CSC384
Game Tree Search
Part 2

Bahar Aameri & Sonya Allin

Winter 2020

Credits

These slides are drawn from or inspired by a multitude of sources including :

Faheim Bacchus
Sheila McIlraith
Andrew Moore
Hojjat Ghaderi
Craig Boutillier
Jurgen Strum
Shaul Markovitch

Alpha-Beta Pruning

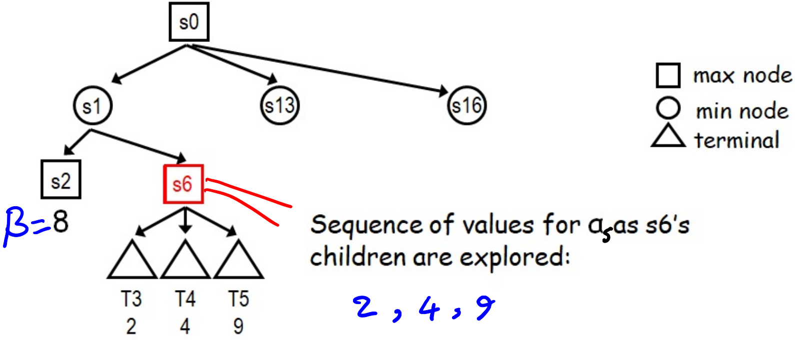
- There are ways to **avoid** examining the **entire** tree to make correct Minimax decision.

- When using depth-first search of a game tree:

  - After generating value for only **some of s's children** we can prove that we **never reach** s in a Minimax strategy.

  - So we need NOT generate or evaluate any further children of s.
    These other children can be **pruned**.

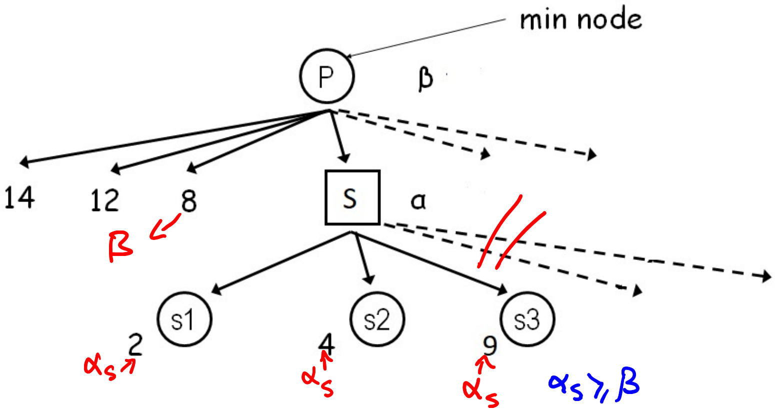
At a Max node s:
α_s: The **highest value of s's children** examined so far (changes as children of s are examined).
β: The **best option for MIN** (i.e., lowest value) found so far (fixed as children of s are examined);



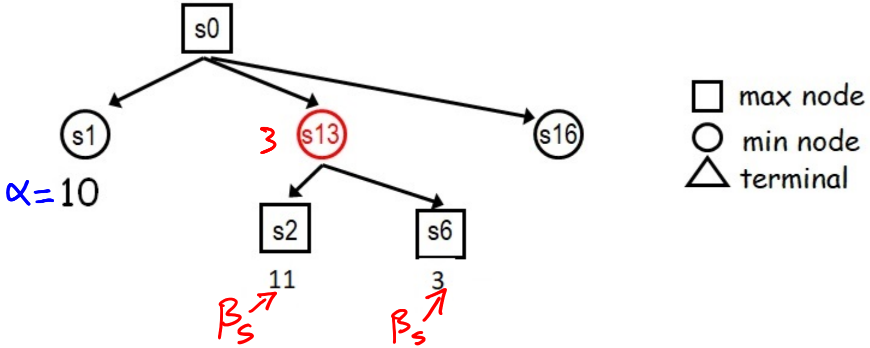
Sequence of values for α_s as s6's children are explored:

2, 4, 9

If α_s becomes **greater than or equal** to β, we can stop expanding the children of s:
Min will **never choose** to move from s's parent to s since it would choose one of s's **lower valued siblings**.

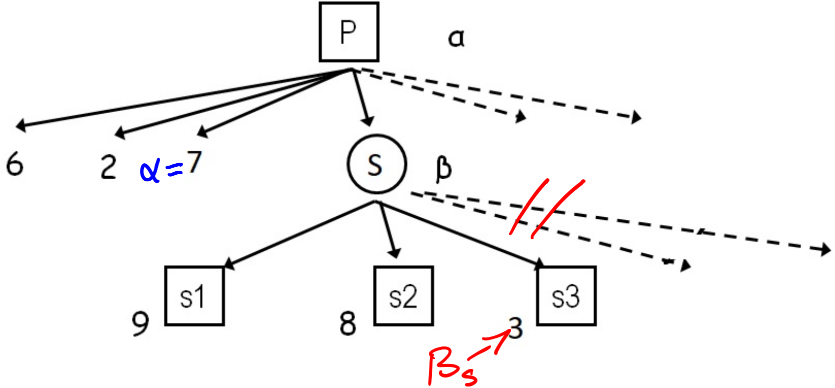


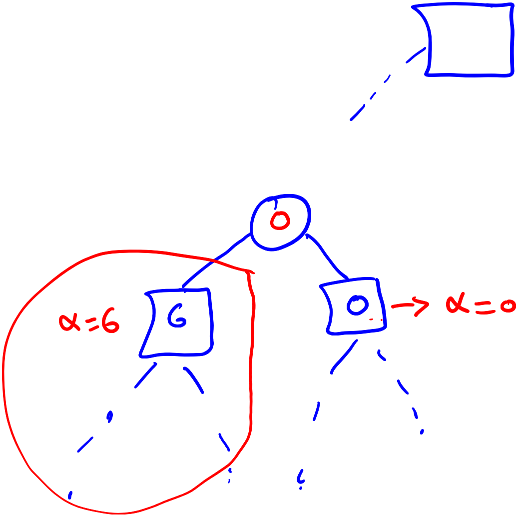
At a Min node s:
α: The **best option for MAX** (i.e., highest value) found so far (fixed as children of s are examined).
β_s: The **lowest value of s's children** examined so far (changes as children of s are examined);



Cutting Min Nodes (Beta Cuts)

- If α becomes **greater than or equal** to β_s, we can stop expanding the children of s: Max will **never choose** to move from s's parent to s since it would choose one of s's **higher value siblings**.





0
α=6
6
0
α=0

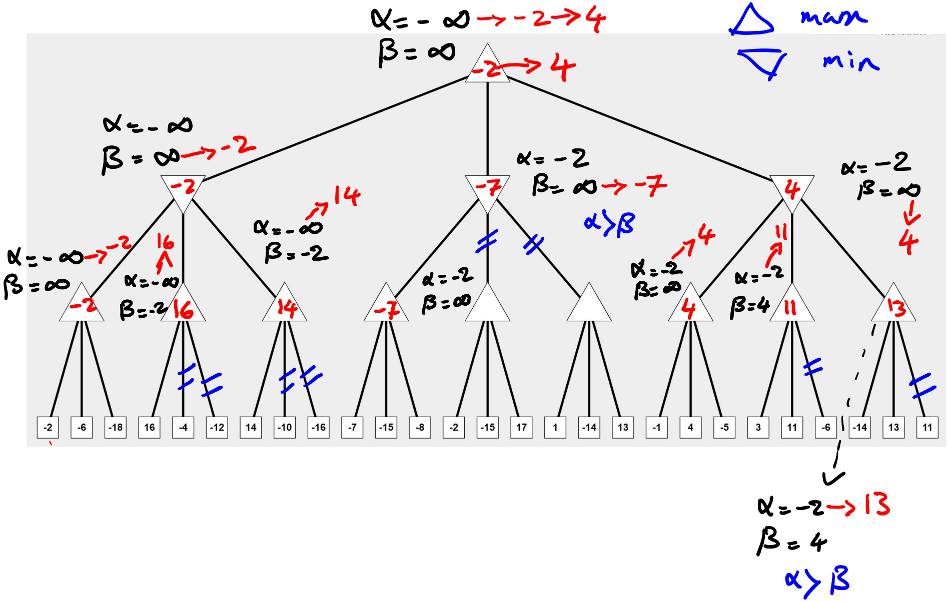
α: **Best** already explored option along the **path to the root** for **MAX**.
β: **Best** already explored option along the **path to the root** for **MIN**.

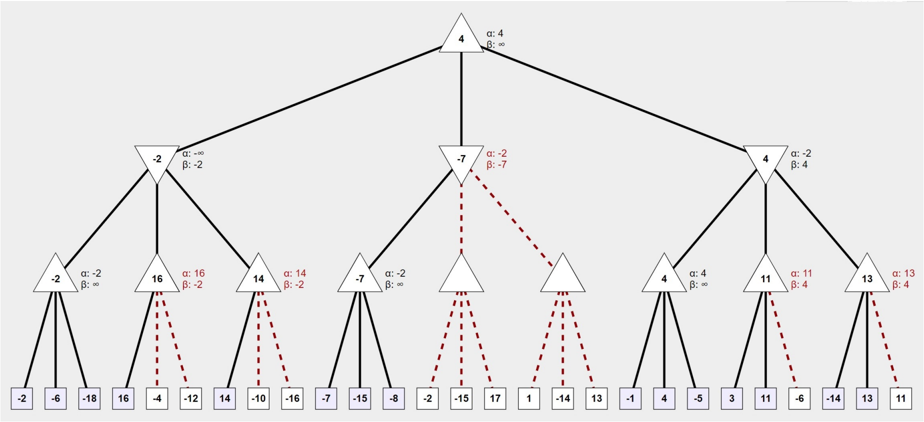
Alpha-Beta Pruning:

- Set initial values: $\alpha = -\infty$ and $\beta = \infty$
- While backing the utility values up the tree, identify α and β for each node.
- At every node s, if $\alpha \geq \beta$, **prune** (remaining) children of s.

α**-cuts**: Pruning of **MAX** nodes.
β**-cuts**: Pruning of **MIN** nodes.

Alpha-Beta Pruning – Example




4
α: 4
β: ∞
-2
α: -∞
β: -2
-7
α: -2
β: -7
4
α: -2
β: 4
-2
α: -2
β: ∞
16
α: 16
β: -2
14
α: 14
β: -2
-7
α: -2
β: ∞
4
α: 4
β: ∞
11
α: 11
β: 4
13
α: 13
β: 4
-2 -6 -18 16 -4 -12 14 -10 -16 -7 -15 -8 -2 -15 17 1 -14 13 -1 4 -5 3 11 -6 -14 13 11

Alpha-Beta Pruning Implementation

```
def AlphaBeta(s,Player,alpha,beta):
// Return Utility of state s given that Player is MIN or MAX
If s is TERMINAL
    Return U(s)   # Return terminal states utility
ChildList = s.Successors(Player)
If Player == MAX
    ut_val = -infinity
    for c in ChildList
         ut_val = max(ut_val, AlphaBeta(c,MIN,alpha,beta))
         If alpha < ut_val
            alpha = ut_val
            If beta <= alpha: break
    return ut_val
 Else # Player is MIN
    ut_val = infinity
    for c in ChildList
         ut_val = min(ut_val, AlphaBeta(c,MAX,alpha,beta))
         If beta > ut_val
            beta = ut_val
            If beta <= alpha: break
    return ut_val
```

Ordering of Moves

- For MIN nodes the best pruning occurs if the best move for MIN (child yielding lowest value) is **explored first**.

- For MAX nodes the best pruning occurs if the best move for MAX (child yielding highest value) is **explored first**.

- We don't know which child has highest or lowest value without doing all of the work! But we can use **heuristics** to estimate the value, and then choose the child with highest (lowest) heuristic value.

Effectiveness of Alpha-Beta Pruning

- With no pruning, $\mathcal{O}(b^d)$ nodes are explored, which makes the run time of a search with pruning the same as plain Minimax.

  If, however, the move **ordering** for the search is **optimal** (meaning the best moves are searched first), the number of nodes we need to search using alpha beta pruning is $\mathcal{O}(b^{d/2})$.

- In Deep Blue, they found that **alpha-beta pruning** meant the average branching factor at each node was about **6 instead of 35**.

Practical Matters

- **Real** games are **too large** to enumerate tree.
  Example:
  - Chess branching factor is roughly 35.
  - Depth 10 tree: 2,700,000,000,000,000 nodes
  - Even Alpha-Beta pruning won't help here!
- We must **limit depth** of search tree:
  - Must stop the search at some non-terminal nodes.
  - We must make heuristic estimates about the values of the non-terminal positions where we terminate the search.
  - These heuristics are often called evaluation functions.
  - Evaluation functions are often learned.

Examples of Heuristics in Games:

- **Tic Tac Toe**:
  $h(n) =$ [# of 3 lengths that are left open for player A] - [# of 3 lengths that are left open for player B].

- **Chess**: Alan Turing's function
  $h(n) = A(n)/B(n)$ where $A(n)$ is the sum of the point value for player A's pieces and $B(n)$ is the sum for player B.

- Many evaluation functions can be specified as a **weighted sum of features**:
  $h(n) = w_1 \times feature_1(n) + w_2 \times feature_2(n) + ...w_i \times feature_i(n)$.
  The weights can be **learnt**.

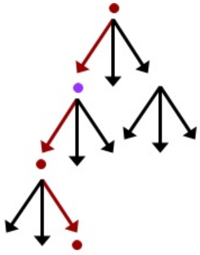
An Aside on Large Search Problems

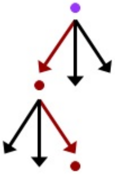
- Inability to expand tree to terminal nodes is relevant even in **standard search**: Often we can't expect the search to reach a goal by expanding full frontier.

- **Real-time (or online) Search:** We limit our look-ahead, and make moves before we actually know the true path to the goal.

- In real-time search, we use the heuristic function not just to guide our search, but also to commit to moves we actually make.
  In general, guarantees of optimality are lost, but we reduce computational/memory expense dramatically.

Real-time Search Graphically





1. We run our favorite search algorithm until we are **forced to make a move** or **run out of memory**.
   Note: no leaves are goals yet.

2. We use **evaluation function** $f(n)$ to decide which path **looks best** (let's say it is the red one).

3. We take the first step along the **best path** (red), by actually making that move.

4. We **restart search** at the **node we reach** by making that move.