Minimax-Decision(*state*)

returns *an action*

inputs: *state*, current state in game

return the a in Actions(*state*) maximizing

MIN-VALUE(RESULT(*a,state*))

Max-value algorithm

Function MAX-VALUE(*state*)

returns *a utility value*

inputs: *state*, current state in game

if TERMINAL-TEST(*state*) **then**

return UTILITY(*state*)

v← -∞

for each <*a, s*> in SUCCESSORS(*state*) **do**

v ← MAX(*v*, MIN-VALUE(*s*))

return *v*

Min-value algorithm

Function MIN-VALUE(*state*)

returns *a utility value*

inputs: *state*, current state in game

if TERMINAL-TEST(*state*) **then**

return UTILITY(*state*)

v← ∞

for each <*a, s*> in SUCCESSORS(*state*) **do**

v ← MIN(*v*, MAX-VALUE(*s*))

return *v*

Properties of minimax

Complete: Yes, if the 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 ≈ 35, m ≈ 100 for "reasonable" games

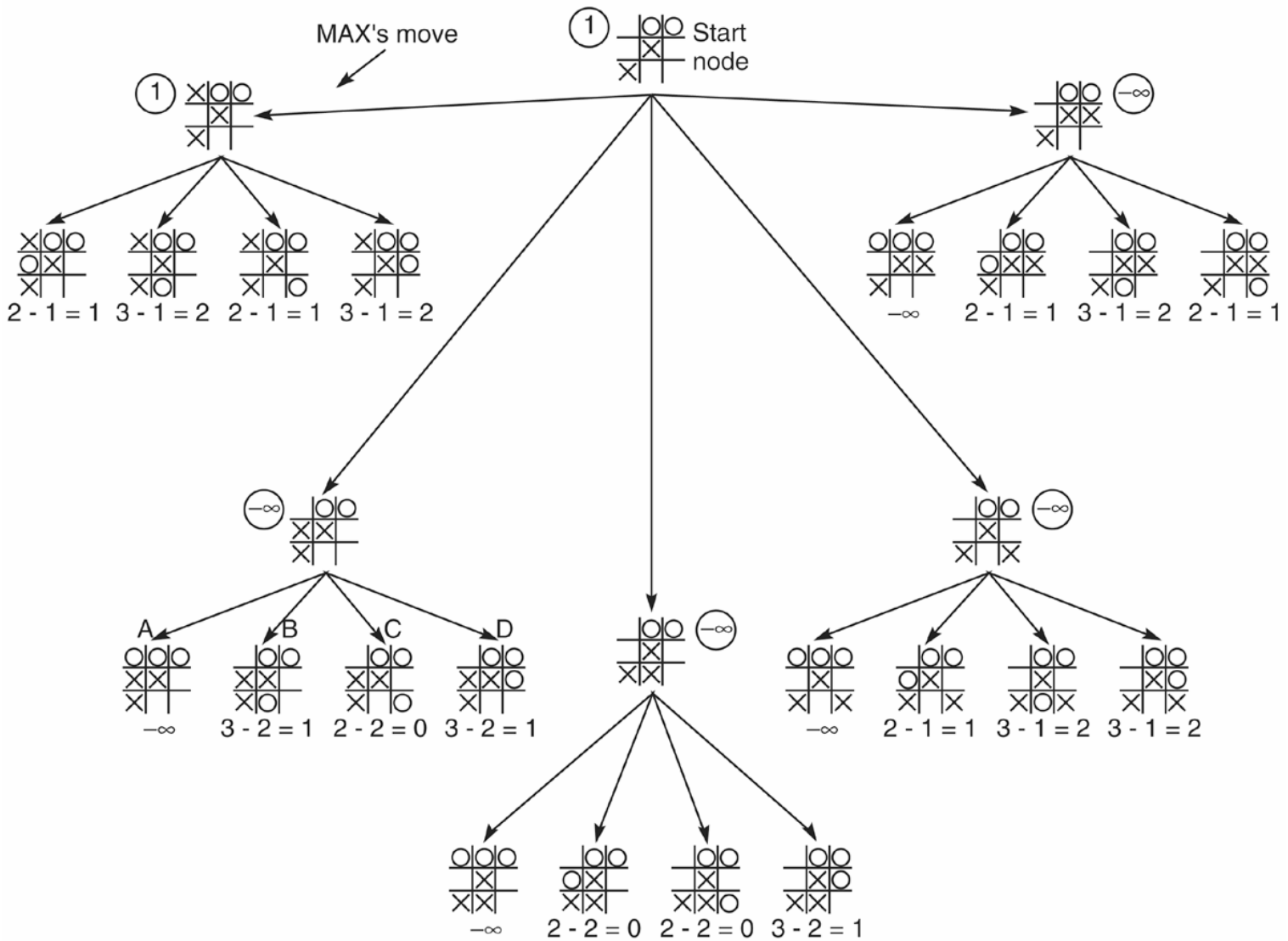
exact solution is completely infeasible

But do we need to explore every path?

Using the Minimax algorithm

- MAX generates the full search tree (up to the leaves or terminal nodes or final game positions) and chooses the best one:
  win or tie
- To choose the best move, values are propogated upward from the leaves:
  - MAX chooses the maximum
  - MIN chooses the minimum
- This assumes that the full tree is not prohibitively big
- It also assumes that the final positions are easily identifiable
- We can make these assumptions for now, so let's look at an example

Two-ply minimax applied to X's move near the end of the game (Nilsson, 1971)

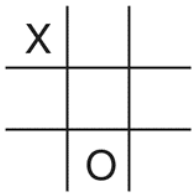


Using cut-off points

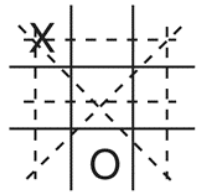
- Notice that the tree was not generated to full depth in the previous example
- When time or space is tight, we can't search exhaustively so we need to implement a cut-off point and simply not expand the tree below the nodes who are at the cut-off level.
- But now the leaf nodes are not final positions but we still need to evaluate them:

  use heuristics
- We can use a variant of the "most wins" heuristic

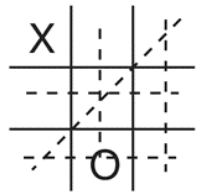
Heuristic measuring conflict



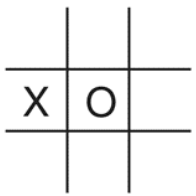
X has 6 possible win paths:



O has 5 possible wins:

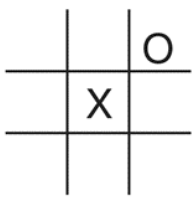


E(n) = 6 – 5 = 1



X has 4 possible win paths;
O has 6 possible wins

E(n) = 4 – 6 = –2



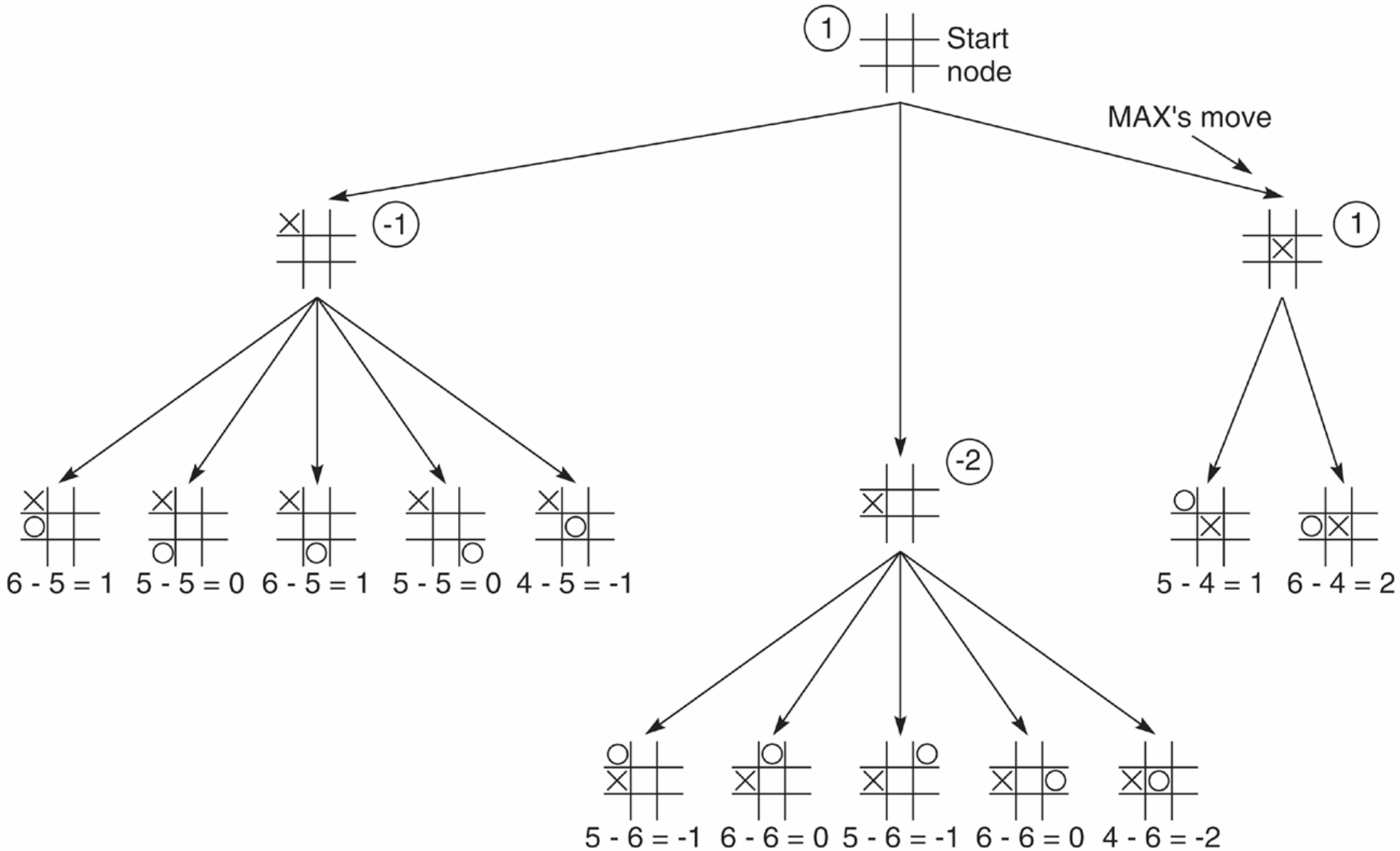
X has 5 possible win paths;
O has 4 possible wins

E(n) = 5 – 4 = 1

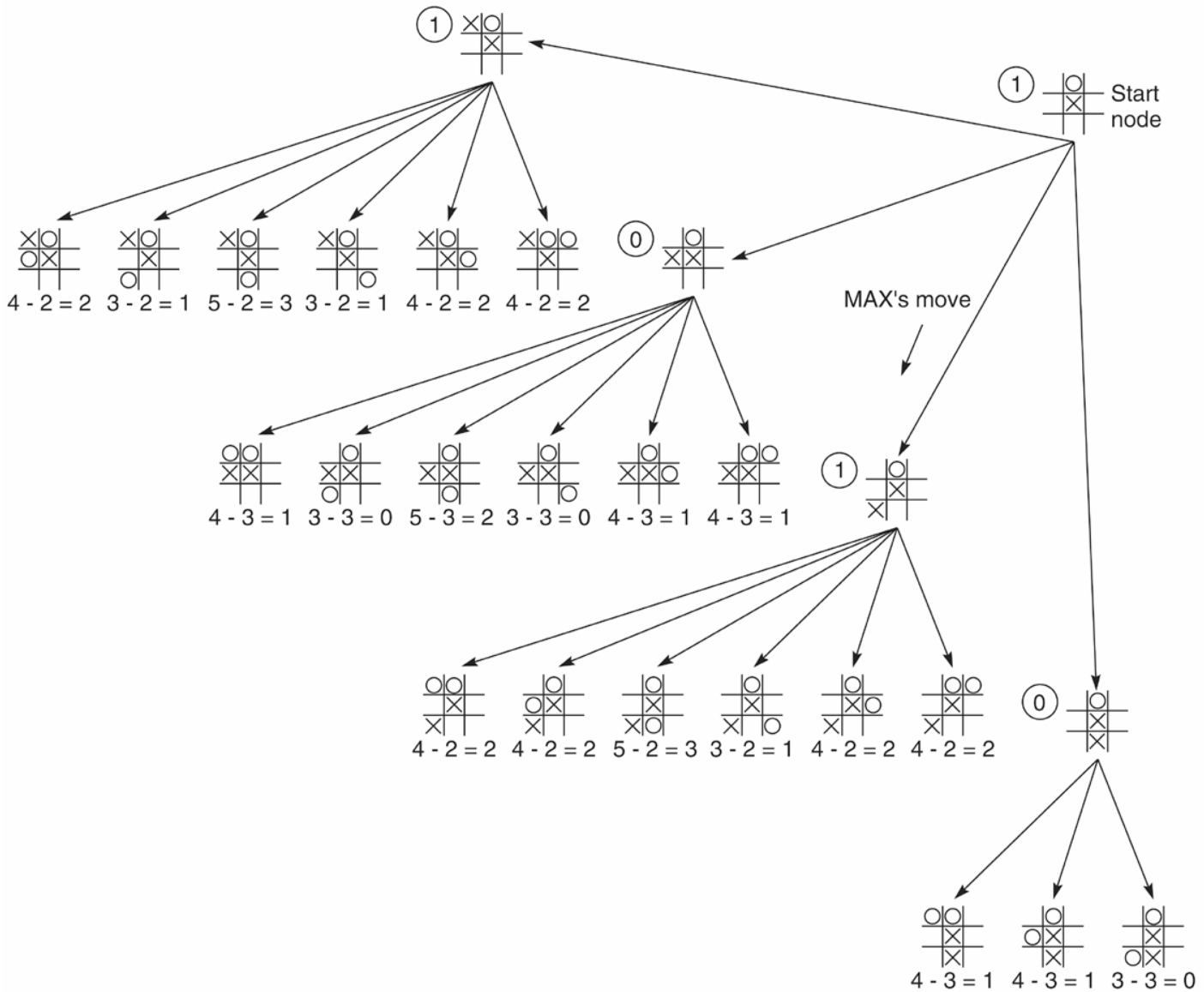
Calculation of the heuristic

- E(n) = M(n) – O(n) where
  - M(n) is the total of My (MAX) possible winning lines
  - O(n) is the total of Opponent's (MIN) possible winning lines
  - E(n) is the total evaluation for state n
- Take another look at the previous example
- Also look at the next two examples which use a cut-off level (a.k.a. *search horizon*) of 2 levels

Two-ply minimax applied to the opening move of tic-tac-toe (Nilsson, 1971)



Two-ply minimax and one of two possible second MAX moves (Nilsson, 1971)



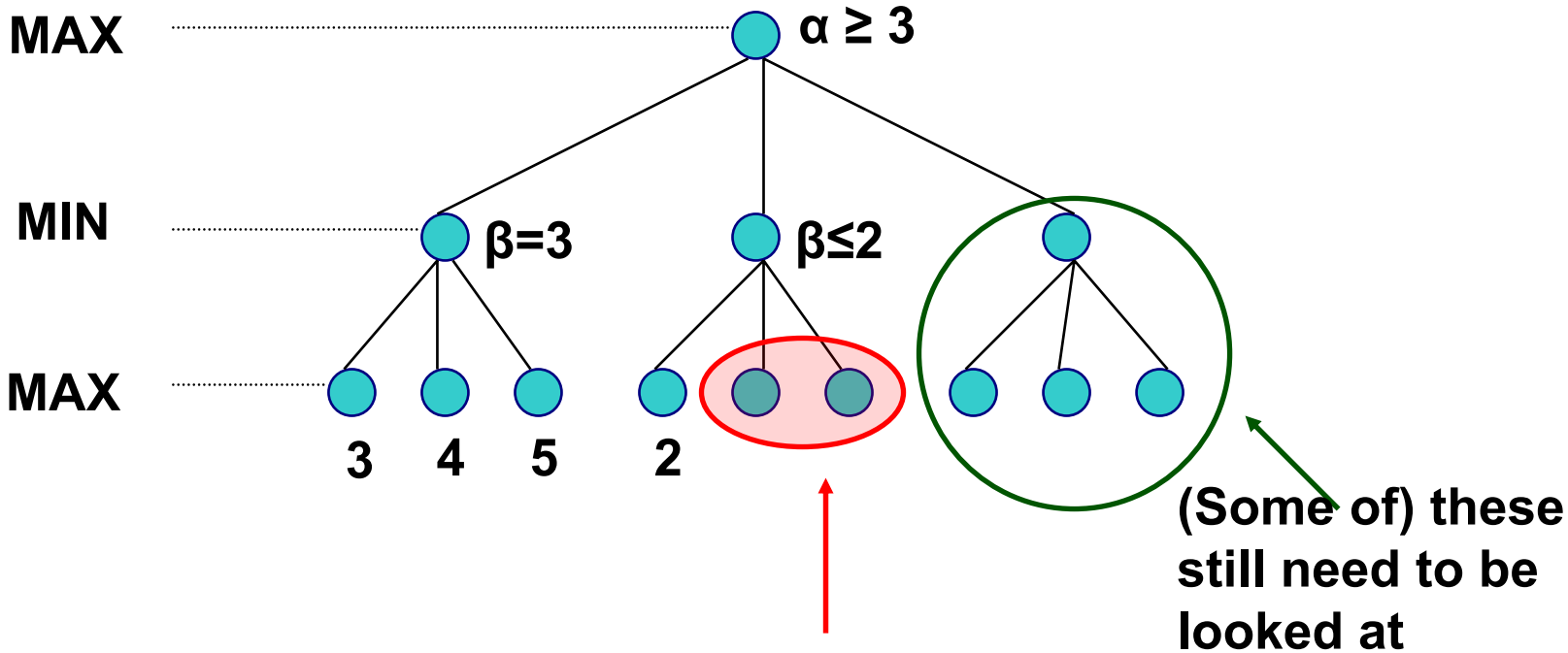
Pruning the search tree

- The technique is called *alpha-beta pruning*
- Basic idea: if a portion of the tree is obviously good (bad) don't explore further to see how terrific (awful) it is
- Remember that the values are propagated upward. Highest value is selected at MAX's level, lowest value is selected at MIN's level
- Call the values at MAX levels *α values*, and the values at MIN levels *β values*

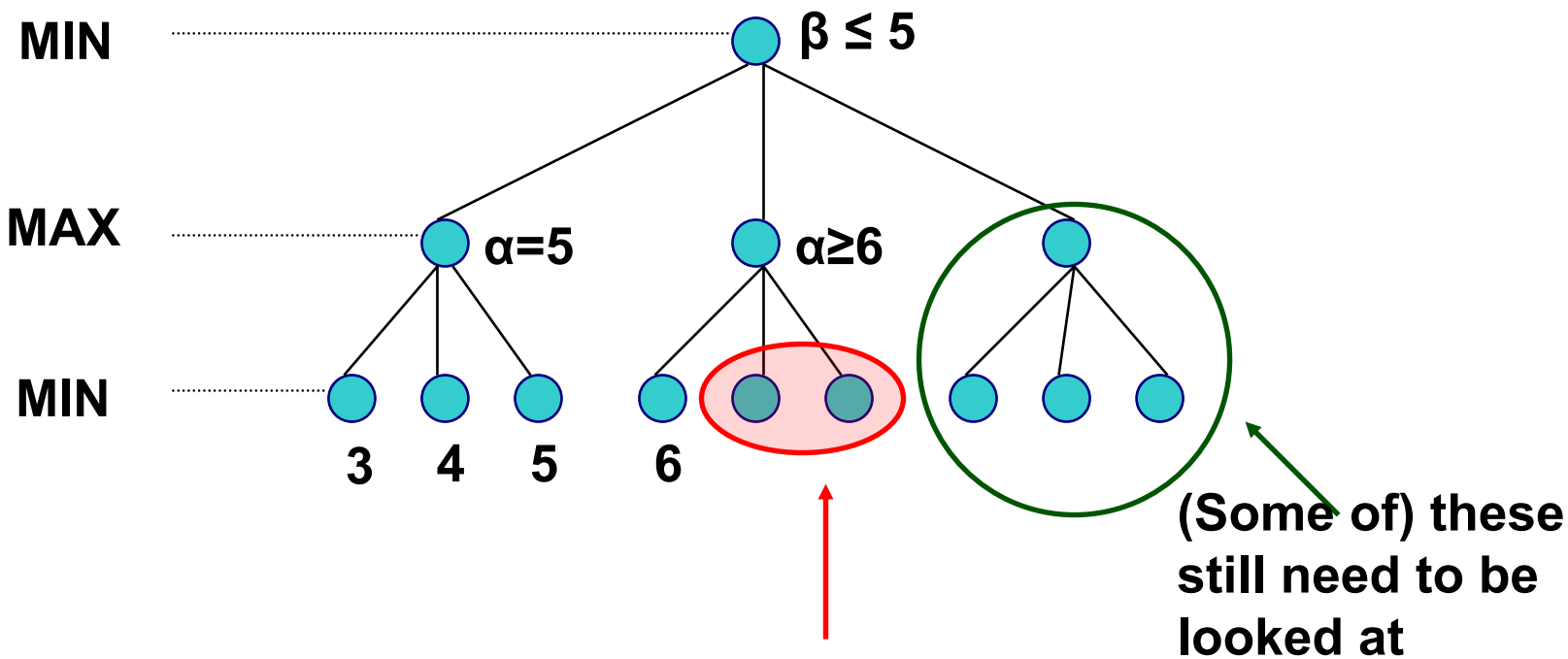
The rules

- **Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors**
- **Search can be stopped below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN node ancestors**

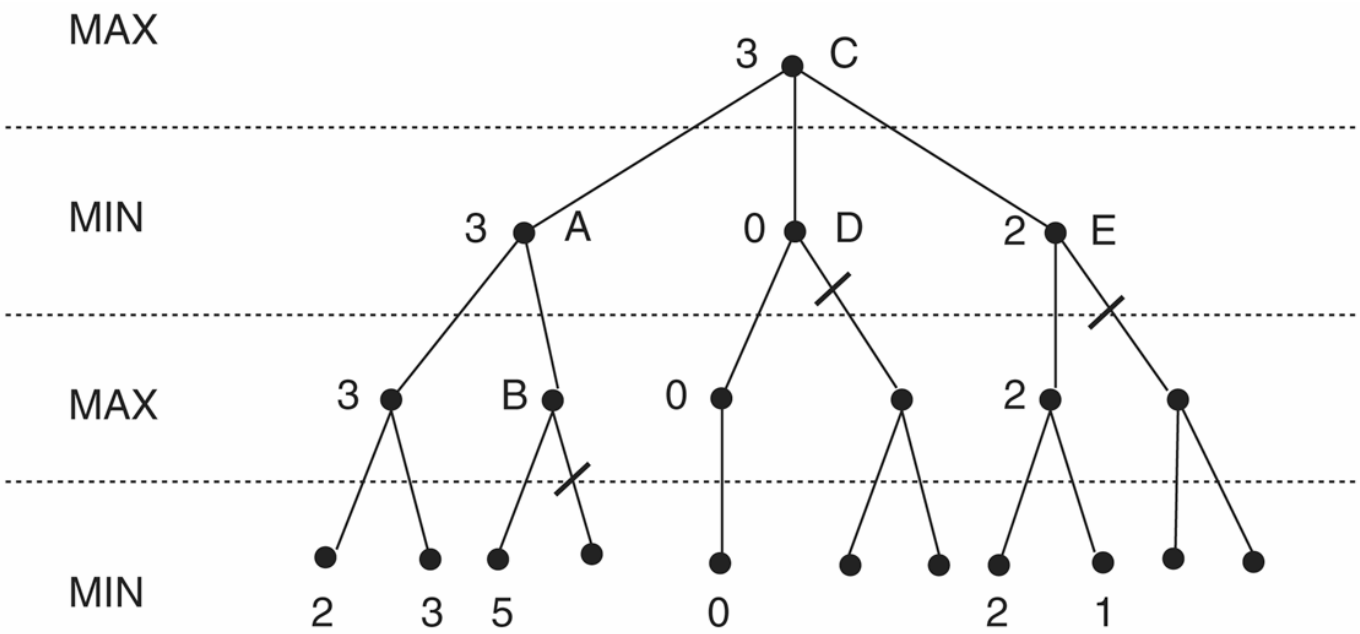
Example with MAX



Example with MIN



Alpha-beta pruning applied to the state space of Fig. 4.15



A has $\beta = 3$ (A will be no larger than 3)

B is β pruned, since 5 > 3

C has $\alpha = 3$ (C will be no smaller than 3)

D is α pruned, since 0 < 3

E is α pruned, since 2 < 3

C is 3

Properties of α-β

Pruning does not affect final result

Good move ordering improves effectiveness of pruning

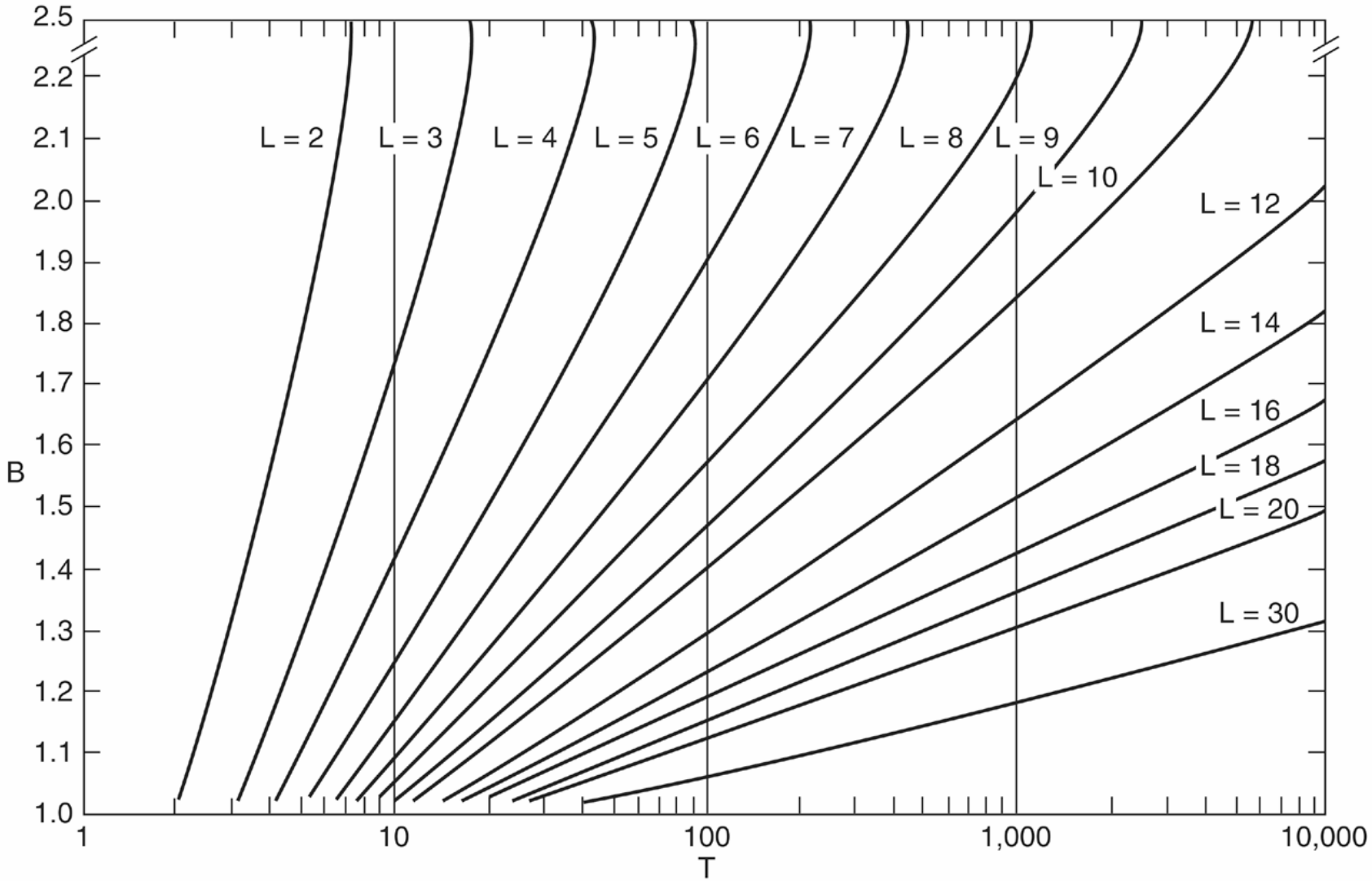
With "perfect ordering," doubles solvable depth

time complexity = $O(b^{m/2})$

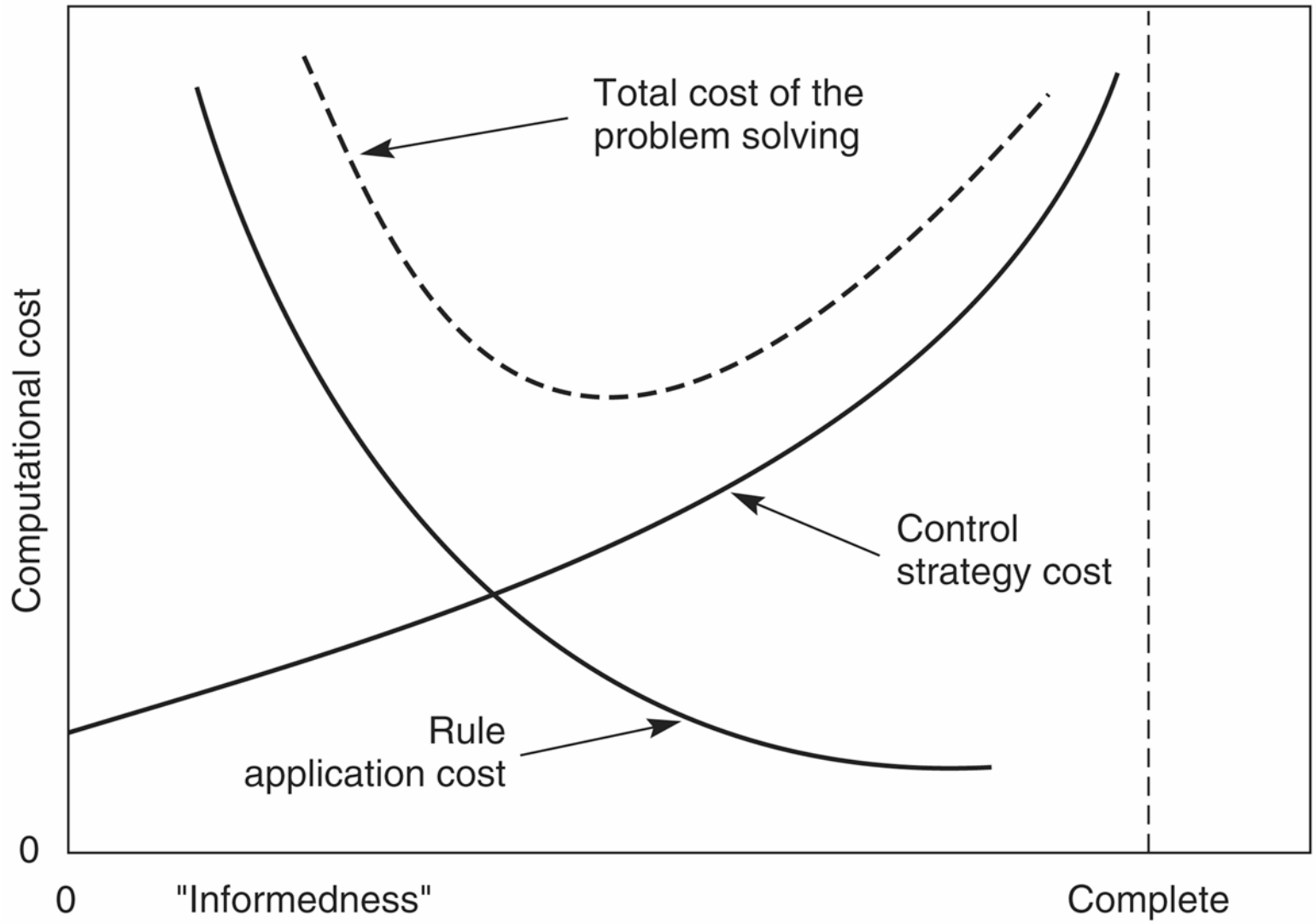
A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

Unfortunately, 35^{50} is still impossible!

Number of nodes generated as a function of branching factor B, and solution length L (Nilsson, 1980)



Informal plot of cost of searching and cost of computing heuristic evaluation against heuristic informedness (Nilsson, 1980)



Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

- perfection is unattainable (must approximate)
- good idea to think about what to think about
- expanding the ideas to uncertain situations (games)
- with imperfect information, optimal decisions depend on information state, not real state