

IN 3130  
14th October, 2020  
(Slides by Stein Krogdahl)

---

– **First hour:**

- Ch. 23.5: Game trees and strategies for two-player games.

– **Second hour:** Guest lecture by Rune Djurhuus (grand master, chess):

- About programs for chess-playing and other games



## Ch. 23.5: Games, game trees and strategies

---

- We have looked at «one player games» (= search) and their decision trees, earlier in Ch 23 (from start to 23.4).
  - This is search for a goal node that **everybody agrees is «good»**.
- Then you can for instance use A\*-search for e.g.:
  - Solve the 15-puzzle from a given position.
  - Find the shortest path between nodes in a graph (better than plain Dijkstra)

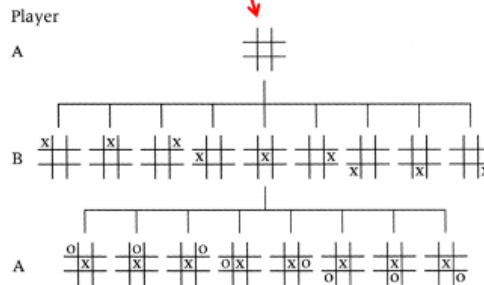
### **BUT:**

- When two players are playing *against* each other, things get very different. What is **good** for one player is **bad** for the other.
  - The tree of possible plays is often enormous. For chess it is estimated to have ca  $10^{100}$  nodes, and can therefore never (?) be searched exhaustively!
- We look at “zero-sum” games. This roughly means:
  - If, during a move, the “chances to win” is increased for one of the players, then it is decreased by the corresponding amount for the other.

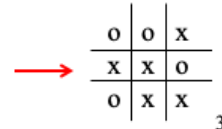
## Example: Game trees and Tic-Tac-Toe

- The board has 3 x 3 squares.
- The game: Repeat the following moves
  - *Player A* chooses an unused square and writes 'x' in it,
  - *Player B* does the same, but writes 'o'.
- Player A (always) starts
- When a player has three-in-a-row, he/she has won.
- The game stops when A or B wins, or when all squares are filled (maybe with a “draw” = neither A nor B has three-in-a-row)

The start/root node of the game tree for Tic-Tac-Toe.



A final situation without a winner.



## Number of nodes in a fully expanded Tic-Tac-Toe tree

1 node

9 nodes

$9 \times 8 = 72$  nodes

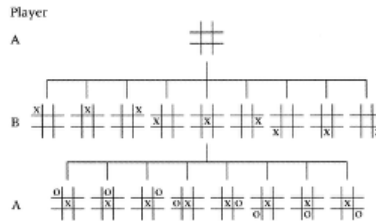
$9 \times 8 \times 7 = 504$  nodes

$9 \times 8 \times 7 \times 6 = 3024$  nodes

$9 \times 8 \times 7 \times 6 \times 5 = 15120$  nodes

.....

$9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 9!$  ("factorial") = 362 880 nodes



**Comment:** By searching **depth-first** in this tree, you never need to store more than 9 nodes, but it will take some time to go through all 362 880 nodes (and for "interesting games" there are usually *a lot more!*).

The same situation may occur many places in the tree  
We may represent each game situation by only one node

1 node

9 nodes

$9 \times 8 = 72$  nodes

$9 \times 8 \times 7 = 504$  nodes

$9 \times 8 \times 7 \times 6 = 3024$  nodes

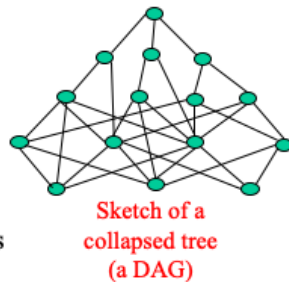
$9 \times 8 \times 7 \times 6 \times 5 = 15120$  nodes

$9 \times 8 \times 7 \times 6 \times 5 \times 4 = 60480$  nodes

.....

$9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 362\,880$  nodes

This usually requires a lot of memory!



Sketch of a  
collapsed tree  
(a DAG)

As before

1 node

9 nodes

72 **different** nodes

252 **different** nodes

756 **different** nodes

1260 **different** nodes

1680 **different** nodes

.....

126 **different** nodes =  $\binom{9}{4}$

$$\binom{9}{4} = (9 \times 8 \times 7 \times 6) / (1 \times 2 \times 3 \times 4) = 126$$

Fewer than before

In some games, e.g. Tic-Tac-Toe, you can gain a lot by recognizing equal nodes, and not repeat the analysis for these. In Tic-Tac-Toe we then never need more than 1680 nodes during breath first search. In Chess this is very important!

5

Vi kan ofte komme til samme spill-situasjon på flere måter.

Flere trekk-sekvenser kan føre oss til samme situasjon. I tre-på-rad er det jo bare rekkefølgen av trekk som har noe å si.

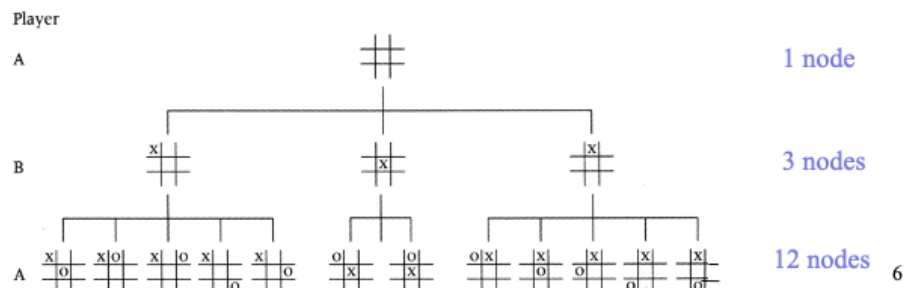
$n!$

-----

$k! (n-k)!$

## Representing symmetric situations by the same node

- One can also gain a lot by looking at symmetries:
  - Two situations are *symmetric* if the rest of the game from these two situations will also be symmetric according to the rules of the game.
  - Represent positions that are symmetries of each other by the same node.
  - Tic-Tac-Toe: Symmetric solutions will always be at the same depth, but this is *not* generally the case!
  - In e.g. chess there are fewer symmetries to utilize.
- Using this will often reduce the needs for memory/time further!



## The “value” of a position, and zero-sum games

- During a game, we will always store:
  - A number (value) characterizing how good the situation is *for player A*.
    - High values are good for A, and low values are bad.
    - Thus all nodes of a game-tree have a value (seen from A)
  - If we want to see the game from B’s point of view, we usually negate the values.
- We want a “strategy for A”.
  - That is: A rule telling A what to do in all possible “A-situations” (those where it is A’s turn to make a move).
  - We will, for a given position, look for a strategy so that A will win.
    - **But note:** Such a strategy *will often not exist!*

7

## Fully analyzable games

---

- “Fully analyzable games” means: The full tree can be traversed and analyzed
  - Then there will be three possible values for each A-situation S (usually represented as +1, -1 or 0)
    1. A has a strategy so that it will win whatever B does, if A follows that strategy from S (score: +1 for A)
    2. Whatever A does from S, B has a winning strategy from the new situation (score: -1 for A).
    3. If A and B both play perfectly, it will end in a tie, or the game will go on for ever (score: 0 for both)
      - Situation 3 can only occur for some games.
      - E.g.: The game Tic-Tac-Toe ends in a tie if both players play as good as possible.

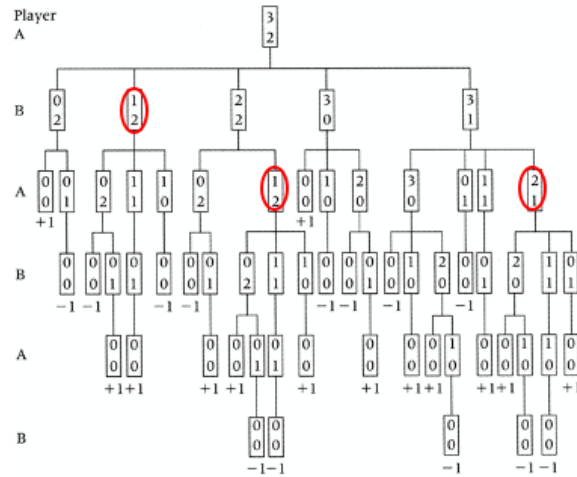
## Another example: The game Nim

The game Nim:

- We start with two (or more?) piles of sticks.
- Number of sticks:  $m$  and  $n$ .
- One player can take any number of sticks from one pile, but have to take at least 1.
- The player taking the last stick has lost.

- Nim will never end in a tie.
- With  $m=3$  and  $n=2$ , the full game tree is shown to the right.
- The value seen from A is indicated for the final situations (leaf nodes).
- Next problem: What is the value of the rest of the nodes?

Here  $m=3$  and  $n=2$



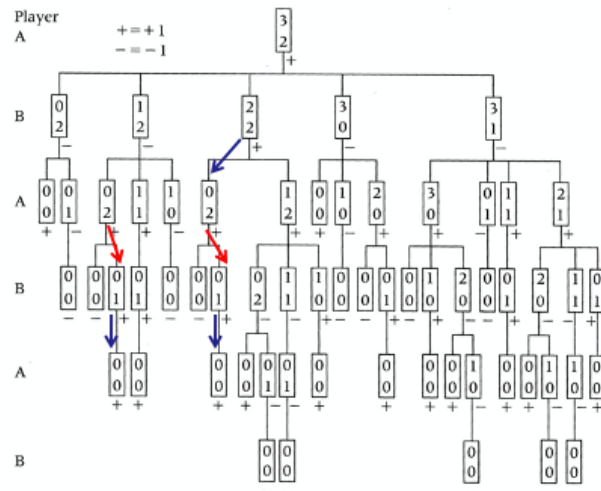
NB: We could reduce the number of separate nodes by recognizing symmetries and equivalent nodes (see e.g. red circles above)

## How can we find a strategy so that A wins? Or prove that no such strategy exists!

- A wants to find an optimal move from a given position.
- We must assume that also B will do optimal moves seen from its point of view.
- Thus B will move to the subnode with *smallest* value (since +1 and -1 are as seen from A).

### Min-Max Strategy:

- To compute the value of a node, we have to know the values of all the subnodes.
- This can be done by a *depth first search*, computing node values during the withdrawal (postfix).



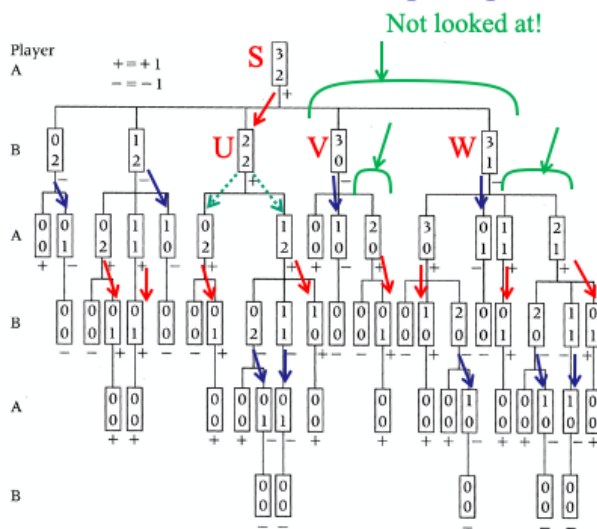
Values for A-nodes: If possible, move to a node with value +1 (and mark current node with +1). Otherwise make a random move.

Values for B-nodes: If possible, move to a node with value -1. Otherwise make a random move. 10

Min-max-strategien

# The Min-Max-Algorithm in action

## With simple alpha-beta cutoff



- Previous slide: The search is done by a depth first traversal of the game tree, computing values on withdrawal (postfix)
- The result of this is given in the figure to the left as + and -.

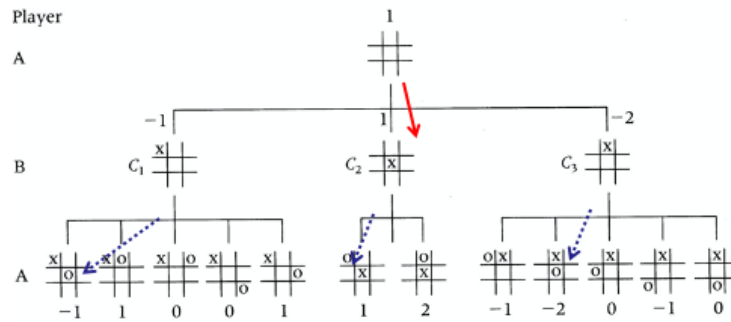
### Possible optimization:

- From the start-position S, assume that A has looked at three of its subtrees (from the left). A has then found a winning node U (marked +1). Then the value of V and W does not matter.
- This is a simple version of alpha-beta cutoff (pruning)

11

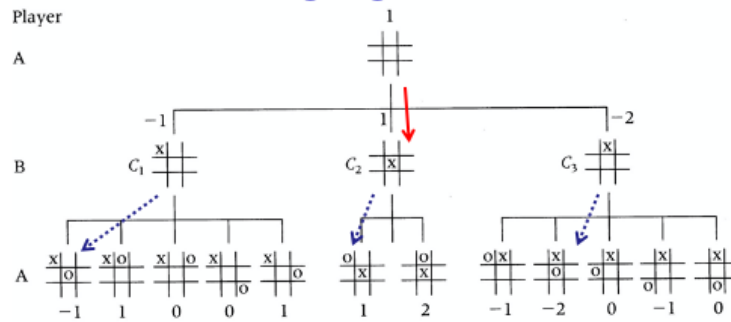
Minimax-algoritmen

## What if the game tree is too large to traverse?



- One then usually searches to a certain depth, and then estimate (with some heuristic function) how good the situation is for A at the nodes at that depth. We then usually use **other values** than only: -1, 0 and +1.
- In the figure above we go to depth 2.
- The heuristic function above is: **Number of «winning lines for A» minus the same number for B (this is given above for each leaf node).**
- A **“winning line”** for A is a column, row or diagonal where B has not filled any of the three positions (so that A can still hope to fill them all, and win).
- The best move for A from the start position is therefore (according to this heuristic) to go to C<sub>2</sub>.

## Too large game trees



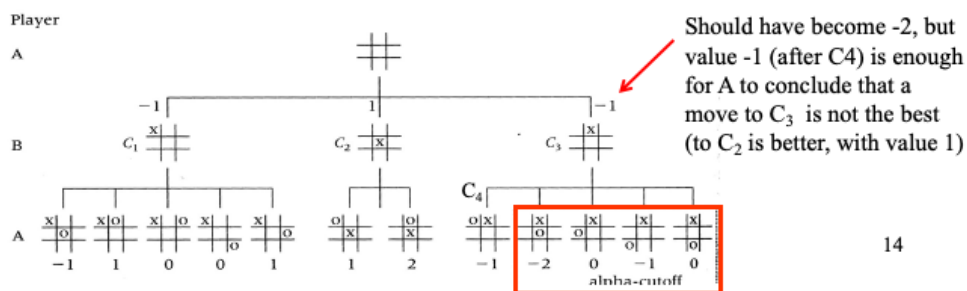
- However, this heuristic is not good later on in the game. It does not take into account that winning is better than any heuristic. We therefore, in addition, give winning nodes **the value  $+\infty$**  (but no such node occur above).
- This will give quite a good strategy. But, as said above: Tic-Tac-Toe will **end in a tie** if both players play perfectly.
- We have to add that the tie-situation (e.g. the one to the right) **gets the value 0**. Thus, if we fully analyze the game, the value of the root node will be 0.
- **NOTE: The difficult choice for a game-programmer is between searching very deep or using a good, but time consuming, heuristic function!**

o	o	x
x	x	o
o	x	x

# General alpha-beta cutoff (pruning)

Intuitively Alpha-beta-cutoff goes as follows (assuming it is A's move):

- A will consider all the possible moves from the current situation, one after the other...
- After a while, A has noted that the best move seen so far is a move in which A can obtain the value  $u$  (after  $C_1$  and  $C_2$ ,  $u = 1$ )
- A looks at the next potential move, which would lead to situation  $C_3$ , and then looks at the subnodes of  $C_3$ . It soon observes that B has a very good move ( $C_4$ ) giving value  $v = -1$ . Thus the value of  $C_3$  cannot be better (for A) than  $-1$  as B will minimize at  $C_3$ . This is true independent of what value the other subtrees of  $C_3$  gives.
- As  $v < u$ , player A has no interest in looking for even better moves for B from situation  $C_3$ . A already knows that it has a better move than to  $C_3$ , which is to  $C_2$ .



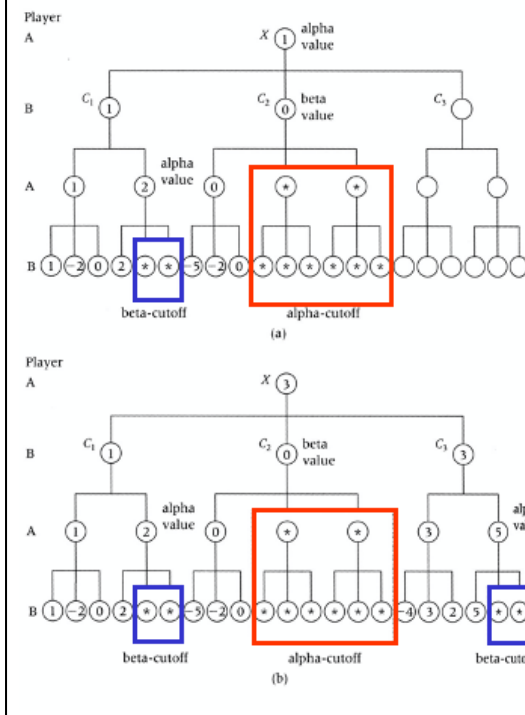
14

Her er vi over på alfa-beta-avskjæring

Intuisjon

Vi trenger ikke beregne me

## Examples showing alpha-beta cutoff



- When A considers the next move:
  - Cutoffs from A-situations is called alpha-cutoffs.
  - Corresponding cutoffs from B-situations are called beta-cutoffs..
- The figures to the left shows alpha- and beta-cutoffs at different stages of a DF-search of a game tree.
- When implementing alpha-beta-cutoffs during a DF-search, it is usual to switch viewpoints between the levels.
  - Then we can always *maximize* the value.
  - But we have to negate all values for each new level.
- Such an implementation is given at the next slide.

15

- 1: Alpha (2) > Beta (1) – beta-cutoff
- 2: Beta (0) < Alpha (0) – alpha-cutoff
- 3: Alpha (5) > Beta (3) – beta-cutoff

## Alpha-beta-search (negating the values for each level)

```

real function ABNodeValue (
  X,           // The node we compute alpha/beta value for. Children: C[1],C[2]... C[k]
  numLev,     // Number of levels left
  parentVal) // The alpha/beta-value from the parent node (-LB from the parent)
// Returned value: The final alpha/beta-value for the node X
{
  real LB; // Will hold current Lower Bound for the alpha/beta value of node X

  if <X is a terminal node> or numLev = 0 then {
    return <An estimate of the quality of the situation (the heuristic)>;
  } else {
    LB := -ABNodeValue(C[1], NumLev-1, ∞); // Recursive call
    for i := 2 to k do {
      if LB >= parentVal then {
        return LB; // Cutoff, no further calculation
      } else {
        LB := max(LB, -ABNodeValue(C[i], Numlev-1, -LB) ); //Recur. call
      }
    }
  }
  return LB;
}

```

Start the recursive call to calculate value for the (current) rootnode (down to depth 10) by calling  
 ABNodeValue(rootnode, 10, -∞) // This "-" is missing in the textbook

## Misprints in the textbook (B&P)

---

- There are some simple misprints in the program at page 741 in the textbook (may be corrected in some editions):
  - "AB" is missing in the name of the procedure in the recursive call.
  - A right parenthesis is missing at the end of the line where **max** is called.
  - A minus ("-") is missing in the arguments of the initial call
- These errors are corrected on the previous slide!