Two-player Games

PAH 2012/2013

Branislav Bošanský

Game-tree Search / Adversarial Search

- planning – only the searching player acts in the environment
- there could be others:
  - Nature – stochastic environment (MDP, POMDP, …)
  - other agents – rational opponents

- Game Theory
  - mathematical framework that describes optimal behavior of rational self-interested agents
  - A4M36MAS (Multi-agent Systems)

Game-tree Search / Adversarial Search

- What are the basic games categories?
  - perfect / imperfect information
  - deterministic / stochastic
  - zero-sum / general-sum
  - finite / infinite
  - two-player / n-player
  - …

Game-tree Search / Adversarial Search

- What are the basic games categories?
  - **perfect** / imperfect information
  - **deterministic** / stochastic
  - **zero-sum** / general-sum
  - **finite** / infinite
  - **two-player** / n-player
  - …

Game-tree Search / Adversarial Search

- What are the basic games categories?
  - **perfect** / imperfect information
  - **deterministic** / stochastic
  - **zero-sum** / general-sum
  - **finite** / infinite
  - **two-player** / n-player
  - …

- What is the goal?

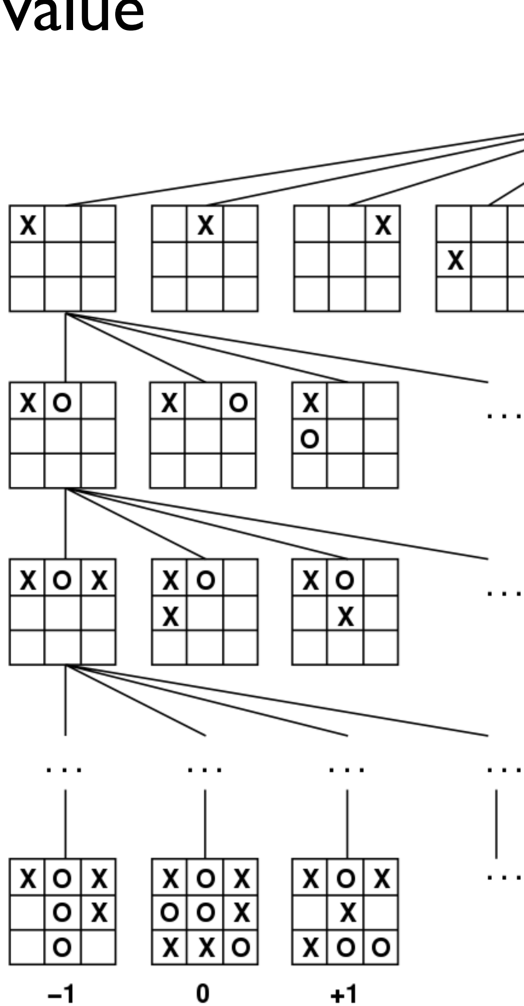
Game-tree Search / Adversarial Search

- What are the basic games categories?
  - **perfect** / imperfect information
  - **deterministic** / stochastic
  - **zero-sum** / general-sum
  - **finite** / infinite
  - **two-player** / n-player
  - …

- What is the goal?
  - finding an optimal **strategy** (i.e., selecting an action to play in each possible situation)

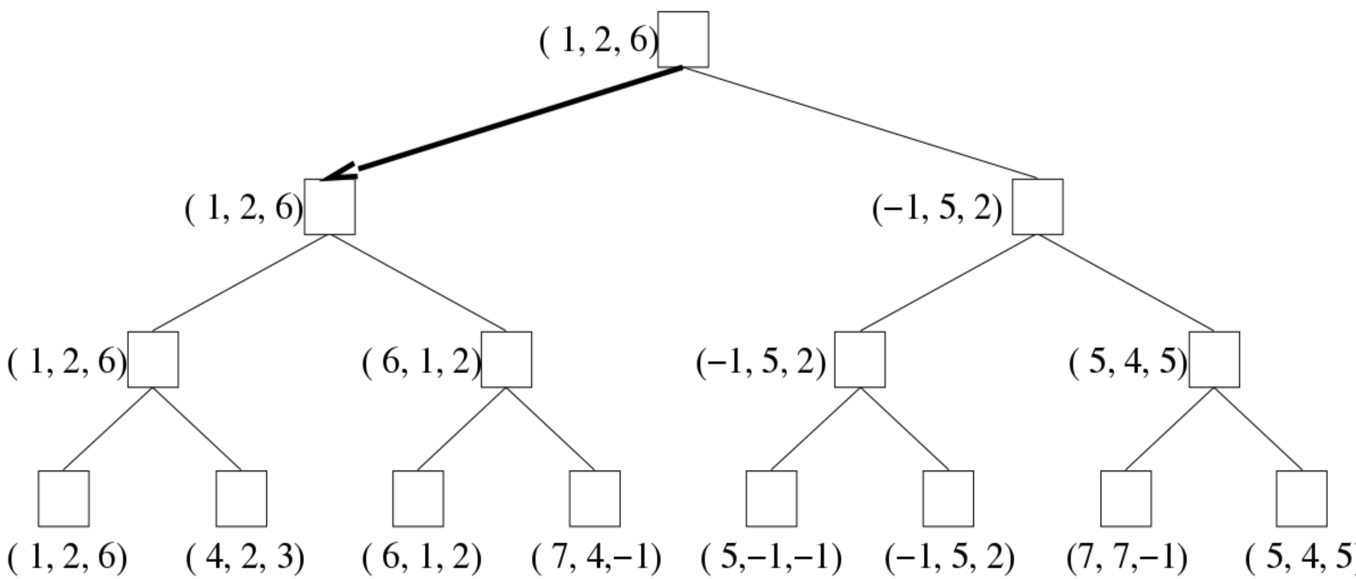
Game-tree Search / Adversarial Search

- Players are rational – each player wants to maximize her/his utility value



Game-tree Search / Adversarial Search

- Players are rational – each player wants to maximize her/his utility value



Minimax

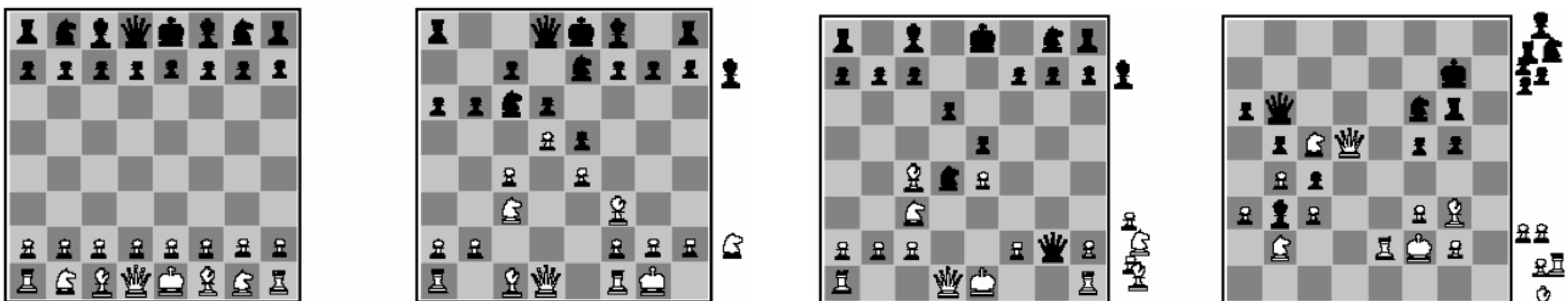
- **function** minimax(node, Player)
- **if** (node is a terminal node) **return** utility value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v ← max(v, minimax(child, switch(Player)))
- **return** v
- **else**
- **for each** child of node
- v ← min(v, minimax(child, switch(Player)))
- **return** v

Minimax in Real Games

- search space in games is typically **very large**
  - exponential in branching factor b^d
    - e.g., 35 in chess, up to 360 in Go, up to 45000 in Arimaa
- we have to limit the depth of the search
- we need an evaluation function

Minimax in Real Games

- search space in games is typically **very large**
  - exponential in branching factor b^d
    - e.g., 35 in chess, up to 360 in Go, up to 45000 in Arimaa
- we have to limit the depth of the search
- we need an evaluation function



(a) White to move
Fairly even

(b) Black to move
White slightly better

(c) White to move
Black winning

(d) Black to move
White about to lose

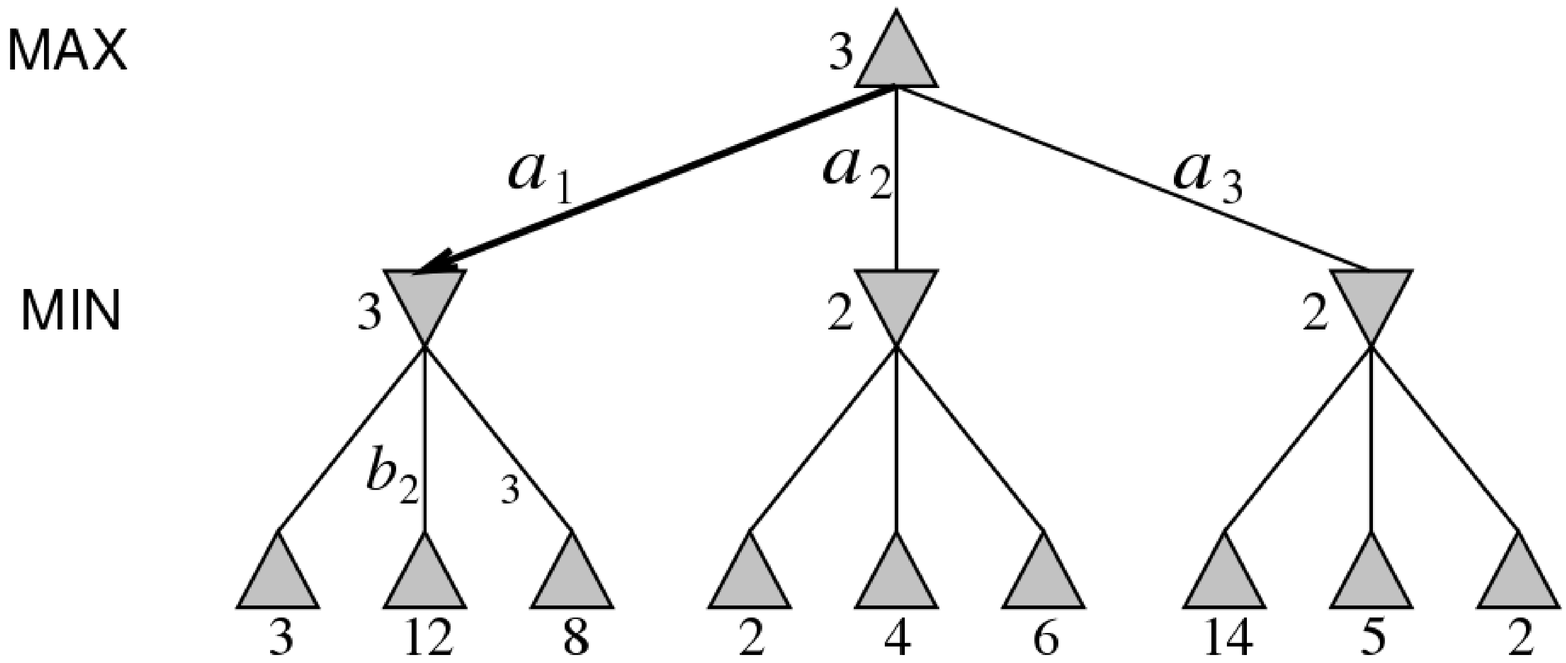
Minimax

- **function** minimax(node, depth, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v ← max(v, minimax(child, depth-1, switch(Player)))
- **return** v
- **else**
- **for each** child of node
- v ← min(v, minimax(child, depth-1, switch(Player)))
- **return** v

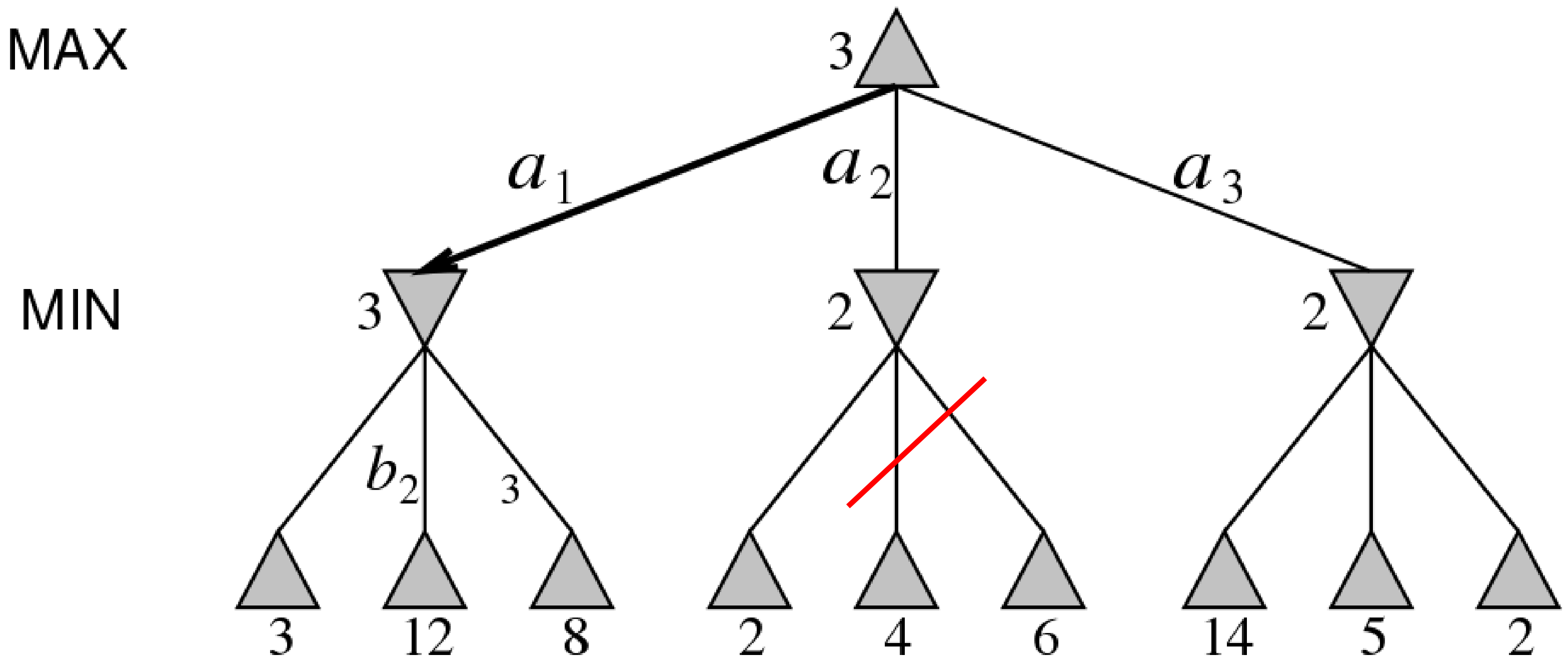
Minimax in Real Games - Problems

- good evaluation function
- depth?
  - horizon problem
  - iterative deepening
  - not always searching deeper improve the results
- caching the results (transposition tables)
- …

Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning

- **function** alphabeta(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v ← max(v, alphabeta(child, depth-1, α, β, switch(Player)))
- α := max(α,v)
- **if** (β≤α) break
- **return** v
- **else**
- **for each** child of node
- v ← min(v, alphabeta(child, depth-1, α, β, switch(Player)))
- β := min(β, v)
- **if** (β≤α) break
- **return** v

Negamax

- **function** negamax(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v ← max(v, -negamax(child, depth-1, -β, -α, switch(Player)))
- α := max(α,v)
- **if** (β≤α) break
- **return** v
- **else**
- **for each** child of node
- v ← min(v, alphabeta(child, depth-1, α, β, switch(Player)))
- β := min(β, v)
- **if** (β≤α) break
- **return** v

Aspiration Search

- $[\alpha, \beta]$ interval – window
- alphabeta initialization $[-\infty, +\infty]$
- what if we use $[\alpha_0, \beta_0]$
  - x = alphabeta(node, depth, α_0, β_0,player)
  - $\alpha_0 \leq x \leq \beta_0$ - we found a solution
  - $x \leq \alpha_0$ - failing low (run again with $[-\infty, x]$)
  - $x \geq \beta_0$ - failing high (run again with $[x, +\infty]$)

Scout – Idea

- assume we are in a MAX node
- we are about to search a child 'c'
- we already have obtained a lower bound 'α'

- Is it worth searching the branch 'c'?

- we need to have some test ...

Scout –Test

- what we really need at that moment is a bound (not the precise value)
- Remember Aspiration Search?
  - $x \leq \alpha_0$ - failing low (we know, that solution is $\leq x$)
  - $x \geq \beta_0$ - failing high (we know, that solution is $\geq x$)
- What if we use a null-window $[\alpha, \alpha+1]$ (or $[\alpha,\alpha]$)?
  - we obtain a bound …

NegaScout

function negascout(node, depth, α, β, Player)

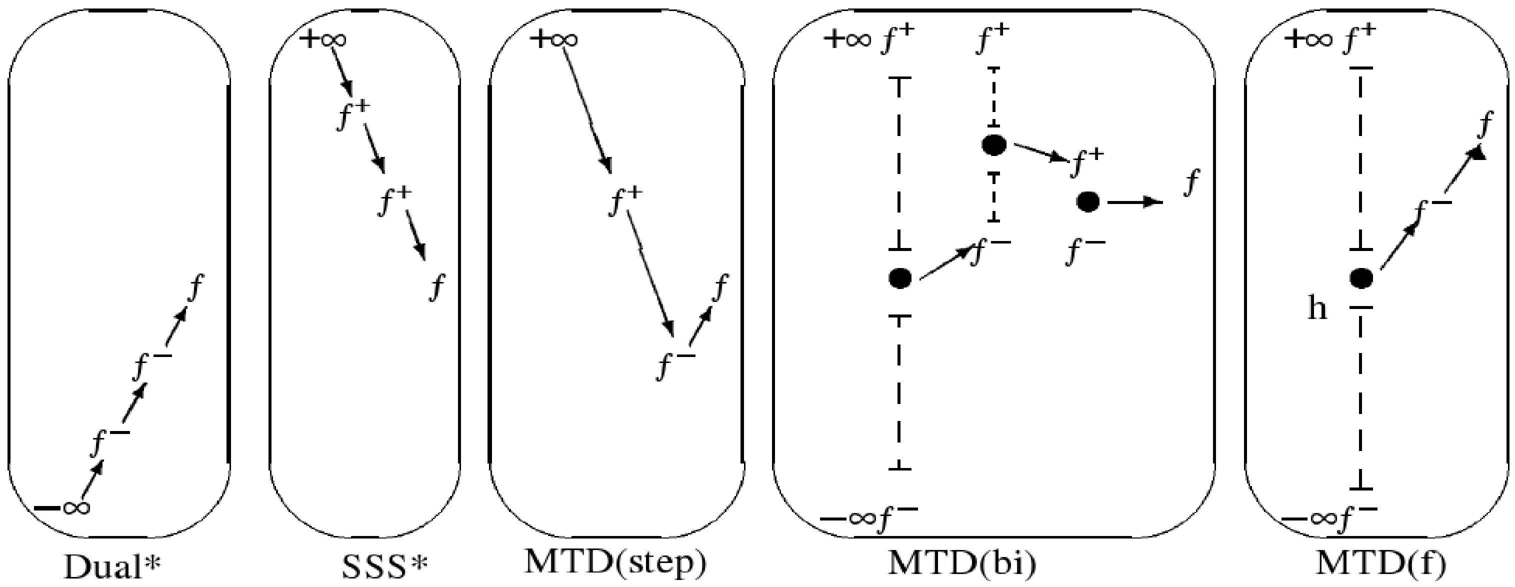
- **if** ((depth = 0) or (node is a terminal node)) **return** eval(node)
- b := β
- **for each** child of node
- v := max(v, -negascout(child, depth-1, -b, -α, switch(Player))))
- **if** ((α < v < β) and (child is not the first child))
- v := max(v, -negascout(child, depth-1, -β, -α, switch(Player))))
- α := max(α, v)
- **if** (β≤α) break
- b := α + 1
- **return** v

NegaScout

- also termed Principal Variation Search (PVS)
- dominates alpha-beta
  - never evaluates more different nodes than alpha-beta
  - can evaluate some nodes more than once
- depends on the move ordering
- can benefit from transposition tables
- generally 10-20% faster compared to alpha-beta

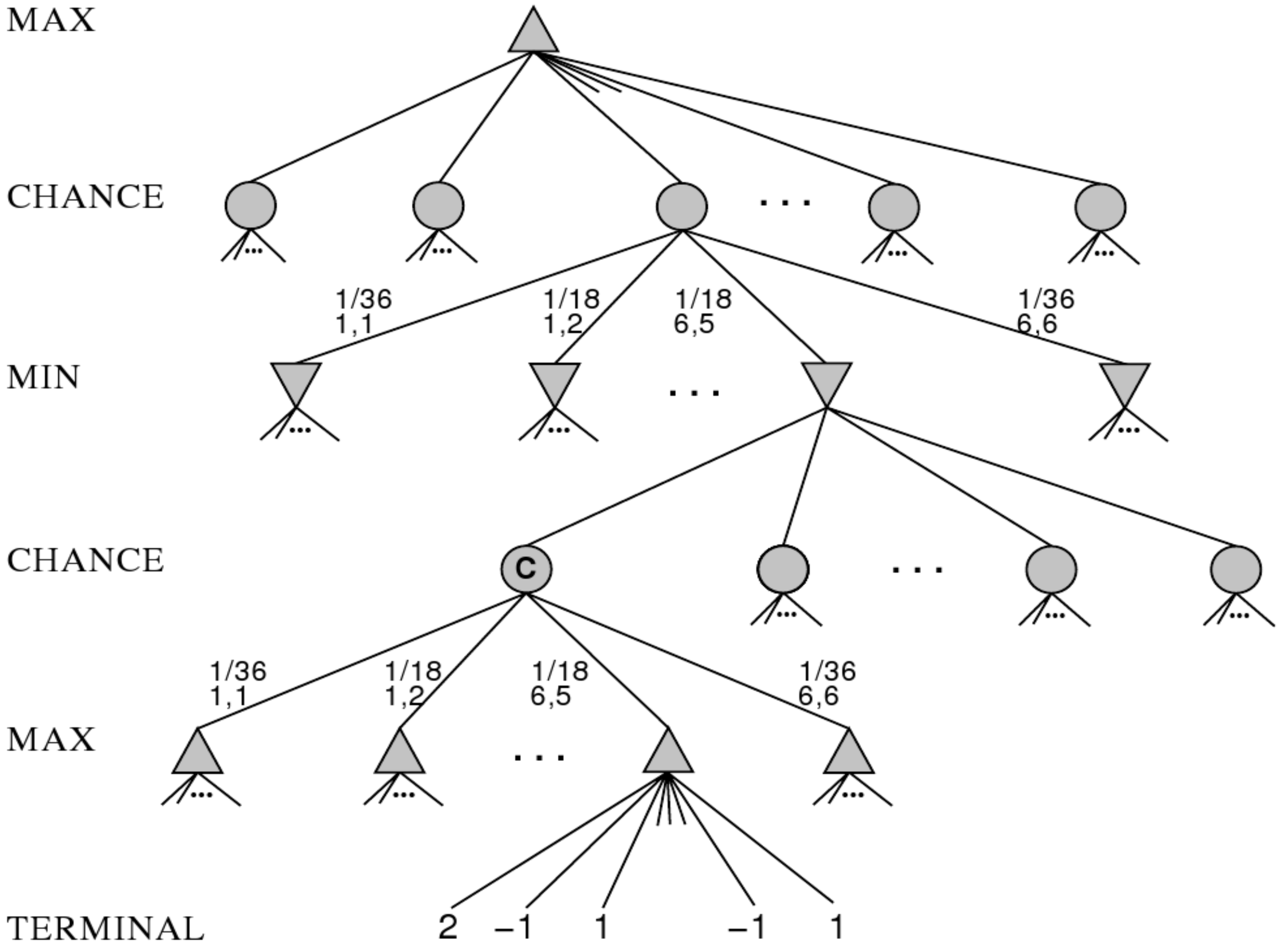
MTD

- Memory-enhanced Test Driver

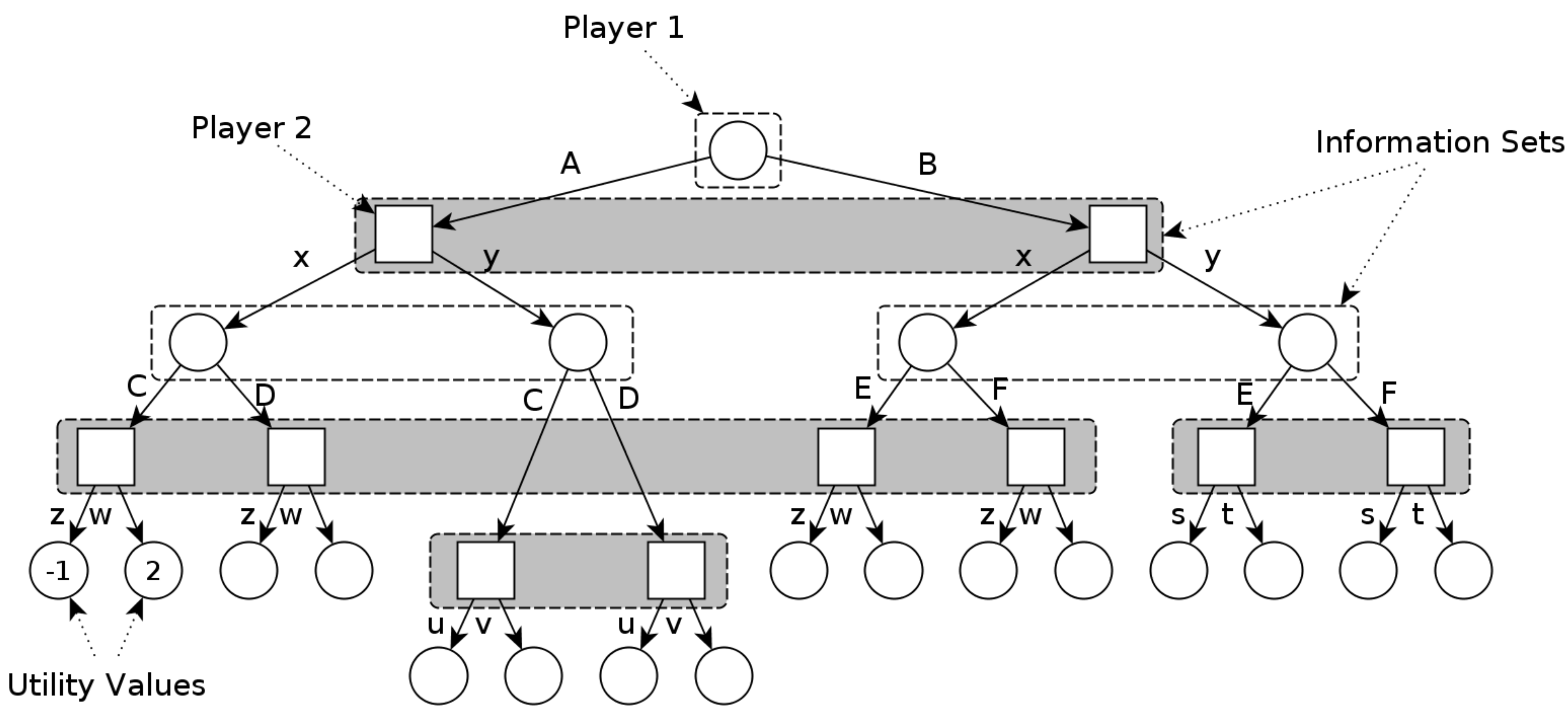


- Best-first fixed-depth minimax algorithms. Plaat et. al. , In *Artificial Intelligence,* Volume 87, Issues 1-2, November 1996, Pages 255-293

Other Games - Chance nodes



Other Games – Imperfect Information



Other Algorithms

- Sequence-form linear program
- Counterfactual Regret Minimization
- Monte-carlo Tree Search
- Double-oracle methods
- …

Game Theory in ATG

- generic algorithms
- sequential games
  - with simultaneous moves
  - with imperfect information (Poker, Security Games)
- complex strategies, abstractions
- more general types of ‘solutions’
- …