L05. Search for Two Player Games
(if You Haven't got X GPUs and a metric sh*t-ton of data)

Jeremy Pitt
Department of Electrical & Electronic Engineering
Email: j.pitt _at_ imperial _dot_ ac _dot_ uk
Phone: (int) 46318
Office: 1010 EEE

Games: Brief Overview

- **Search in problem solving (puzzles, route finding...) is under full control**
  - Can always choose to which state go next
- **Two player games are more complicated**
  - Existence of a *hostile* opponent
  - The opponent *chooses* next state
  - This selection is essentially *unpredictable*
- **Examples of two player games**
  - Tic-tac-toe
  - Draughts (checkers)
  - Chess
  - ...

Games: Brief Overview

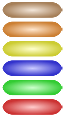
- **Consequences**
  - Greater difficulty in developing search algorithms
  - Opportunity for developing heuristics
  - Important application area (game industry)
- **Looking for a forced win**
  - I make a move
  - For every move my opponent makes
    - There is a sequence of moves leading to a goal state
  - Various different ways of doing this
    - General graph search algorithm can be used

Exhaustive Search

- **Games with a state space small enough can be exhaustively searched**
  - Analyse the entire state space
  - Systematically search all possible moves and counter-moves by opponent
- **Primary difficulty in such cases is in accounting for the actions of the opponent**
- **One approach to this is to assume that:**
  - The opponent has the same knowledge of the state space
  - Applies that knowledge in a consistent effort to win the game

Example: Game of Nim (variant)

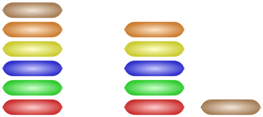
- **Game played with a pile of tokens** (initially only one)
    - Local rules: one player specifies the size of the pile; the other player goes first
- **At each turn, the player must divide one pile into two non-empty piles of different sizes**



6

Example: Game of Nim (variant)

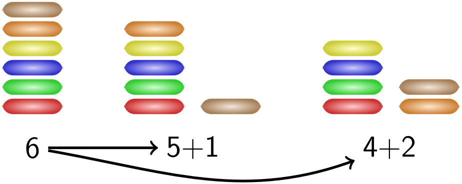
- **Game played with a pile of tokens** (initially only one)
  - Local rules: one player specifies the size of the pile; the other player goes first
- **At each turn, the player must divide one pile into two non-empty piles of different sizes**



$6 \longrightarrow 5+1$

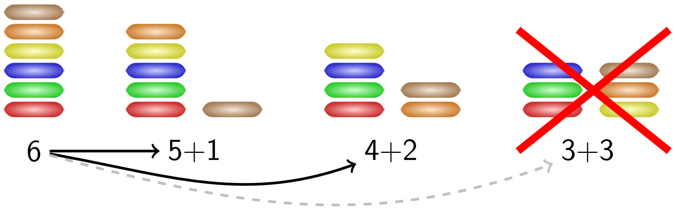
Example: Game of Nim (variant)

- **Game played with a pile of tokens** (initially only one)
  - Local rules: one player specifies the size of the pile; the other player goes first
- **At each turn, the player must divide one pile into two non-empty piles of different sizes**



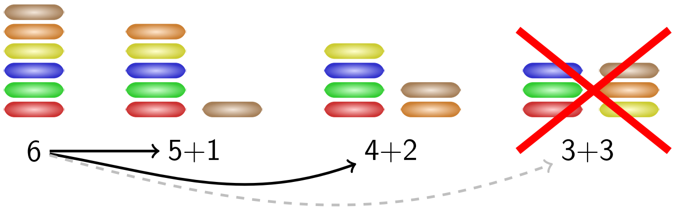
Example: Game of Nim (variant)

- **Game played with a pile of tokens** (initially only one)
  - Local rules: one player specifies the size of the pile; the other player goes first
- **At each turn, the player must divide one pile into two non-empty piles of different sizes**



Example: Game of Nim (variant)

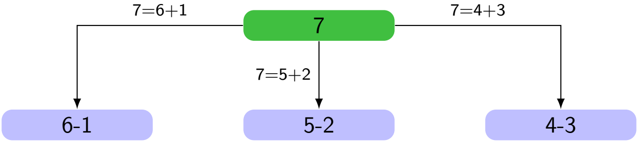
- **Game played with a pile of tokens** (initially only one)
  - Local rules: one player specifies the size of the pile; the other player goes first
- **At each turn, the player must divide one pile into two non-empty piles of different sizes**



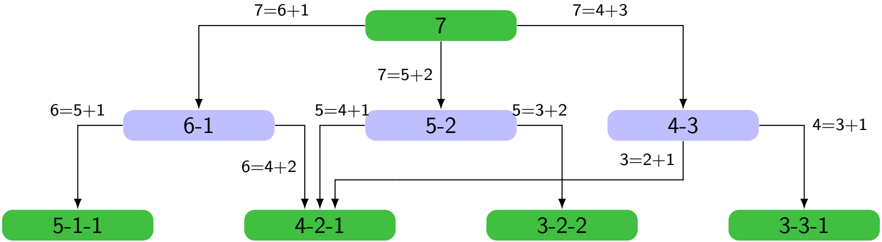
- **First player unable to make a move, loses** (i.e. when there are only piles of size 1 and 2)
- **For a reasonably small number of tokens, state space can be exhaustively searched**
- **Is it better to specify the size of the pile, or to make the first move?**

Nim: State Space with 7 tokens

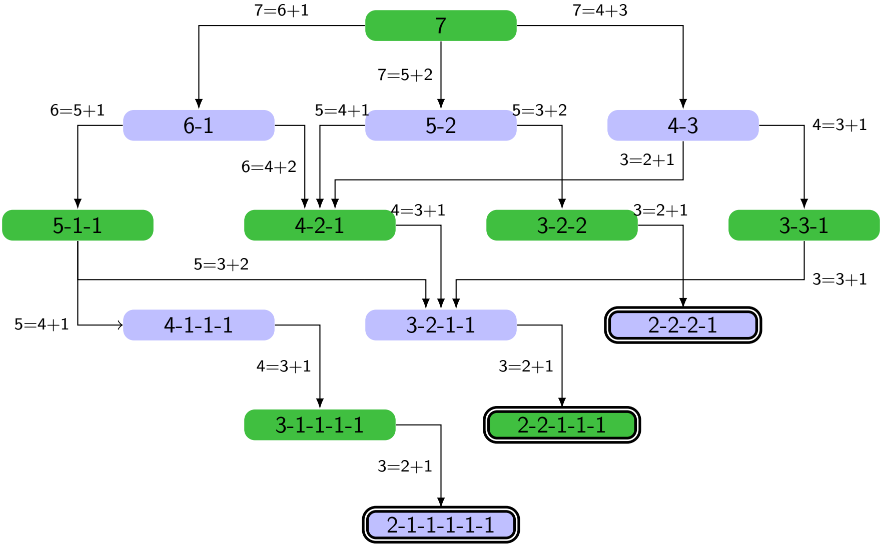
Nim: State Space with 7 tokens



Nim: State Space with 7 tokens



Nim: State Space with 7 tokens



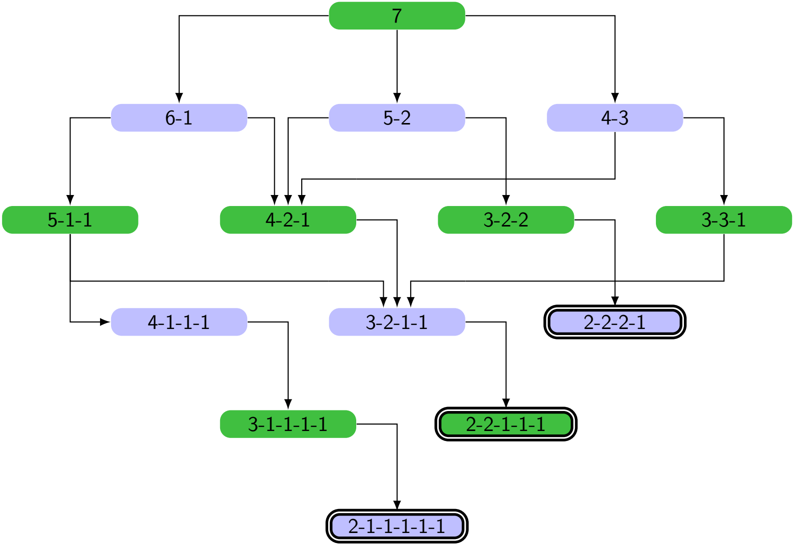
Minimax Procedure

- **Standard algorithm for two player games**
- **Opponents in the game are called MAX and MIN**
  - MAX $\begin{cases} \text{models “us” or “ our computer program”} \\ \text{tries to win the game or MAXimise advantge} \end{cases}$
  - MIN $\begin{cases} \text{models the opponent} \\ \text{tries to stop MAX winning} \end{cases}$
- **Assume MIN uses same information as MAX and always moves to a state that is worst for MAX**

Minimax Procedure

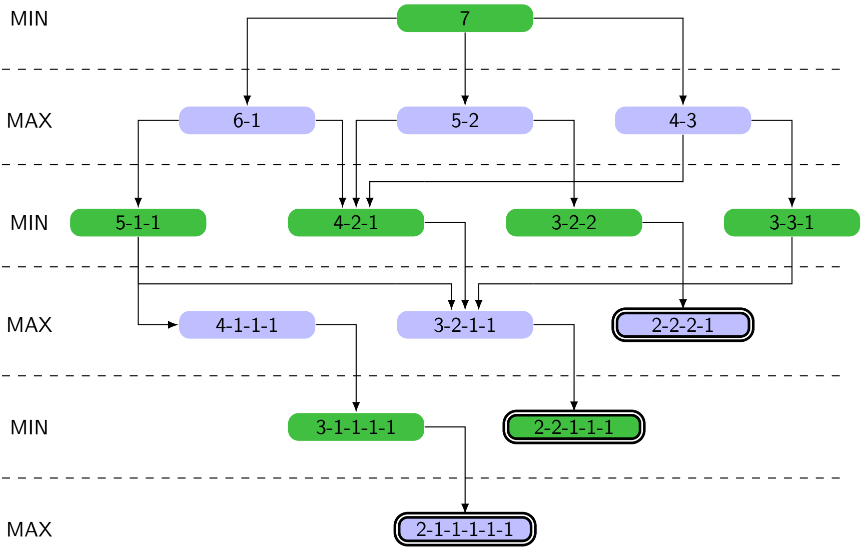
- **Minimax Rules**
  - Label each level in search space according to whose move it is
  - Label each leaf node with a value, as follows:
    - If it is a win for MAX, label it 1
    - If it is a win for MIN, label it 0
  - Propagate these values up the search graph through successive parent nodes according to:
    - If the parent state is a MAX node, give it the maximum value of its children
    - If the parent state is a MIN node, give it the minimum value of its children
- **Label indicates what is the best outcome MAX can obtain from that state**

State Space (MIN to move first)

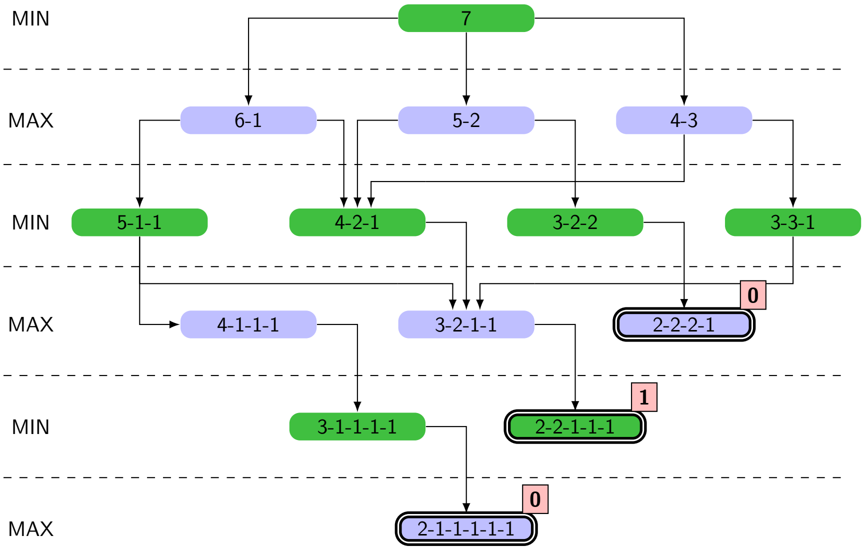


State Space (MIN to move first)

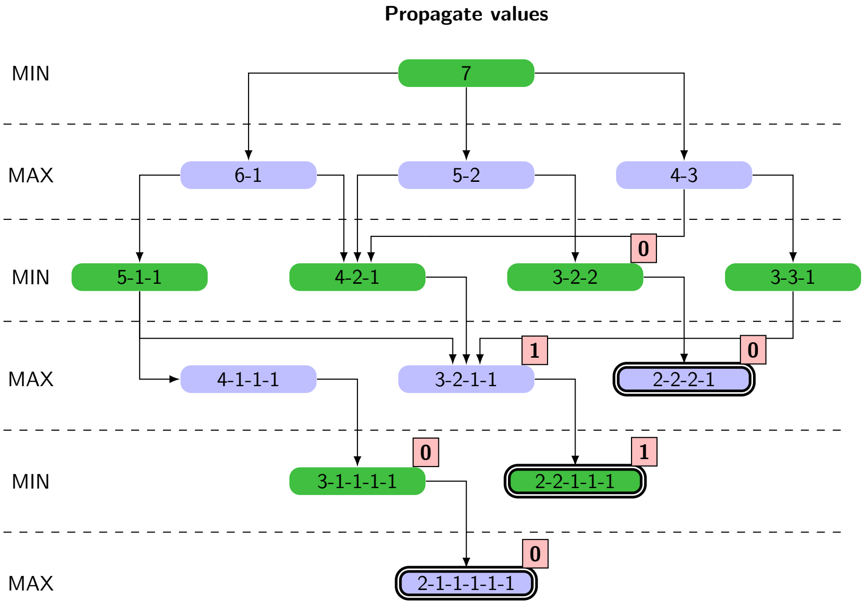
Label levels according to players' turns



Label leaf nodes as win (1) or loss (0) for MAX

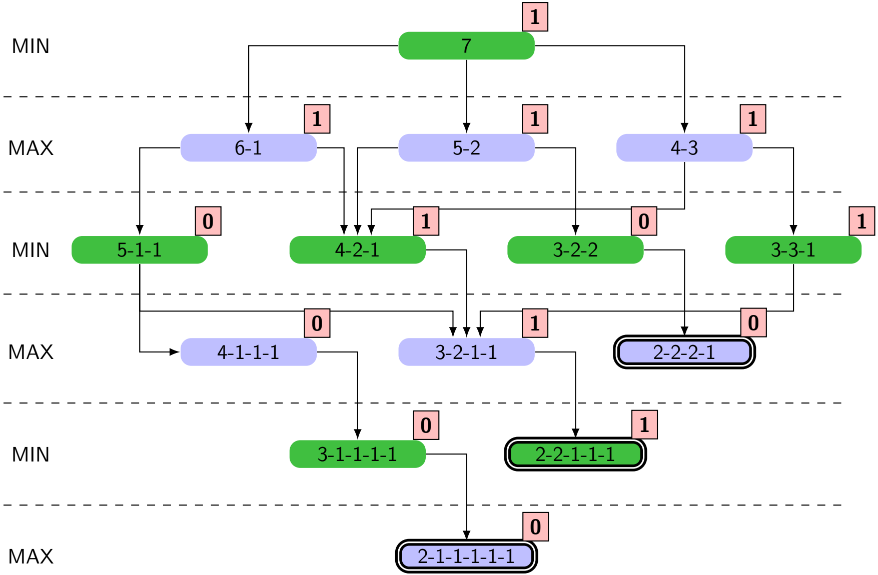


State Space (MIN to move first)

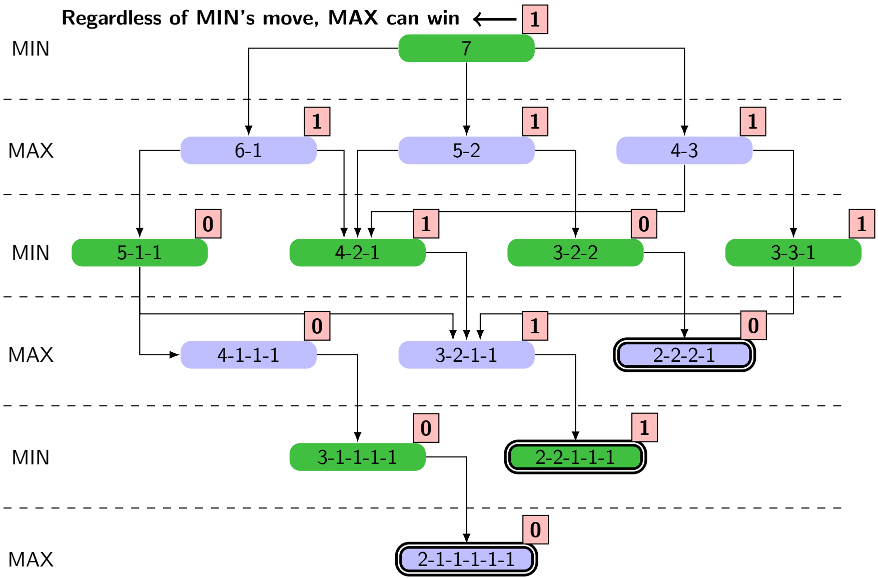


State Space (MIN to move first)

Propagate values

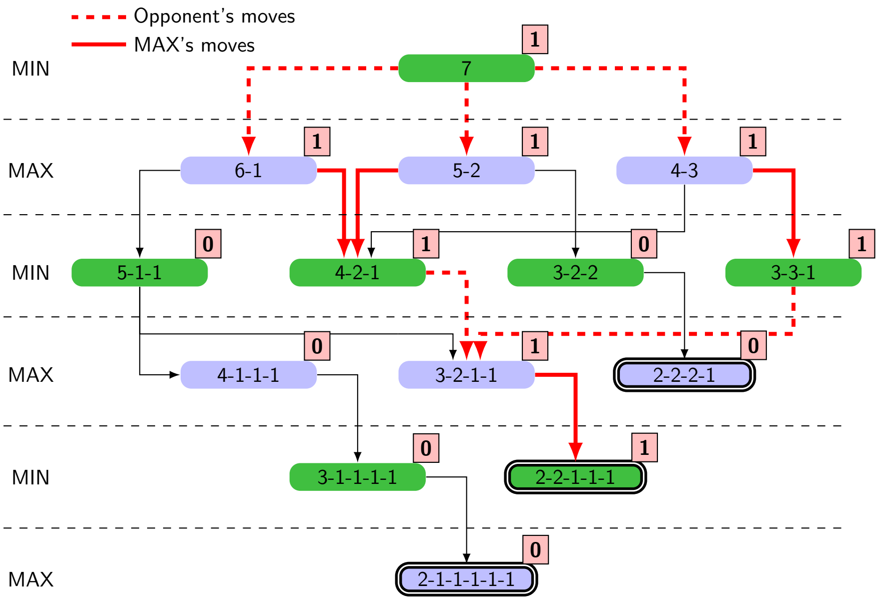


Forced Win



Regardless of MIN's move, MAX can win
1
MIN
7
MAX
6-1
1
5-2
1
4-3
1
MIN
5-1-1
0
4-2-1
1
3-2-2
0
3-3-1
1
MAX
4-1-1-1
0
3-2-1-1
1
2-2-2-1
0
MIN
3-1-1-1-1
0
2-2-1-1-1
1
MAX
2-1-1-1-1-1
0

Forced Win


Opponent's moves
MAX's moves
MIN
MAX
MIN
MAX
MIN
MAX
7
1
6-1
1
5-2
1
4-3
1
5-1-1
0
4-2-1
1
3-2-2
0
3-3-1
1
4-1-1-1
0
3-2-1-1
1
2-2-2-1
0
3-1-1-1-1
0
2-2-1-1-1
1
2-1-1-1-1-1
0

Minimax: limitations

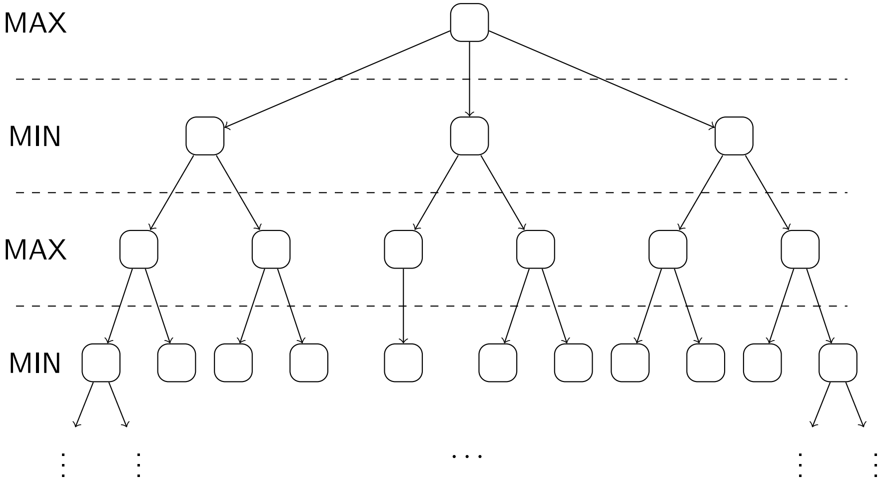
- **Seldom possible to expand search graph to leaf nodes**
- **Some examples (approximated)**
  - Tic-tac-toe: $\approx$ 1000 states
  - Checkers: $\approx 10^{20}$ states — about one trillion
  - Chess: $\approx 10^{47}$ states — about one octillion!
    - *Deep Blue*[1] able to compute 200 million positions/second – it would need more than 10^{25} million years to explore all states
- **Therefore, limit search to predefined number of levels**
  - As determined by available resources (time, memory)
  - Called *n-ply look ahead*, where *n* is the number of levels

[1] IBM's chess-playing computer that beat Gary Kasparov in 1997

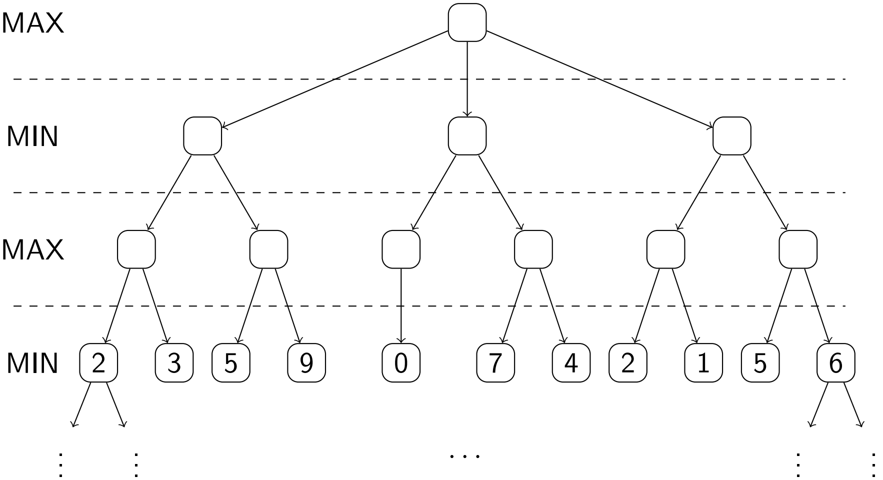
Minimax to Fixed Ply

- **WARNING: Leaf nodes of search are not leaf nodes of graph!**
  - Not possible to give values to reflect win/lose
  - Each leaf node is given a value according to *heuristic evaluation* function
  - Value that is propagated back is not an indication of win/lose, but simply an heuristic value of *best state* that can be reached in *n* moves from the start node
- **Method**
  - Select ply (i.e. number of levels)
  - Search to ply depth
  - Assign an (heuristic) evaluation to each leaf node
  - Propagate values back using the same rules as in Minimax

Example (hypothetical state space)

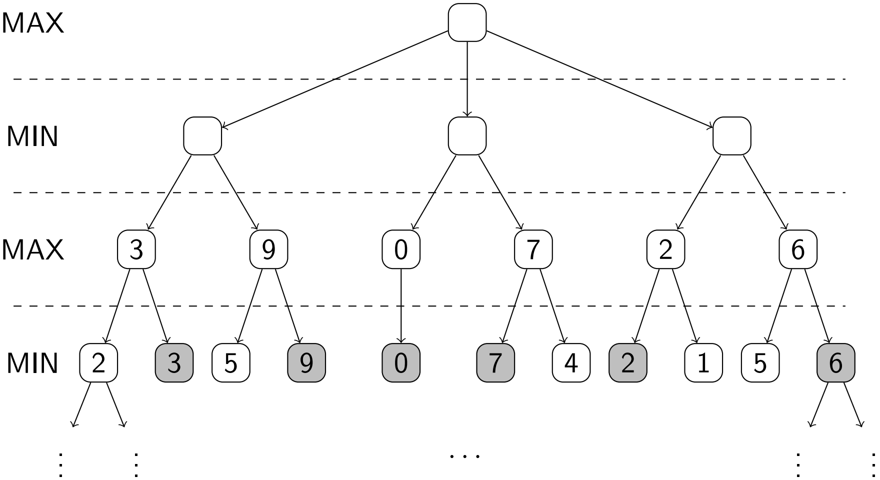


Apply heuristic evaluation to leaf nodes



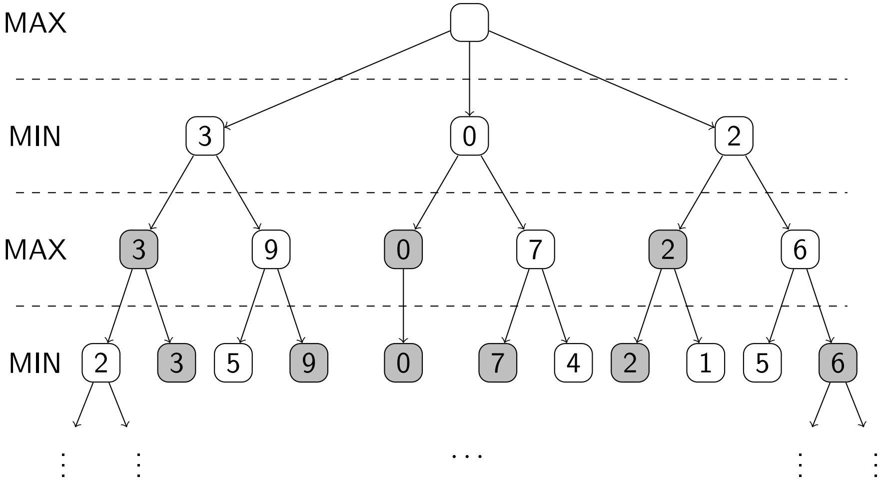
Example (hypothetical state space)

Propagate values up

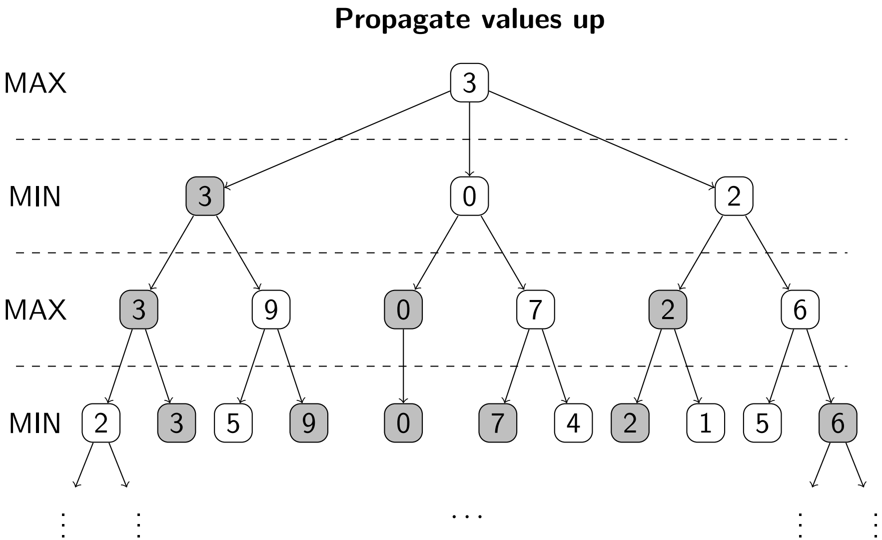


Example (hypothetical state space)

Propagate values up



Example (hypothetical state space)



Examples: Draughts Playing Program

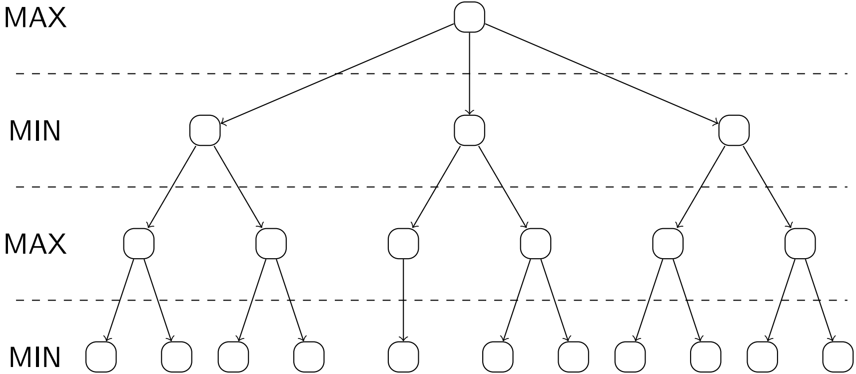
- **Developed by Arthur Samuel in early 1950s**
- **Evaluated board states with a weighted sum of several different heuristic measures**
  - $\sum w_i \cdot h_i$
  - Each heuristic represented features of the board
    - Piece advantage, location, control, sacrifice, inertia...
  - The coefficients were tuned weights that tried to model the importance of the feature
- **Program incorporated an element of learning**
  - After a loss, heuristics with high coefficients were reduced
  - After a win, the opposite was done
- **Program was able to play to high level**
  - Trained by playing against itself
  - Vulnerable to inconsistent play: widely varying strategies, foolish play

Alphabeta Search

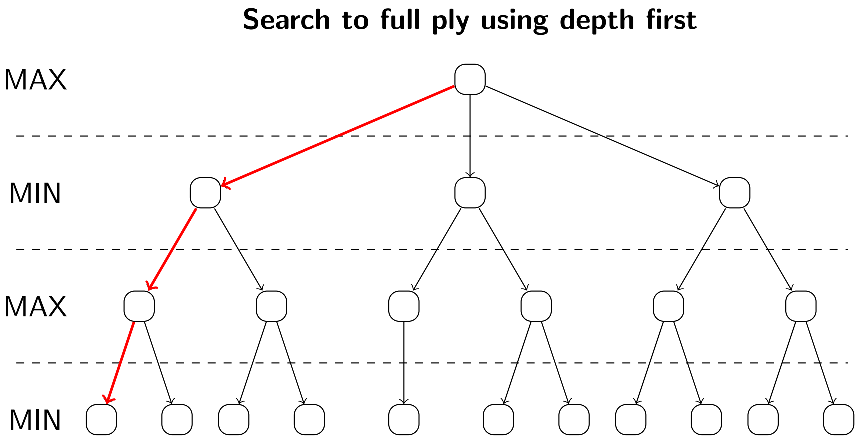
- **Exhaustive search explores all branches, even those that can be safely ignored**
- **Instead, use a class of algorithm called alphabeta pruning to improve search efficiency in two-player games**
- **In alphabeta search**
  - Don't do exhaustive search to fixed ply
  - Search depth first
  - Associate one of two values with each node
    - *Alpha* value, associated to MAX nodes, which can never decrease. It's the least MAX can get
    - *Beta* value, associated to MIN nodes, which can never increase
  - Prune search space
    - e.g., suppose MAX *alpha* value is 6, and a value propagated up to a child MIN node is less than 6: no point searching other children of MIN node.

- **Algorithm**
    - Search to full ply using depth first
    - Apply heuristic evaluation to all siblings at ply (assume these are MIN nodes)
    - Propagate value of siblings to parent using Minimax rules
    - Offer this value to **grandparent** MIN node as possible *beta* cutoff
    - Descend to other grandchildren
    - Terminate (prune) exploration of parent if any of their values is greater than or equal to the *beta* cutoff
    - Do the "same" for MAX nodes
    - Two rules for terminating search
        - Search stopped below any MIN node having a *beta* value less than or equal to *alpha* value of any of its MAX ancestors
        - Search stopped below any MAX node having an *alpha* value greater than or equal to *beta* value of any of its MIN ancestors

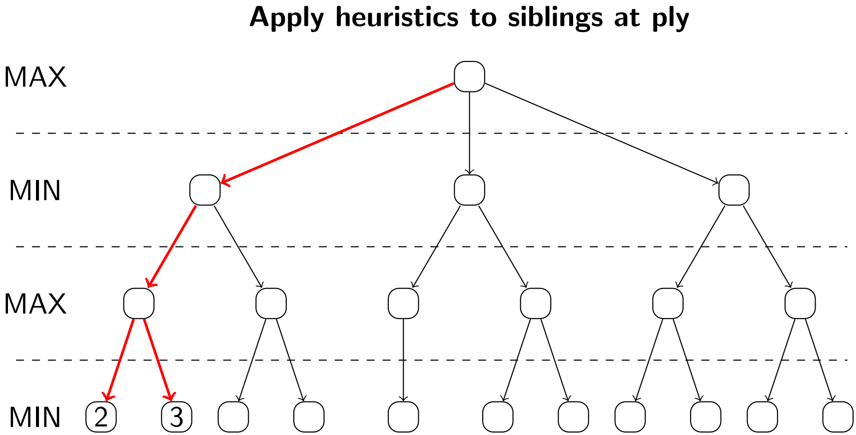
Example (hypothetical state space)



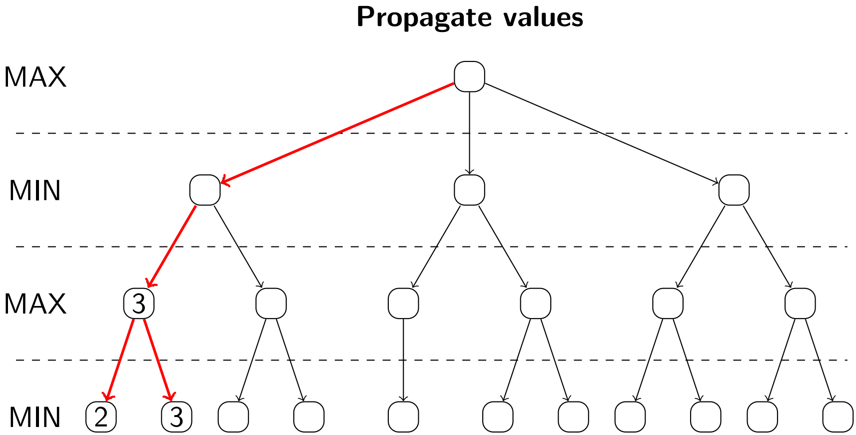
Example (hypothetical state space)



Example (hypothetical state space)

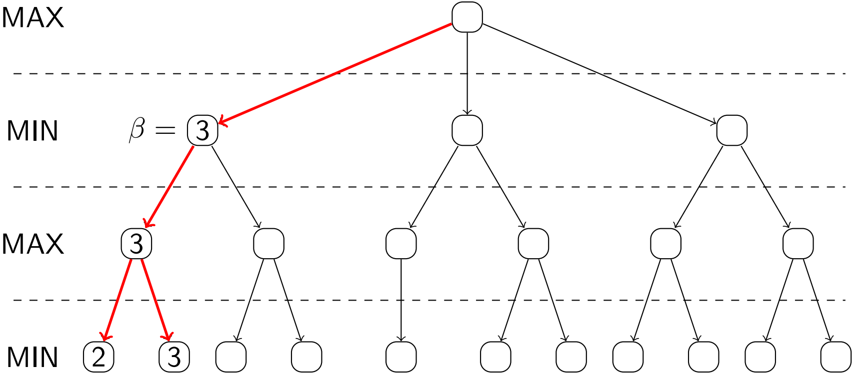


Example (hypothetical state space)

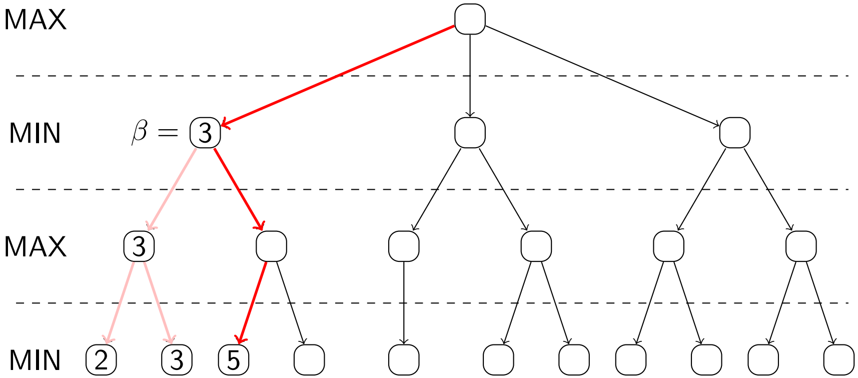


Example (hypothetical state space)

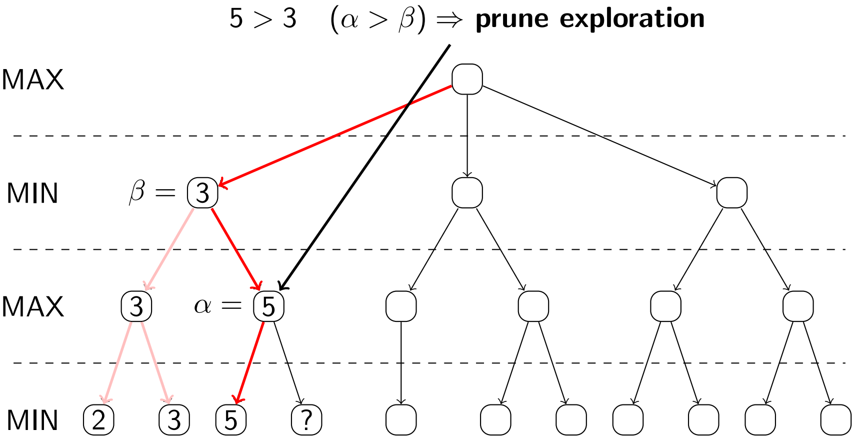
Offer value to grandparent MIN node as β cutoff



Descend to grandchildren

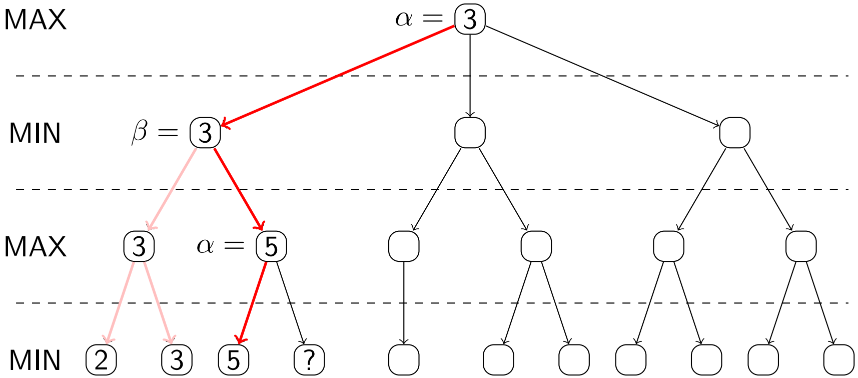


Example (hypothetical state space)



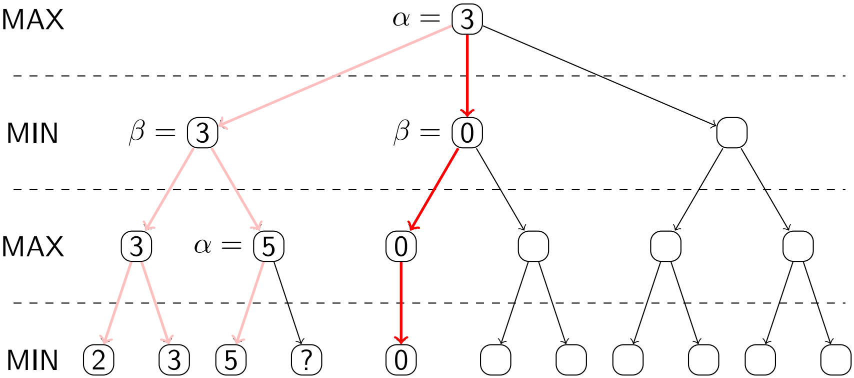
Example (hypothetical state space)

Propagate to MAX node

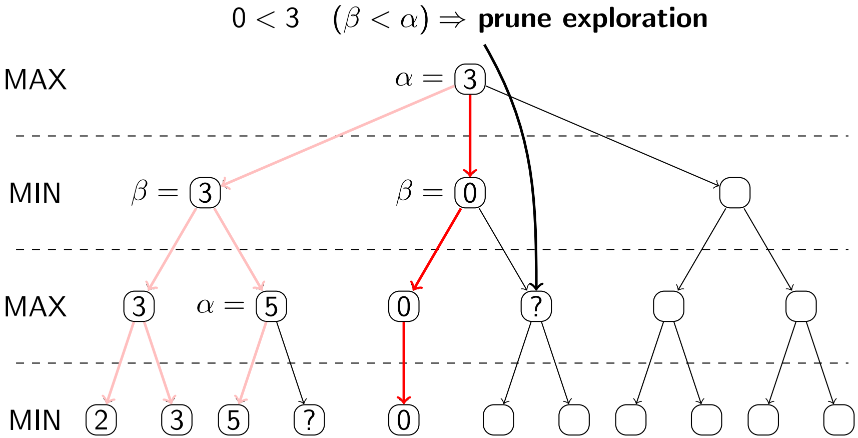


Example (hypothetical state space)

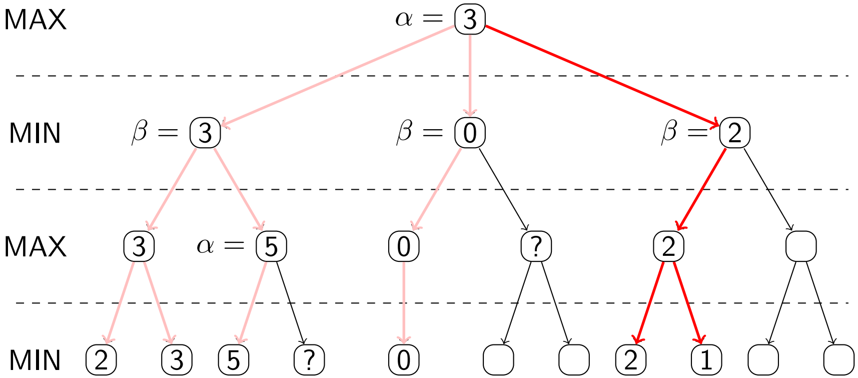
Continue with next sibling



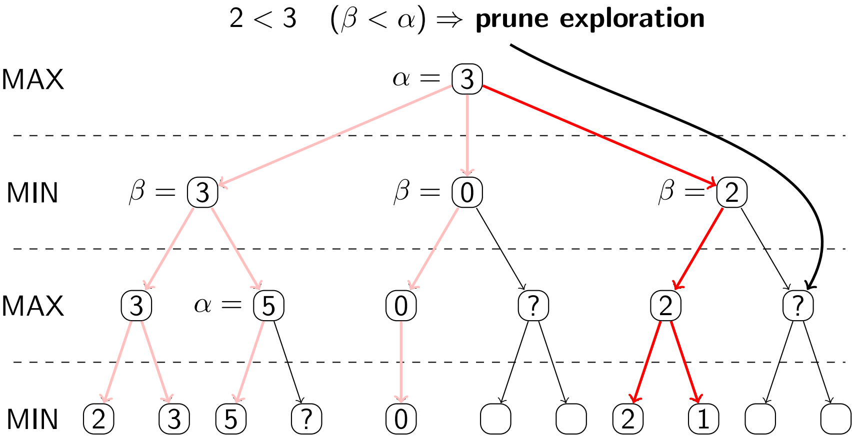
Example (hypothetical state space)



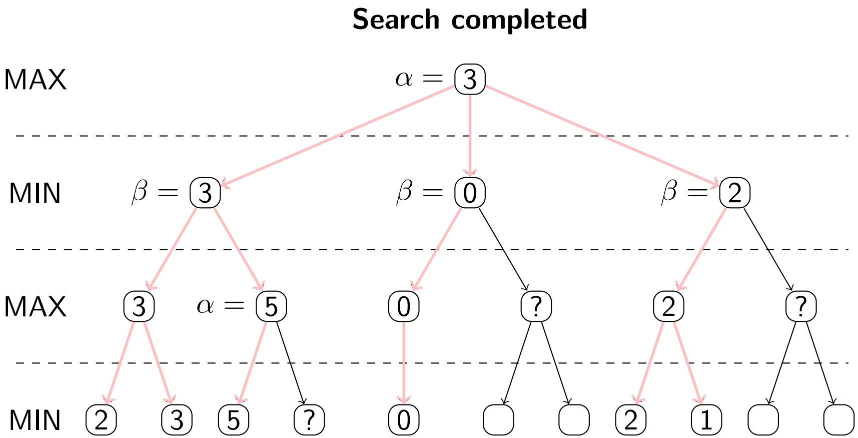
Continue with next sibling



Example (hypothetical state space)



Example (hypothetical state space)



Comments on Alphabeta

- **Concerning the example**
  - Resulting value propagated is identical to Minimax
  - Did not explore full search graph (visited 6 of 11 leaf nodes)
- **Order in which siblings are generated is important**
  - With a good ordering, alphabeta search can effectively double the depth of search space with fixed time/space restriction
  - With an unfortunate ordering, alphabeta searches no less (but no more) than Minimax – consider the same example from right to left

Summary

- **Search path in some applicatons (e.g. two player games) may not be completely under control**
- **In some circumstances, exhaustive search can enable an optimal path (forced win) to be identified**
  - Minimax procedures can be used in such cases
- **Usually exhaustive search is impractical, as the state space is too large**
  - Use Minimax to a fixed ply, plus heuristics
  - Use Alphabeta search, which prunes search space
  - Beware the horizon effect $\Rightarrow$ the Minimax of estimates
- **Effective applicaton of heursitics can lead to (more) efficient search of the state space**
  - Can make the difference between practical implementations and not