Yapay Zeka

Güz 2023 - Hafta 5
(BİL413 - IVB511)

Dr. Arda Akdemir

Bugünün planı

- Jupyter/environment setup
- Graph visualization in python
  - (tekrar) DFS ile TicTacToe hamlelerine bakma
- TicTacToe-AI
  - Implement TicTacToe engine
  - (tekrar) minimax
  - Implement TicTacToe-AI
  - Improvements?

Birkaç hatırlatma

Notlandırma

- **Lisans: (vizeye girebilmek için %70 ve üstü devam zorunluluğu var)**

  - 15% dönem içi ödevler

  - 5% devamlılık

  - 20% ~~vize~~ proje

  - 60% final yüz yüze olacak ve lisansla aynı

- **Lisans üstü:**

  - 15% ödevler

  - 25% ~~vize~~ proje

  - 60% final yüz yüze olacak ve lisansla aynı

  (Lisansüstü’nde çalışan arkadaşlar da olduğu için devamlılık notlandırmaya dahil değil)

Jupyter notebook session

Clone me

(Gecen hafta ile ayni)
Oyunlara Giriş
(Chapter 5. Adversarial Search)

Typical assumptions

- Two agents whose actions alternate
- Utility values for each agent are the opposite of the other
  - creates the adversarial situation
- Fully observable environments
- In game theory terms:
  - "Deterministic, turn-taking, zero-sum games of perfect information"
- Can generalize to stochastic games, multiple players, non zero-sum, etc

Search versus Games

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- Games – adversary
  - Solution is strategy (strategy specifies move for every possible opponent reply).
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate "goodness" of game position
  - Examples: chess, checkers, Othello, backgammon

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

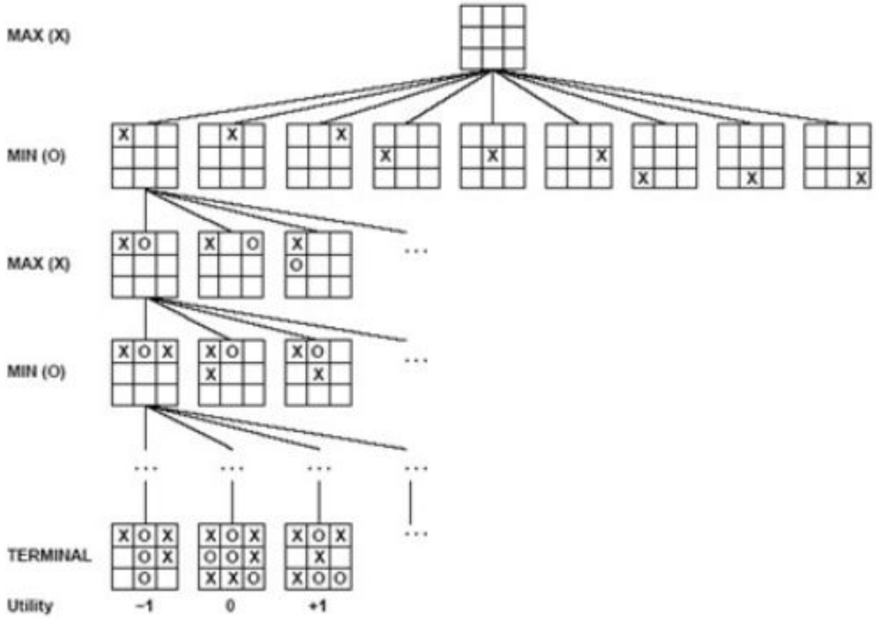
Game Setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  - Winner gets award, loser gets penalty.
- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move, state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states. E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess
- MAX uses search tree to determine next move.

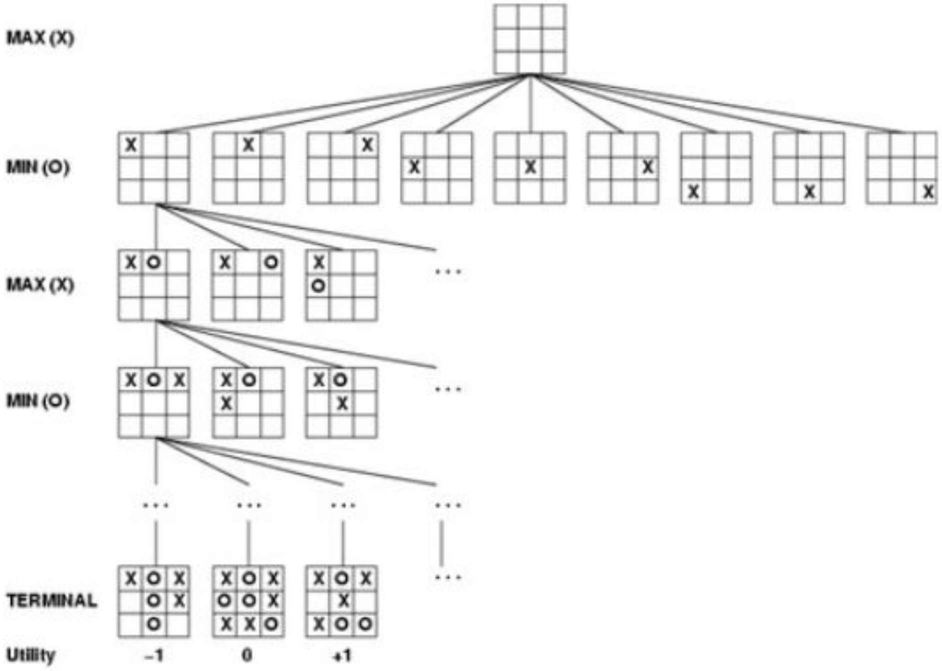
Size of search trees

- b = branching factor
- d = number of moves by both players
- Search tree is $O(b^d)$
- Chess
  - b ~ 35
  - d ~100
    - search tree is ~ 10^{154} (!!)
    - completely impractical to search this
- Game-playing emphasizes being able to make optimal decisions in a finite amount of time
  - Somewhat realistic as a model of a real-world agent
  - Even if games themselves are artificial

Partial Game Tree for Tic-Tac-Toe



Game tree (2-player, deterministic, turns)



How do we search this tree to find the optimal move?

Minimax strategy

- Find the optimal *strategy* for MAX assuming an infallible MIN opponent
  - Need to compute this all the down the tree
- Assumption: Both players play optimally!
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

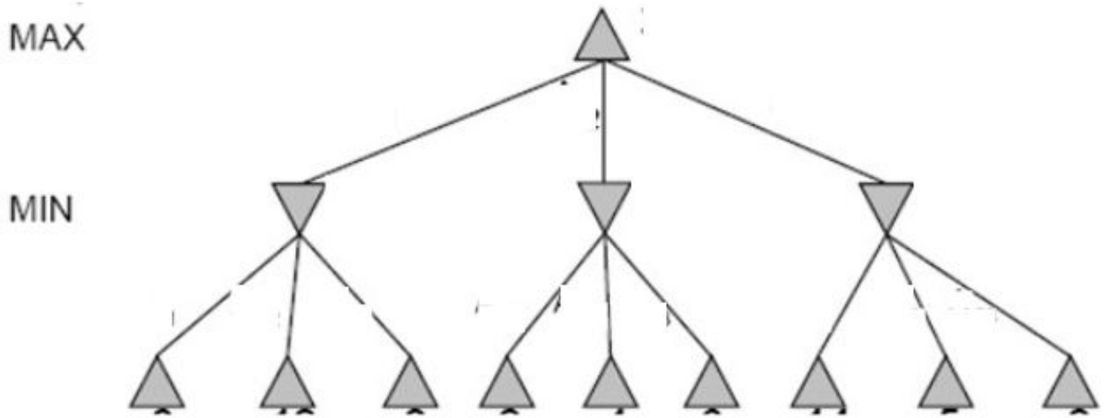
MINIMAX-VALUE(*n*)=

UTILITY(*n*) **If *n* is a terminal**

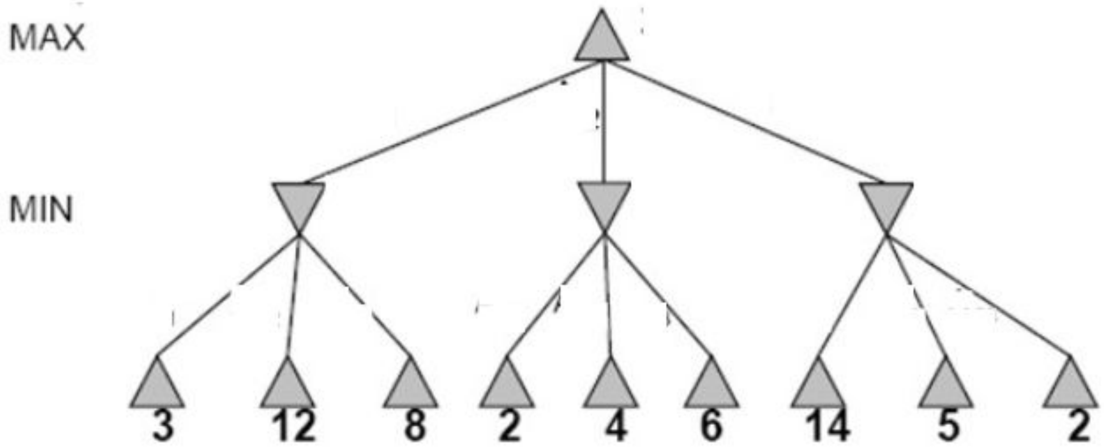
$\max_{s \in successors(n)}$ MINIMAX-VALUE(*s*) **If *n* is a max node**

$\min_{s \in successors(n)}$ MINIMAX-VALUE(*s*) **If *n* is a min node**

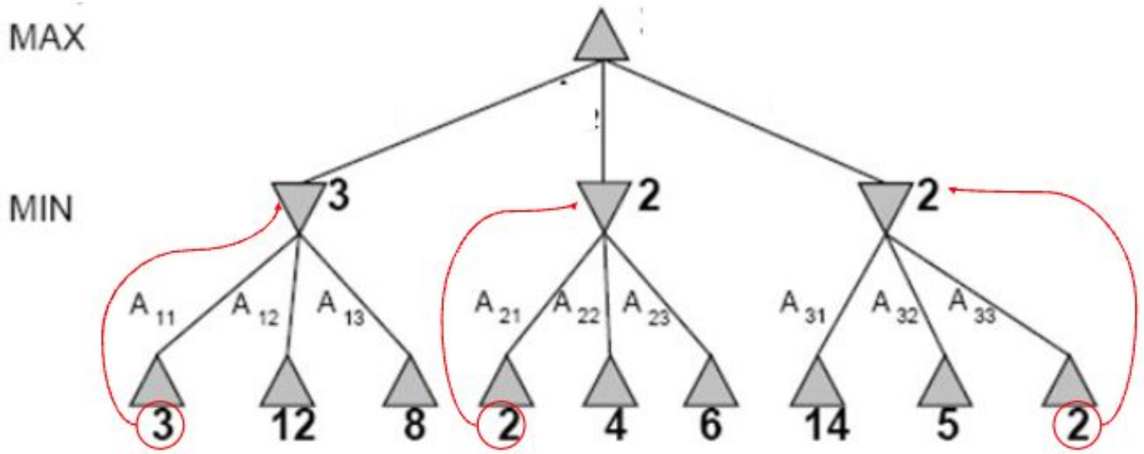
Two-Ply Game Tree



Two-Ply Game Tree

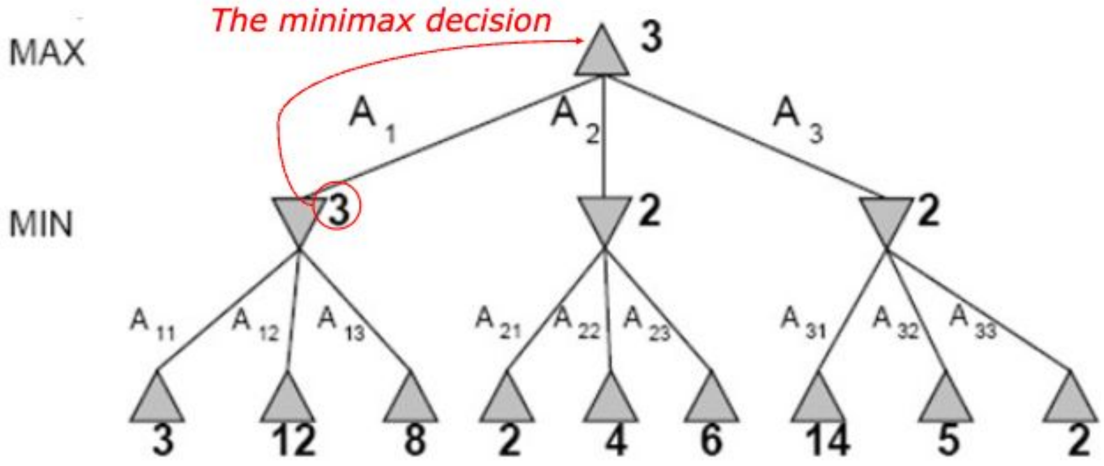


Two-Ply Game Tree



Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max



What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally:
  - maximizes worst-case outcome for MAX
- But if MIN does not play optimally, MAX will do even better

Minimax Algorithm

- Complete depth-first exploration of the game tree

- Assumptions:
  - Max depth = d, b legal moves at each point
  - E.g., Chess: d ~ 100, b ~35

Criterion	Minimax
Time	$O(b^m)$
Space	$O(bm)$

Pseudocode for Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  v←MAX-VALUE(state)
  returnthe action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  fora,s in SUCCESSORS(state) do
    v ← MAX(v,MIN-VALUE(s))
  returnv
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  fora,s in SUCCESSORS(state) do
    v ← MIN(v,MAX-VALUE(s))
  returnv
```

Practical problem with minimax search

- Number of game states is exponential in the number of moves.
  - Solution: Do not examine every node

  => pruning
    - Remove branches that do not influence final decision

- Revisit example …

Alpha-Beta Pruning (budama)

- alt limit ve üst limit gibi düşünebiliriz.
- İçgüdüsel olarak bir hamlenin daha önce hesapladığımız bir hamleden kötü olduğunu garanti edebiliyorsak o hamleye daha fazla bakmamıza gerek yok.

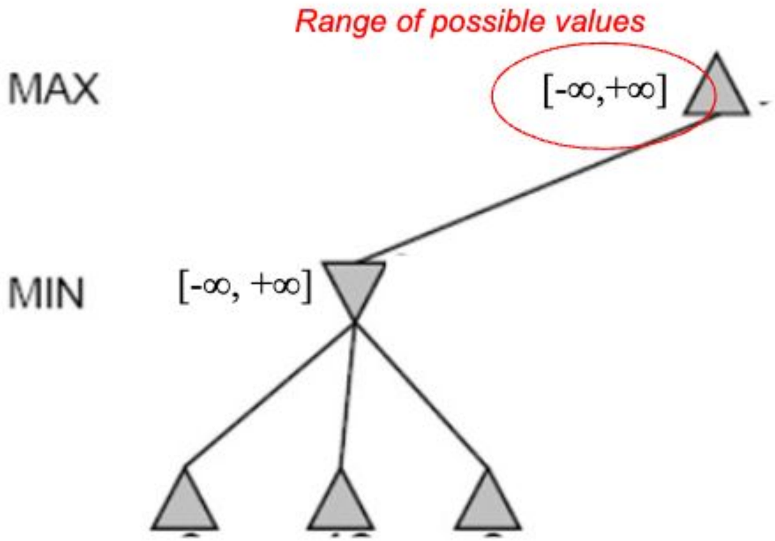
$\alpha =$ the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

$\beta =$ the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

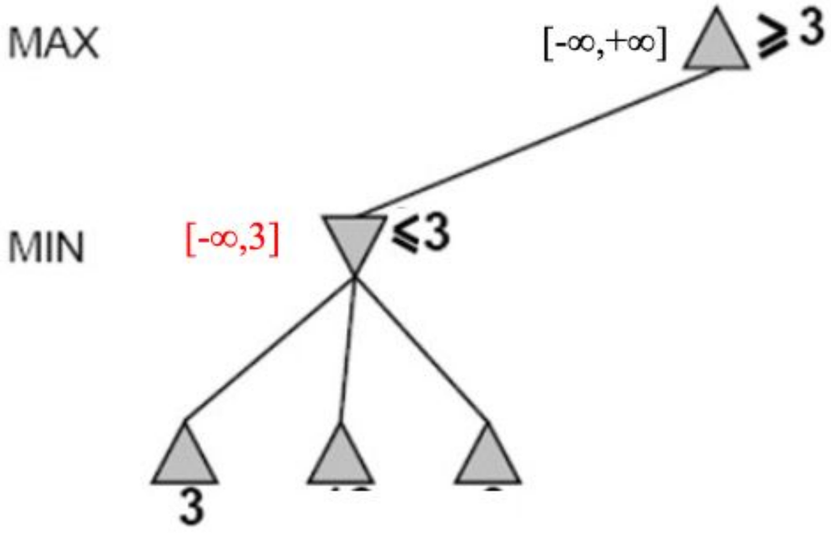
Alpha–beta search updates the values of α and β as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively. The complete algorithm is given in Figure 5.7. We encourage you to trace its behavior when applied to the tree in Figure 5.5.

Alpha-Beta Example

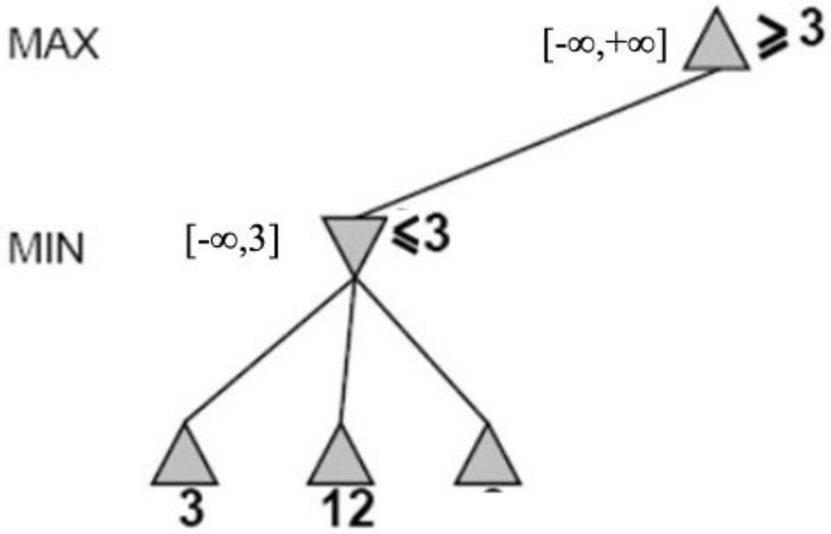
Do DF-search until first leaf



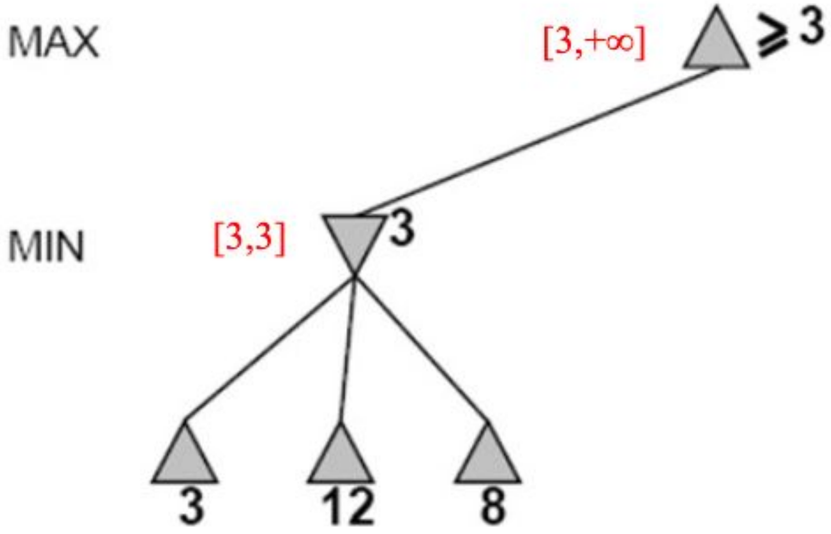
Alpha-Beta Example (continued)



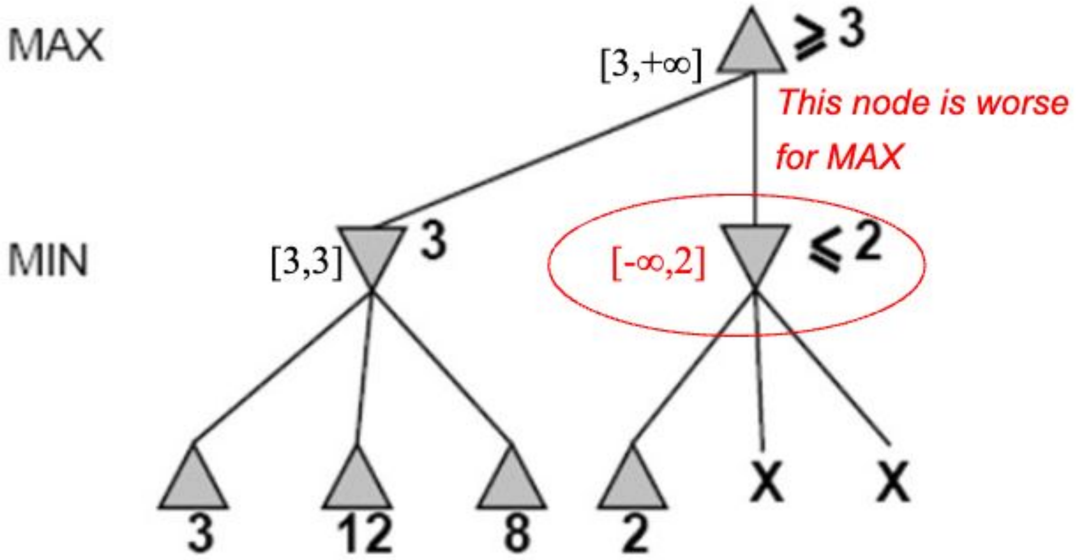
Alpha-Beta Example (continued)



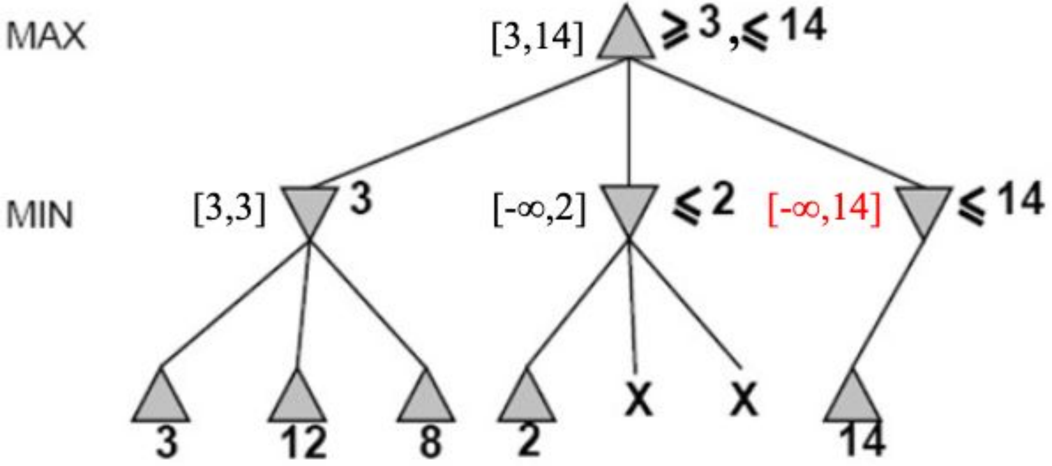
Alpha-Beta Example (continued)



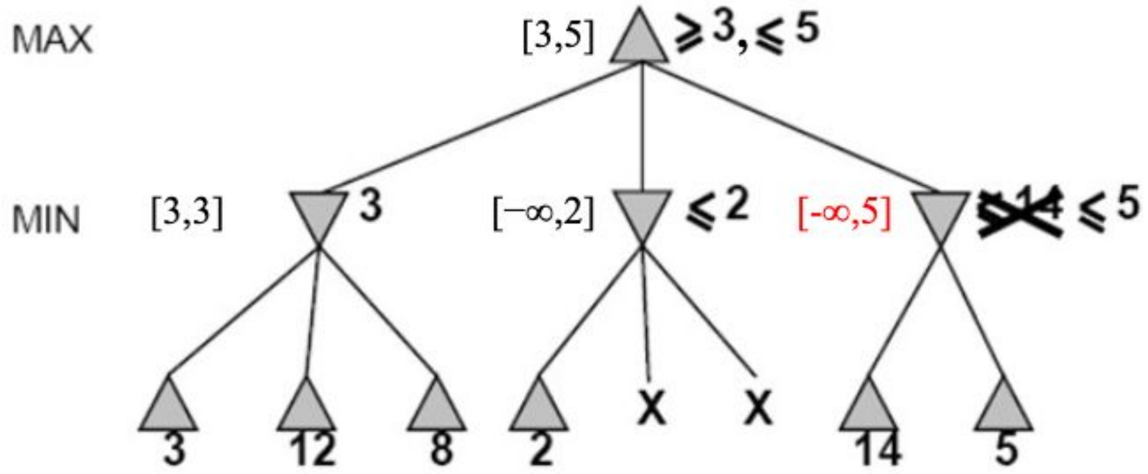
Alpha-Beta Example (continued)



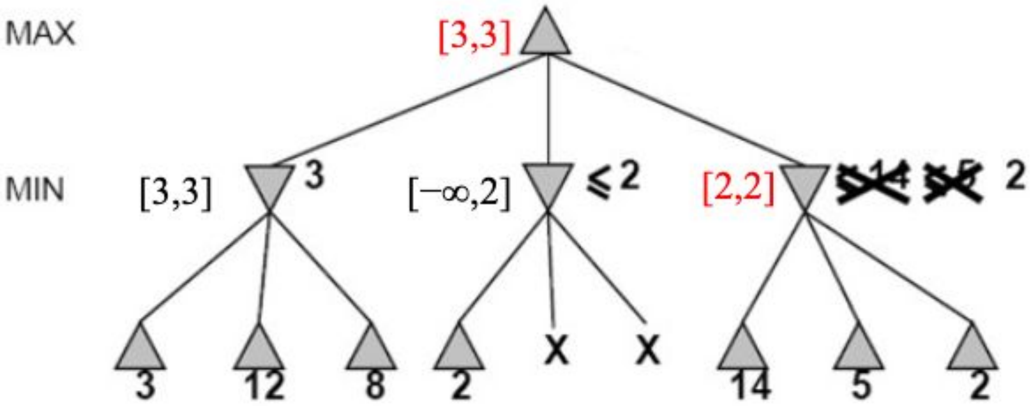
Alpha-Beta Example (continued)



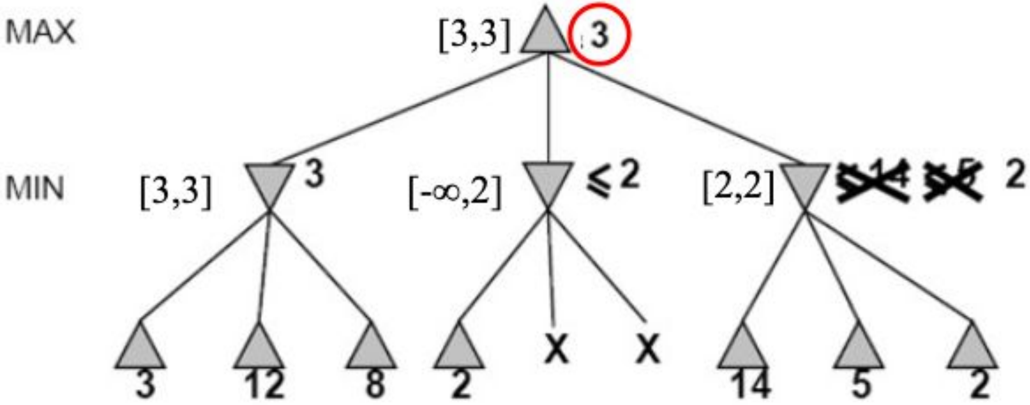
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)

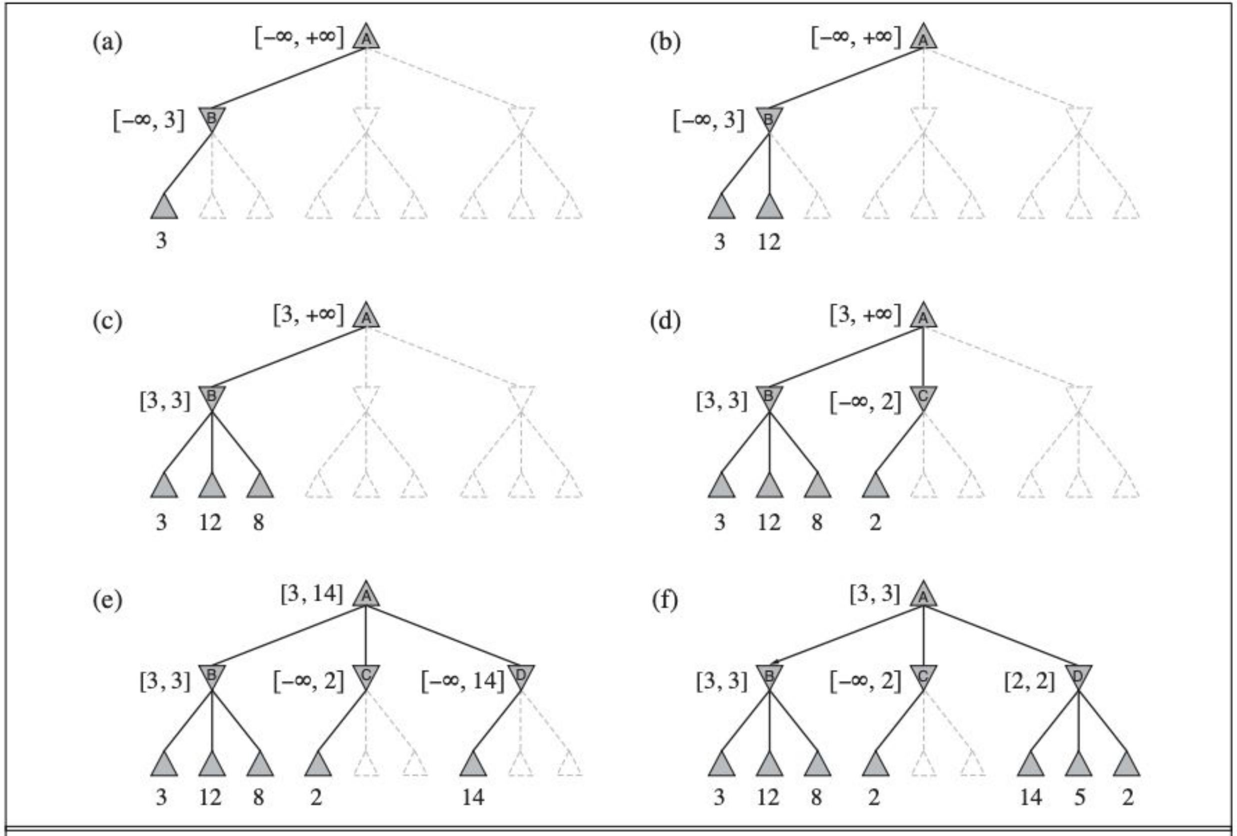


Alpha-Beta Example (continued)



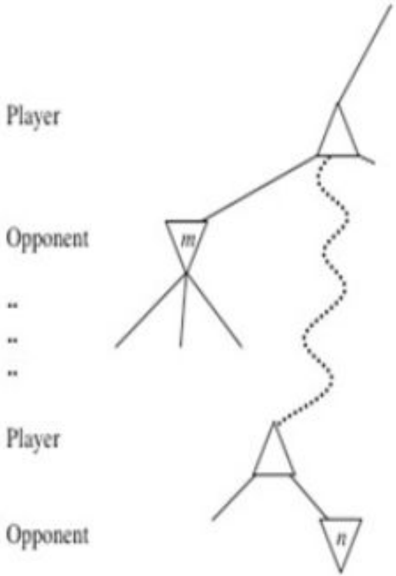
İki derinlik için arama ağacı

- 2 pozisyonu incelememize gerek kalmadı
- Gerçek hayatta her adımda yaklaşık 20-30 olasılık olduğu için kazancımız çok daha fazla olacak.



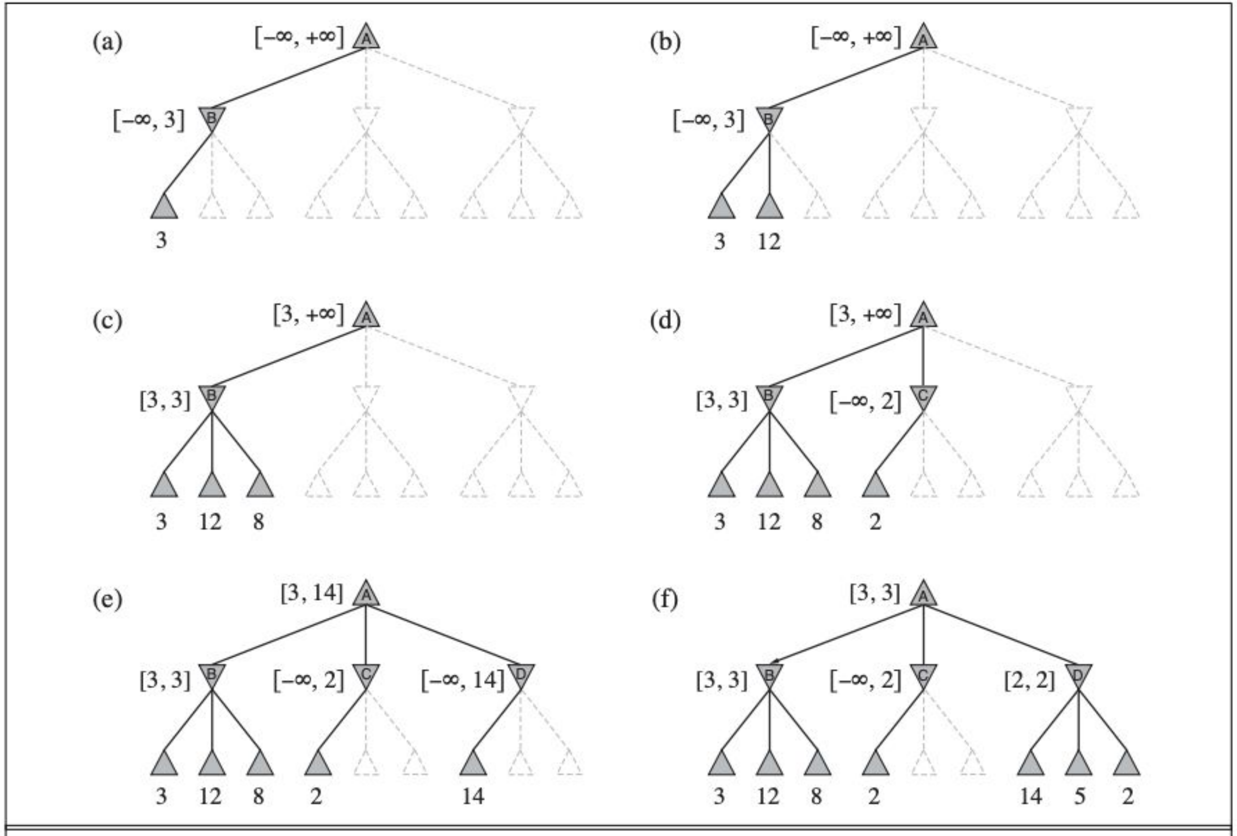
General alpha-beta pruning

- Consider a node *n* somewhere in the tree
- If player has a better choice at
  - Parent node of n
  - Or any choice point further up
- *n* will **never** be reached in actual play.
- Hence when enough is known about *n*, it can be pruned.



İki derinlik için arama ağacı

- 2 pozisyonu incelememize gerek kalmadı
- Gerçek hayatta her adımda yaklaşık 20-30 olasılık olduğu için kazancımız çok daha fazla olacak.



Effectiveness of Alpha-Beta Search

- Worst-Case
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search

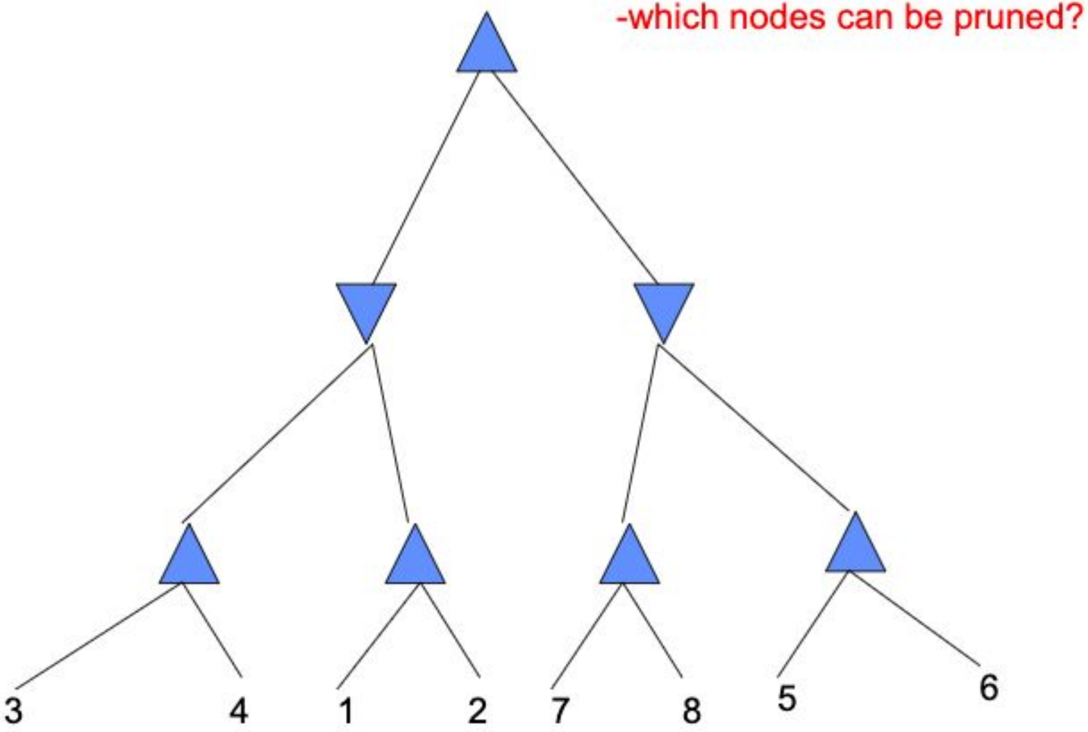
- Best-Case
  - each player's best move is the left-most alternative (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case

- In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$
  - this is the same as having a branching factor of sqrt(b),
    - since $(\mathrm{sqrt}(b))^d = b^{(d/2)}$
    - i.e., we have effectively gone from b to square root of b
  - e.g., in chess go from b ~ 35 to b ~ 6
    - this permits much deeper search in the same amount of time

Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned.
- Good move *ordering* improves effectiveness of pruning
- Repeated states are again possible.
  - Store them in memory = transposition table

Example



Practical Implementation

How do we make these ideas practical in real game trees?

Standard approach:

- cutoff test: (where do we stop descending the tree)
  - depth limit
  - better: iterative deepening
  - cutoff only when no big changes are expected to occur next (quiescence search).
- evaluation function
  - When the search is cut off, we evaluate the current state by estimating its utility. This estimate if captured by the evaluation function.

Gelecek haftaya kısa bir giriş

Evaluation functions

Black to move

White slightly better

White to move

Black winning

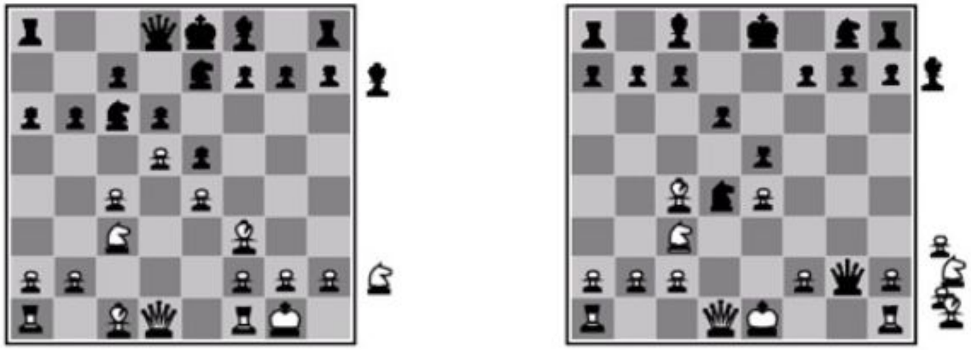
For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Gelecek haftaya kısa bir giriş

Evaluation functions



Black to move

White slightly better

White to move

Black winning

For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Ders Sonu

Teşekkürler!!

Sorularınız veya önerileriniz?

Önümüzdeki Hafta:

- Imperfect decisions
    - **Evaluation Functions** -> Proje bu konuyla ilgili olacak
    - Forward Pruning
    - Şansa dayalı oyunlar (stochastic games)
- Uygulama ve örnek problem çözme