

## CS 1571 Introduction to AI Lecture 10

### Finding optimal configurations Adversarial search

**Milos Hauskrecht**  
[milos@cs.pitt.edu](mailto:milos@cs.pitt.edu)  
5329 Sennott Square

### Announcements

- **Homework assignment 3 due today**
- **Homework assignment 4 is out**
  - Programming and experiments
  - Simulated annealing + Genetic algorithm
  - Competition

**Course web page:**

<http://www.cs.pitt.edu/~milos/courses/cs1571/>

## Parametric optimization

### Optimal configuration search:

- Configurations are described in terms of variables and their values
- Each configuration has a quality measure
- Goal: find the configuration with the best value

When the state space we search is finite, the search problem is called a **combinatorial optimization problem**

When parameters we want to find are real-valued

– **parametric optimization problem**

## Parametric optimization

### Parametric optimization:

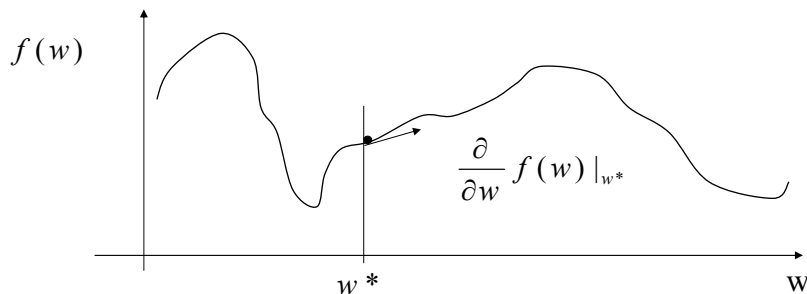
- Configurations are described by a vector of free parameters (variables)  $\mathbf{w}$  with real-valued values
- **Goal:** find the set of parameters  $\mathbf{w}$  that optimize the quality measure  $f(\mathbf{w})$

## Parametric optimization techniques

- **Special cases (with efficient solutions):**
  - Linear programming
  - Quadratic programming
- **First-order methods:**
  - Gradient-ascent (descent)
  - Conjugate gradient
- **Second-order methods:**
  - Newton-Rhapson methods
  - Levenberg-Marquardt
- **Constrained optimization:**
  - Lagrange multipliers

## Gradient ascent method

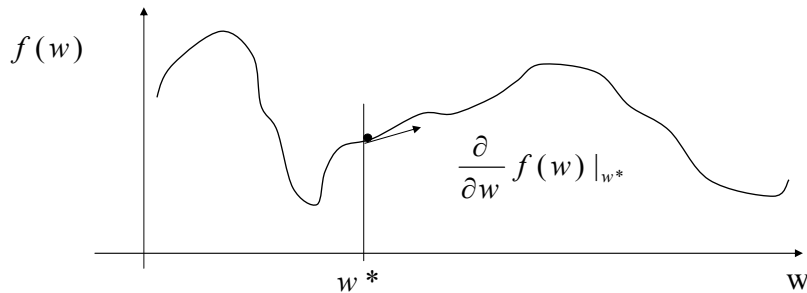
- **Gradient ascent:** the same as hill-climbing, but in the continuous parametric space  $\mathbf{w}$



- What is the derivative of an increasing function?

## Gradient ascent method

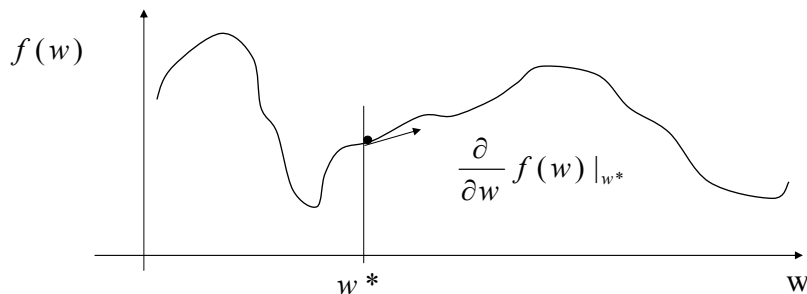
- **Gradient ascent:** the same as hill-climbing, but in the continuous parametric space  $w$



- What is the derivative of an increasing function?
  - positive

## Gradient ascent method

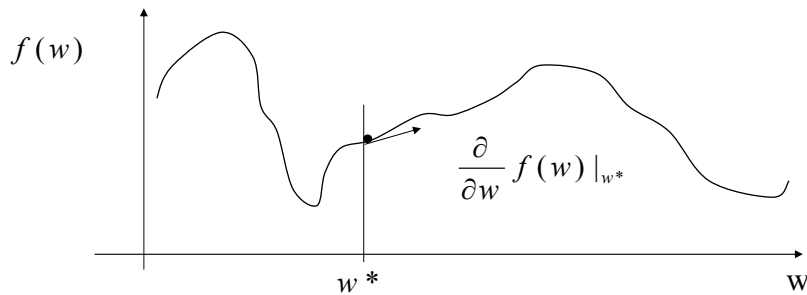
- **Gradient ascent:** the same as hill-climbing, but in the continuous parametric space  $w$



- Change the parameter value of  $w$  according to the gradient

$$w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) |_{w^*}$$

## Gradient ascent method



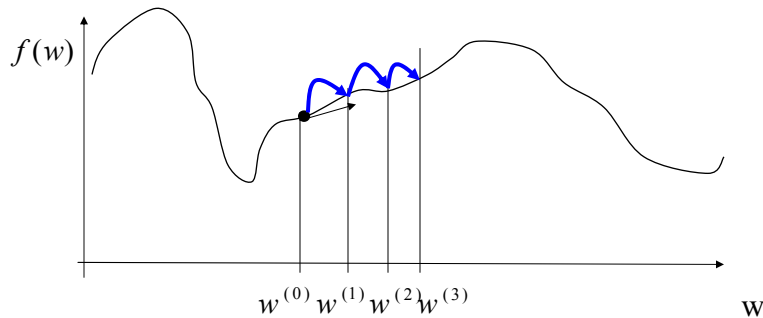
- New value of the parameter

$$w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) |_{w^*}$$

$\alpha > 0$  - a learning rate (scales the gradient changes)

## Gradient ascent method

- To get to the function minimum repeat (iterate) the gradient based update few times



- **Problems:** local optima, saddle points, slow convergence
- More complex optimization techniques use additional information (e.g. second derivatives)

# Adversarial search

## Search review

### Search

- Path search
- Configuration search

### Optimality

- Finding a path versus finding the optimal path
- Finding a configuration satisfying constraints versus finding the best configuration

## Game search

- Game-playing programs developed by AI researchers since the beginning of the modern AI era
  - Programs playing chess, checkers, etc (1950s)
- **Specifics of the game search:**
  - Sequences of player's decisions **we control**
  - Decisions of other player(s) **we do not control**
- **Contingency problem:** many possible opponent's moves must be “covered” by the solution  
Opponent's behavior introduces an uncertainty in to the game
  - We do not know exactly what the response is going to be
- **Rational opponent** – maximizes it own **utility (payoff) function**

## Types of game problems

- **Types of game problems:**
  - **Adversarial games:**
    - win of one player is a loss of the other
  - **Cooperative games:**
    - players have common interests and utility function
  - **A spectrum of game problems in between the two:**

Adversarial games

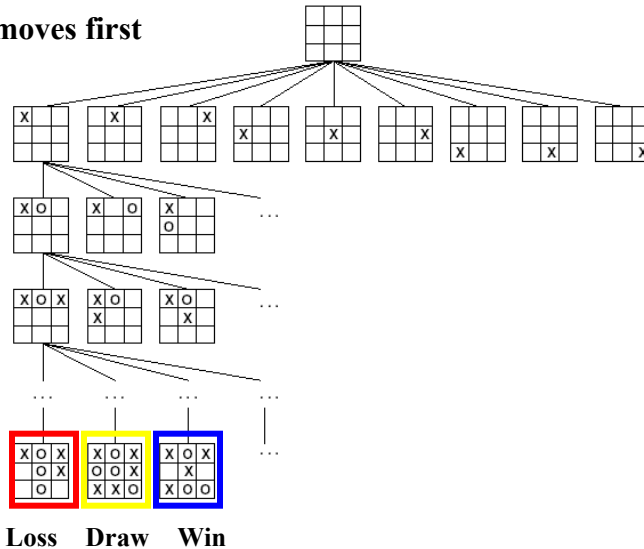
Fully cooperative games



**we focus on adversarial games only!!**

## Example of an adversarial 2 person game: Tic-tac-toe

- Player 1 (x) moves first



## Game search problem

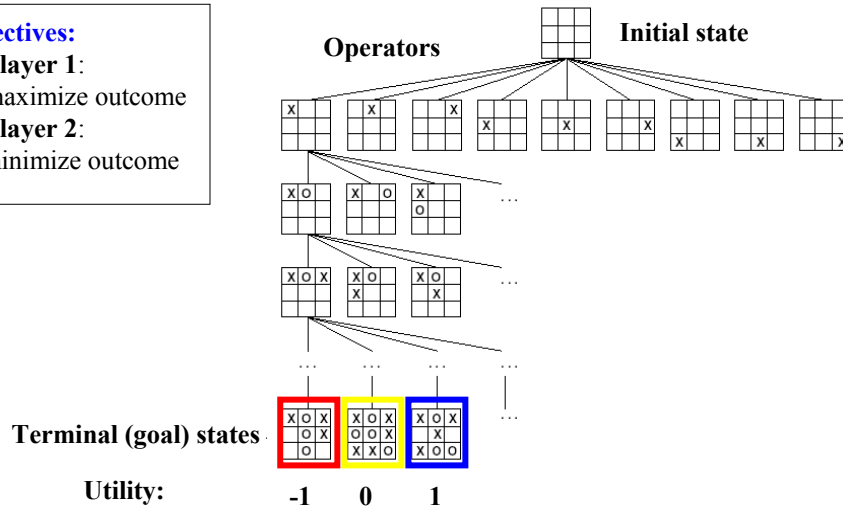
- **Game problem formulation:**
  - **Initial state:** initial board position + info whose move it is
  - **Operators:** legal moves a player can make
  - **Goal (terminal test):** determines when the game is over
  - **Utility (payoff) function:** measures the outcome of the game and its desirability
- **Search objective:**
  - find the sequence of player's decisions (moves) maximizing its utility (payoff)
  - Consider the opponent's moves and their utility



## Game problem formulation (Tic-tac-toe)

### Objectives:

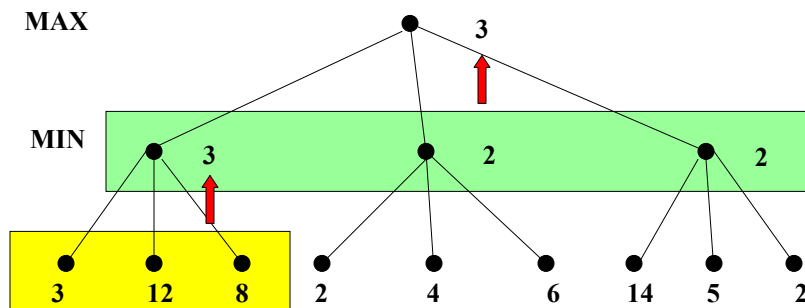
- **Player 1:**  
maximize outcome
- **Player 2:**  
minimize outcome



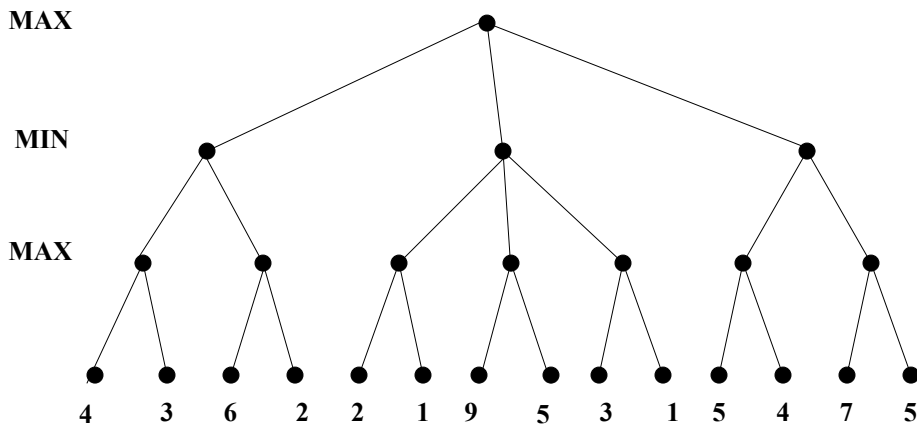
## Minimax algorithm

How to deal with the contingency problem?

- Assuming that the opponent is rational and always optimizes its behavior (opposite to us) we consider **the best opponent's response**
- Then the **minimax algorithm** determines the best move



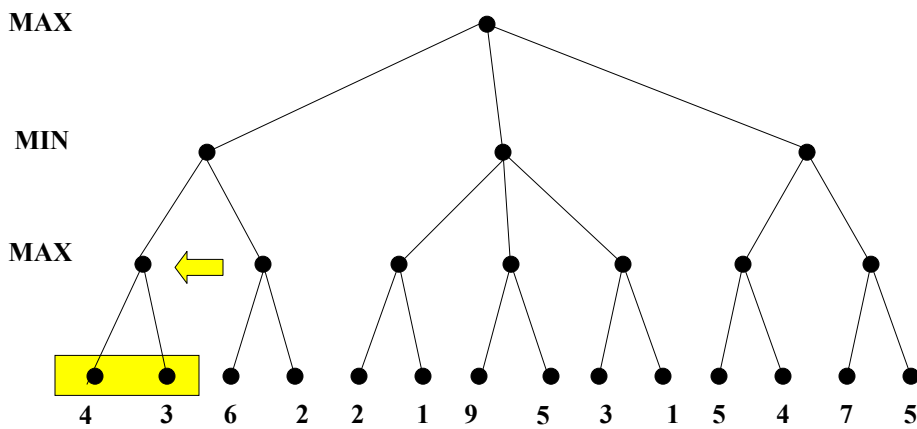
## Minimax algorithm. Example



CS 1571 Intro to AI

M. Hauskrecht

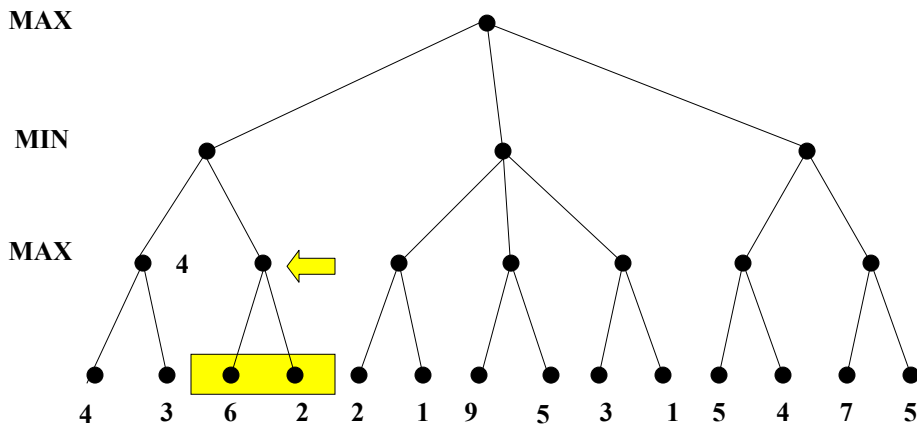
## Minimax algorithm. Example



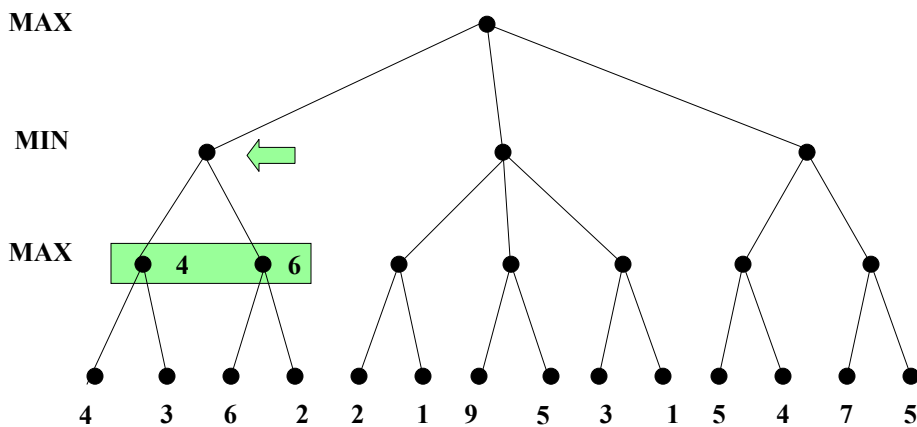
CS 1571 Intro to AI

M. Hauskrecht

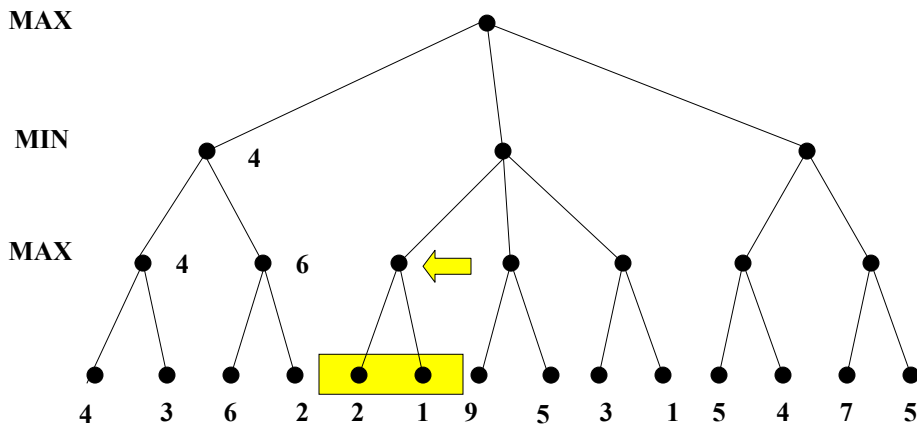
## Minimax algorithm. Example



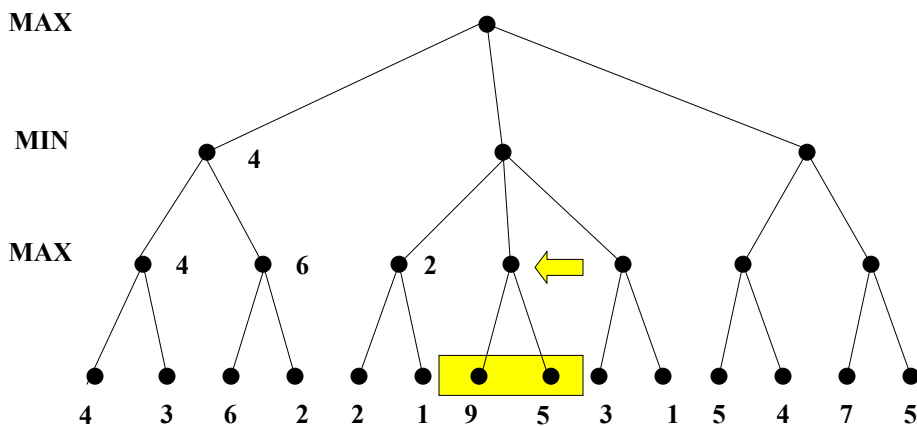
## Minimax algorithm. Example



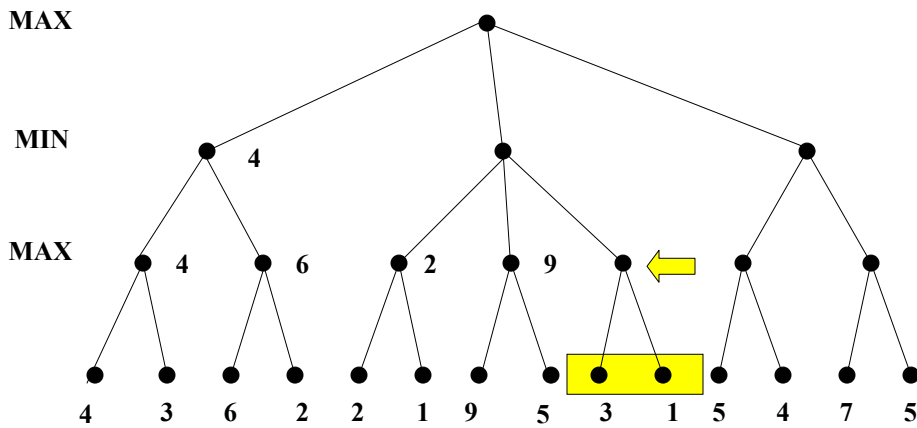
## Minimax algorithm. Example



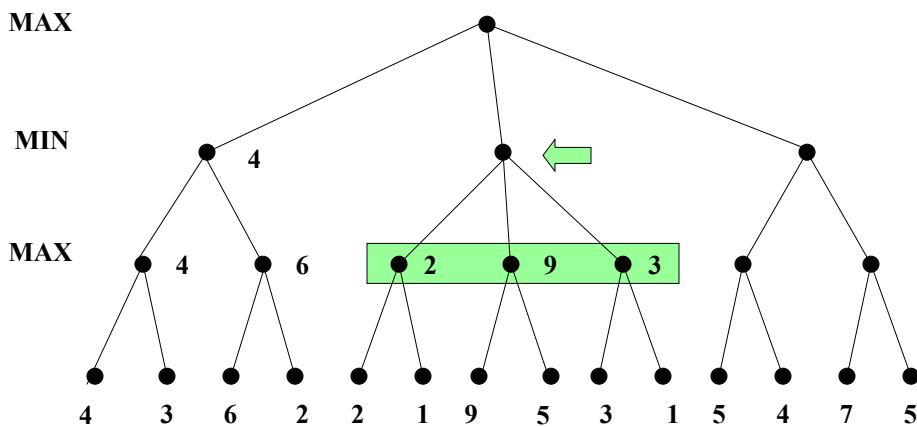
## Minimax algorithm. Example



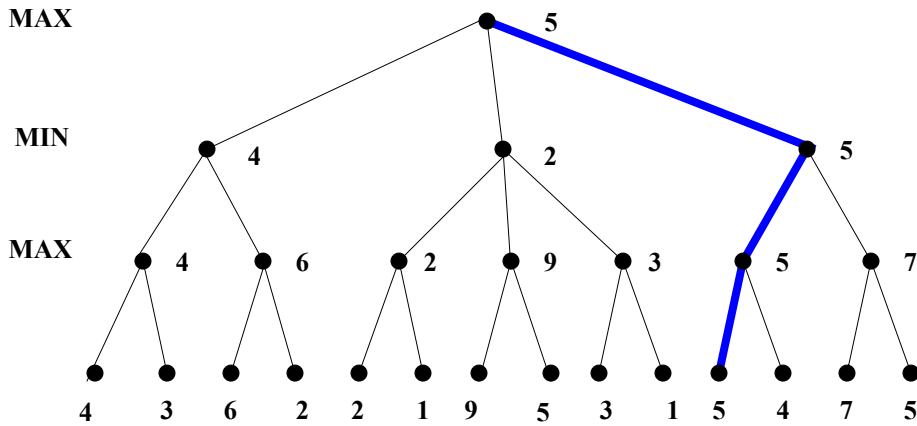
## Minimax algorithm. Example



## Minimax algorithm. Example



## Minimax algorithm. Example



## Minimax algorithm

```

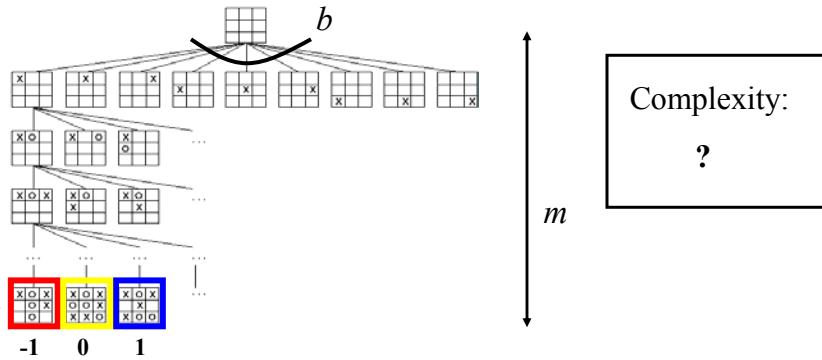
function MINIMAX-DECISION(game) returns an operator
  for each op in OPERATORS[game] do
    VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]
  
```

```

function MINIMAX-VALUE(state, game) returns a utility value
  if TERMINAL-TEST[game](state) then
    return UTILITY[game](state)
  else if MAX is to move in state then
    return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(state)
  
```

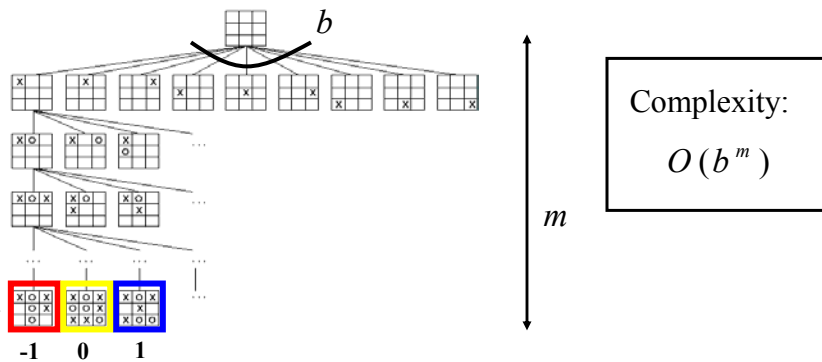
## Complexity of the minimax algorithm

- We need to explore the complete game tree before making the decision



## Complexity of the minimax algorithm

- We need to explore the complete game tree before making the decision



- Impossible for large games
  - Chess: 35 operators, game can have 50 or more moves

## Solution to the complexity problem

**Two solutions:**

**1. Dynamic pruning of redundant branches** of the search tree

- identify a provably suboptimal branch of the search tree before it is fully explored
- Eliminate the suboptimal branch

**Procedure: Alpha-Beta pruning**

**2. Early cutoff of the search tree**

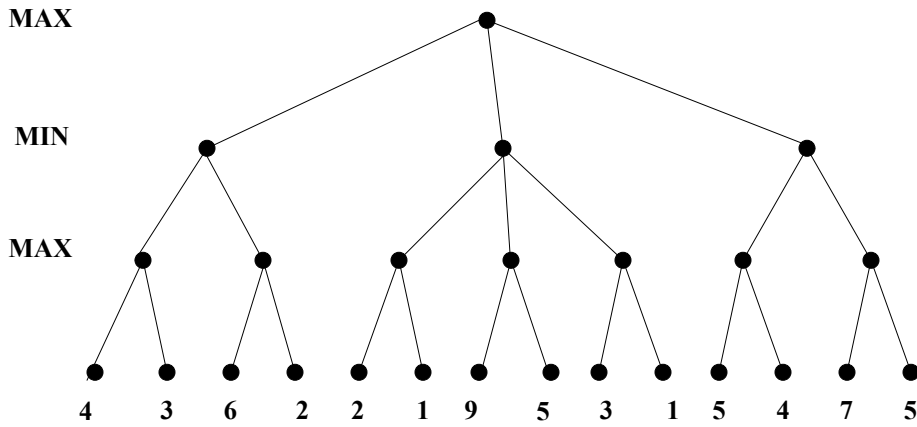
- uses imperfect minimax value estimate of non-terminal states (positions)

## Alpha beta pruning

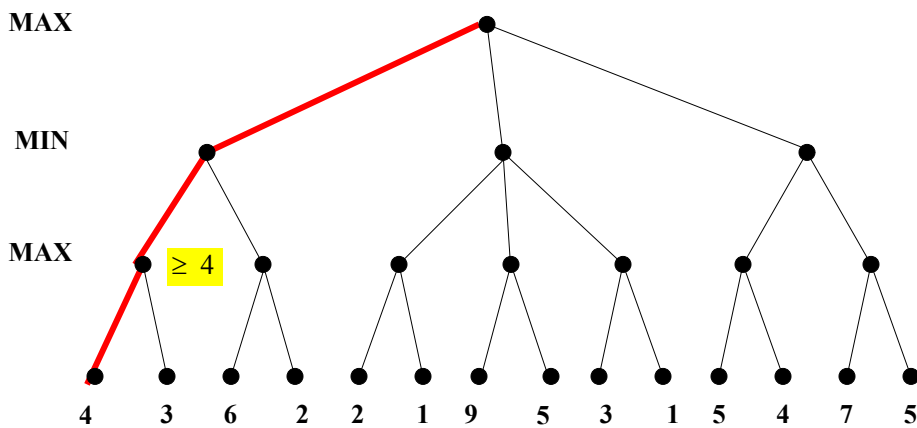
- Some branches will never be played by rational players since they include sub-optimal decisions (for either player)



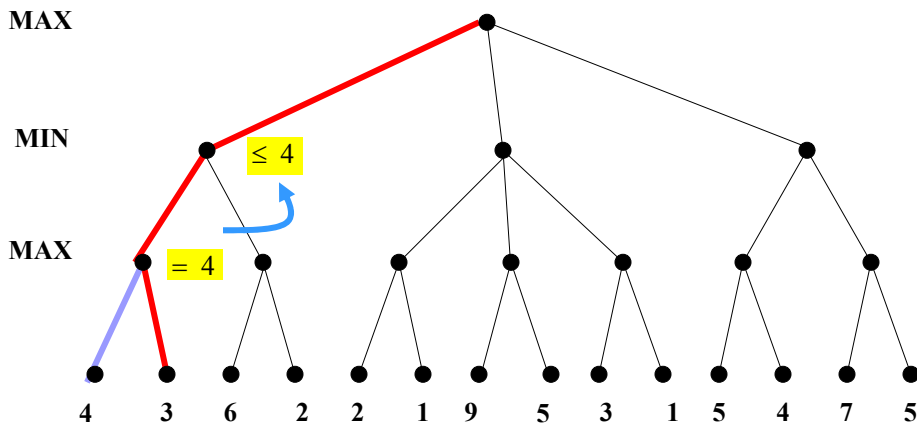
## Alpha beta pruning. Example



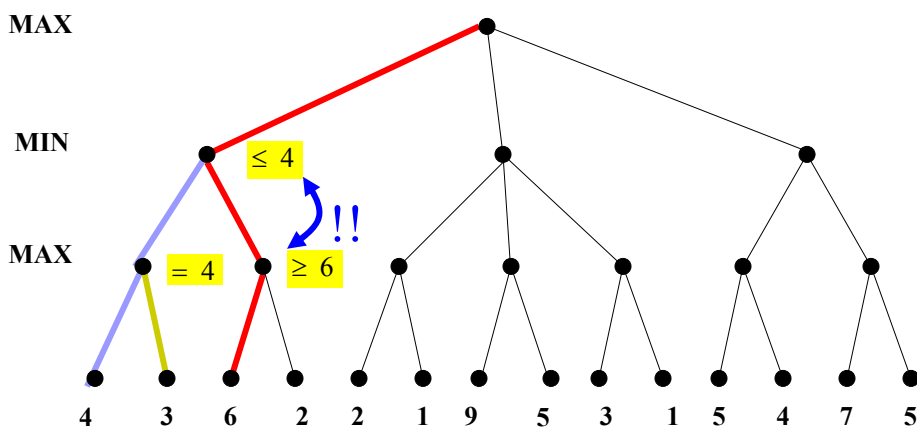
## Alpha beta pruning. Example



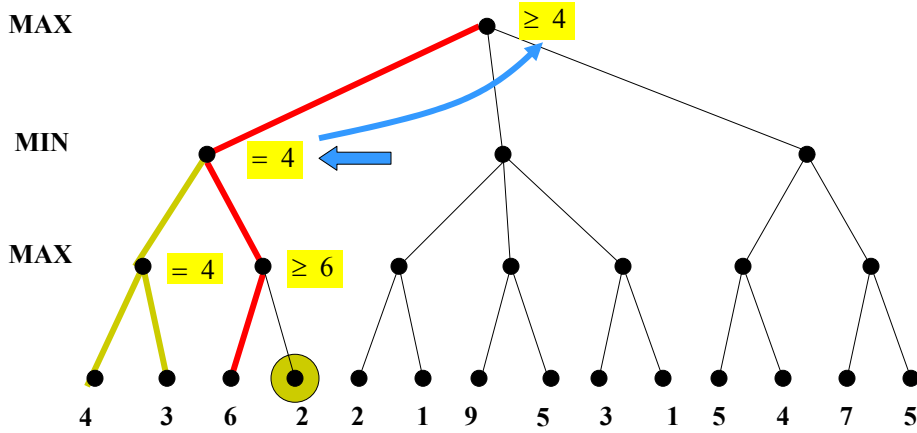
## Alpha beta pruning. Example



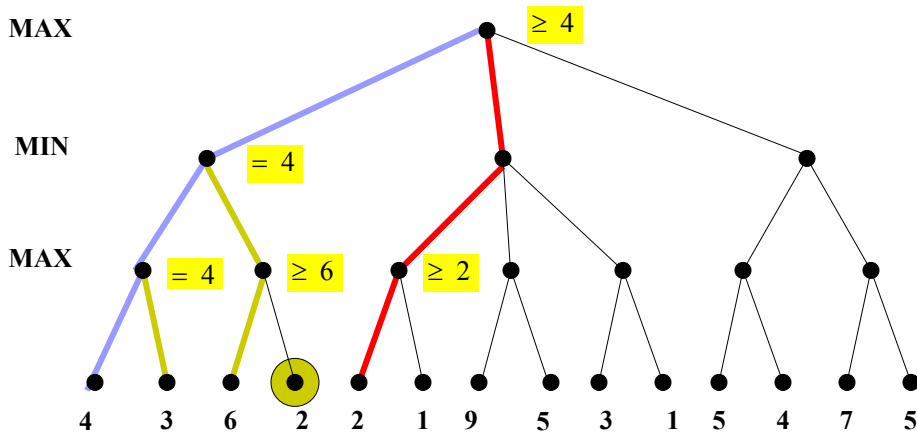
## Alpha beta pruning. Example



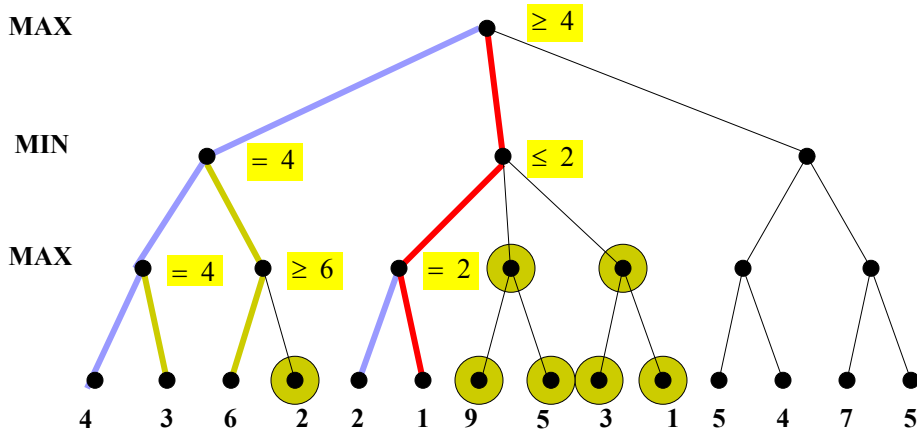
## Alpha beta pruning. Example



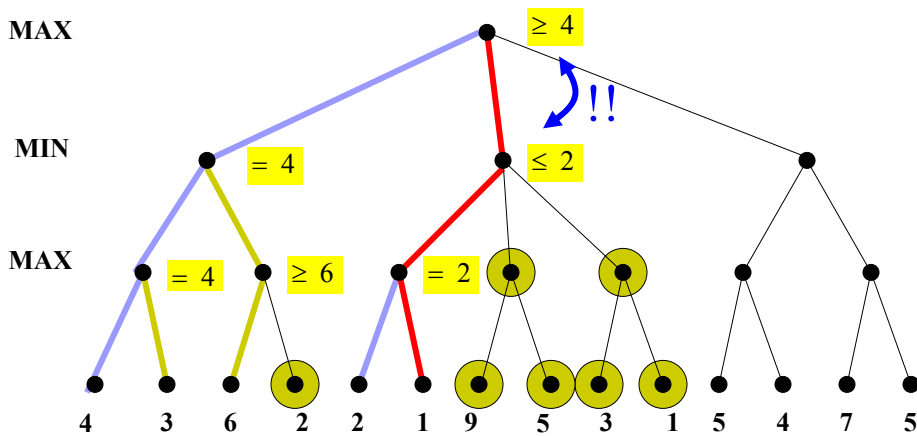
## Alpha beta pruning. Example



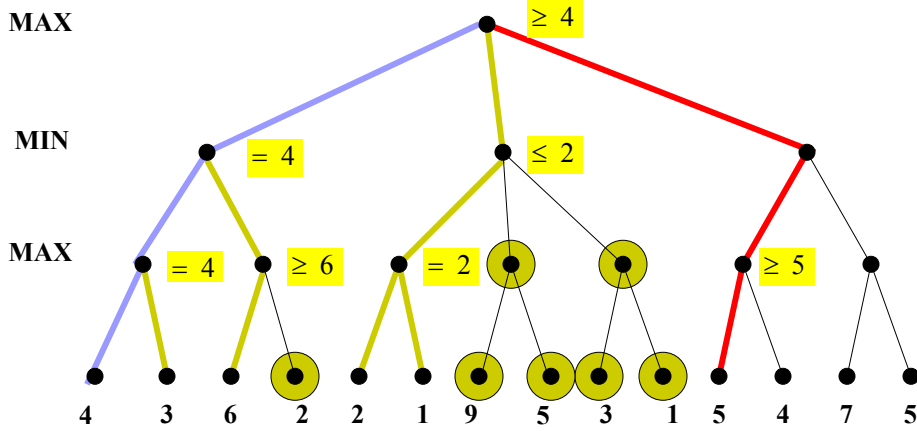
## Alpha beta pruning. Example



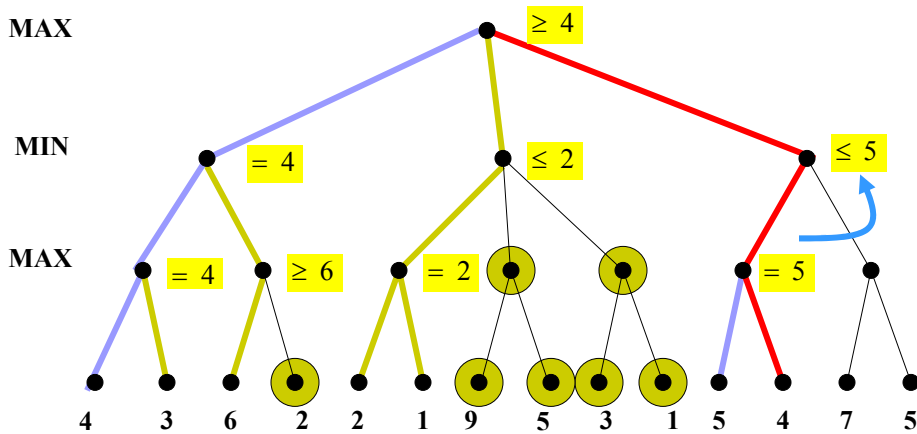
## Alpha beta pruning. Example



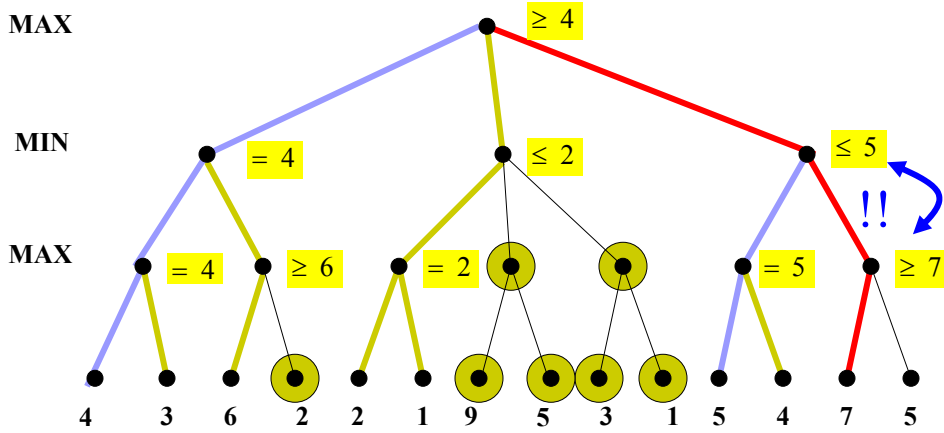
## Alpha beta pruning. Example



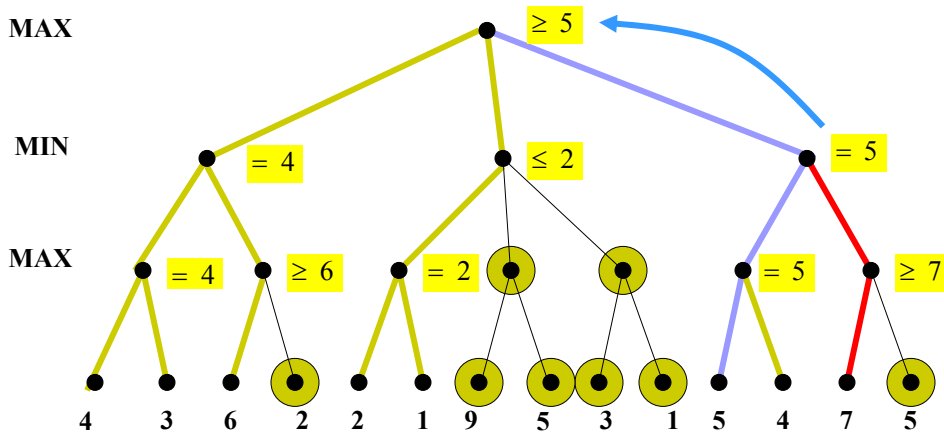
## Alpha beta pruning. Example



## Alpha beta pruning. Example



## Alpha beta pruning. Example



## Alpha beta pruning. Example

