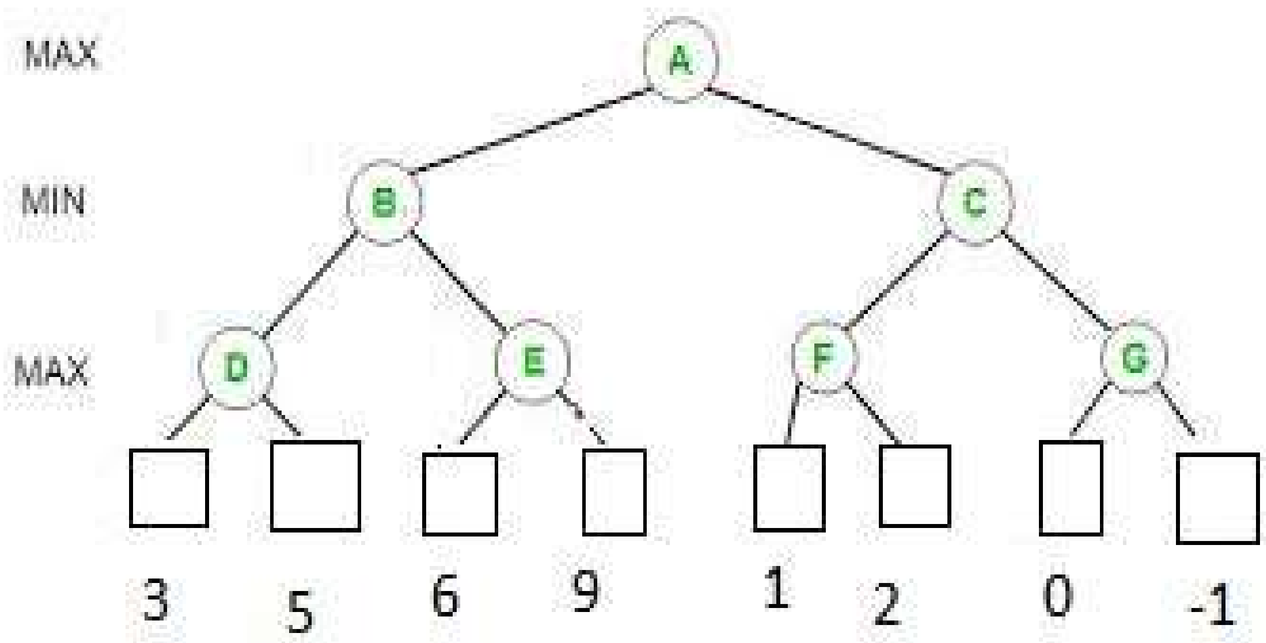
MONTE CARLO TREE SEARCH

RECAP



Rules for Alpha-beta Pruning

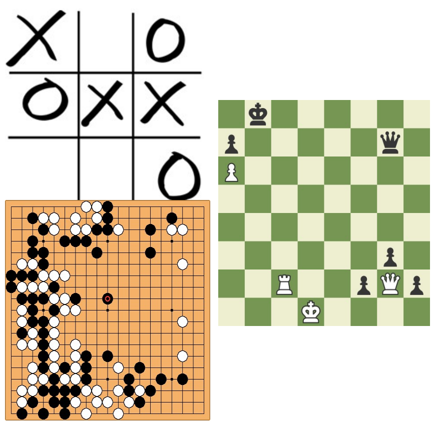
- **Alpha Pruning:** Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors.
- **Beta Pruning:** Search can be stopped below any MAX node having a alpha value greater than or equal to the beta value of any of its MIN ancestors.

PRE-MCTS ALGORITHMS

- Deterministic, Fully Observable Games
- “Perfect information”
- Can construct a tree that contains all possible outcomes because everything is fully determined

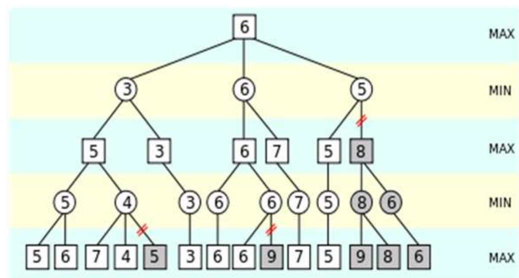
CONVENTIONAL GAME TREE SEARCH

Perfect-information games



- All aspects of the state are fully observable

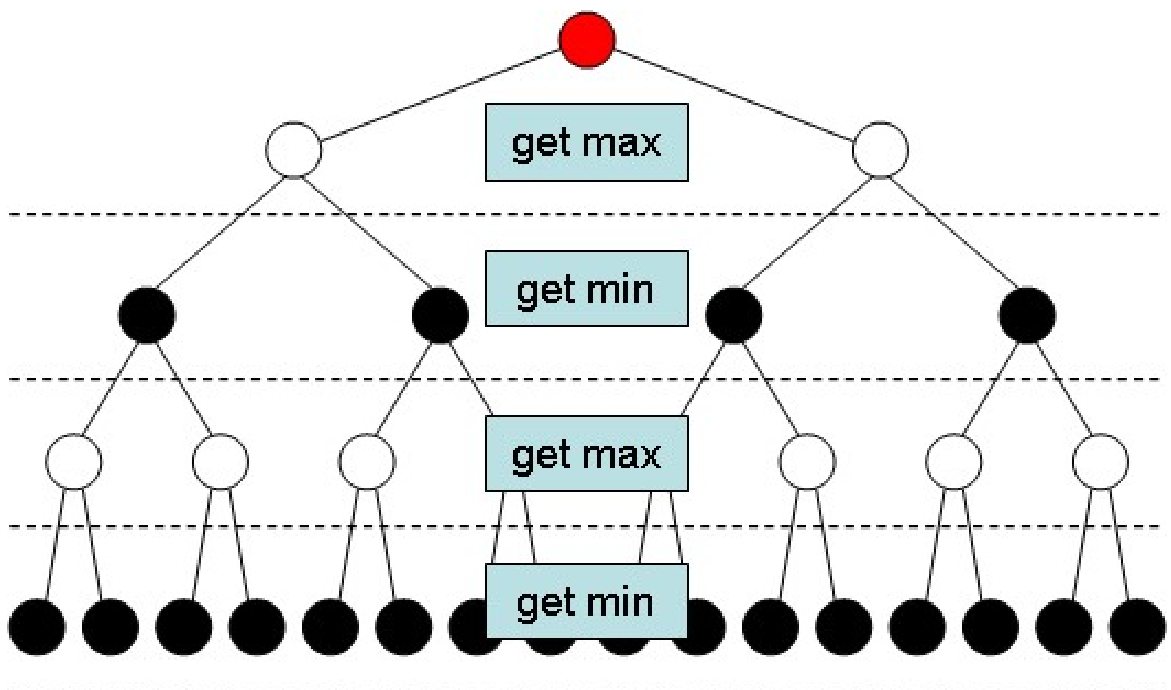
Minimax algorithm with alpha-beta pruning



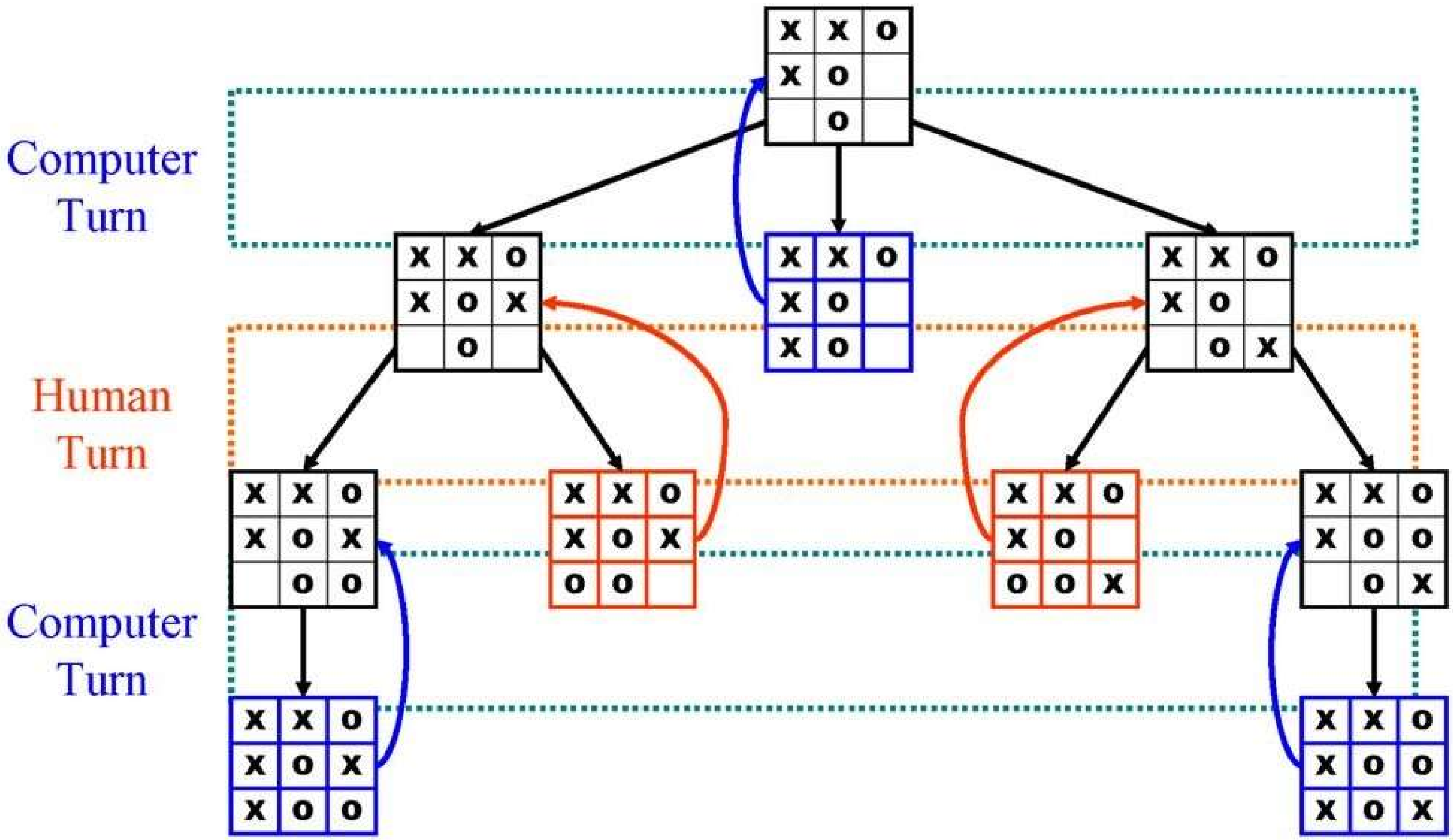
- Effective for
  - Modest branching factor
  - A good heuristic value

    function is known

MINIMIZE THE MAXIMUM POSSIBLE LOSS



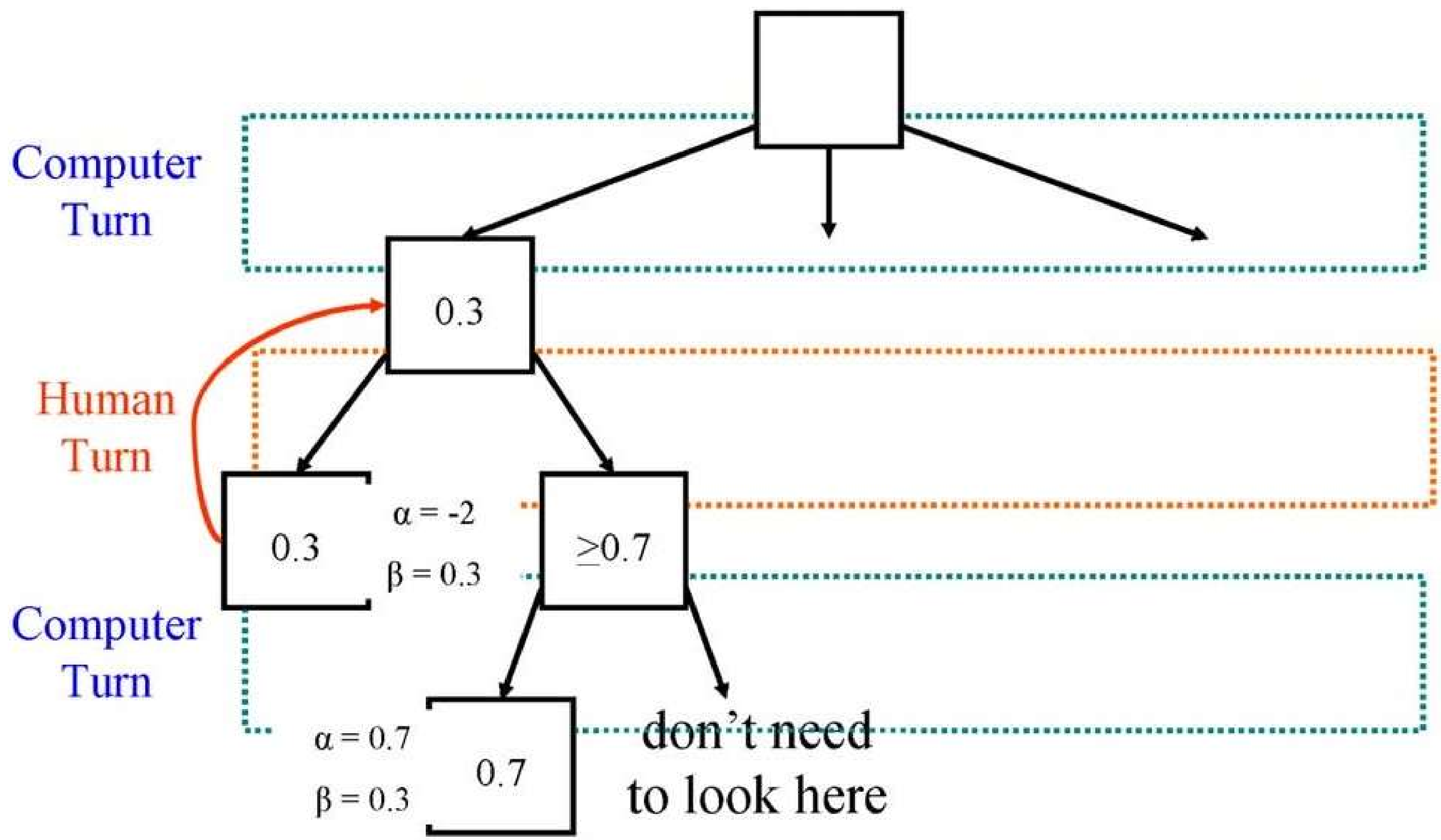
MINIMAX



ALPHA-BETA PRUNING

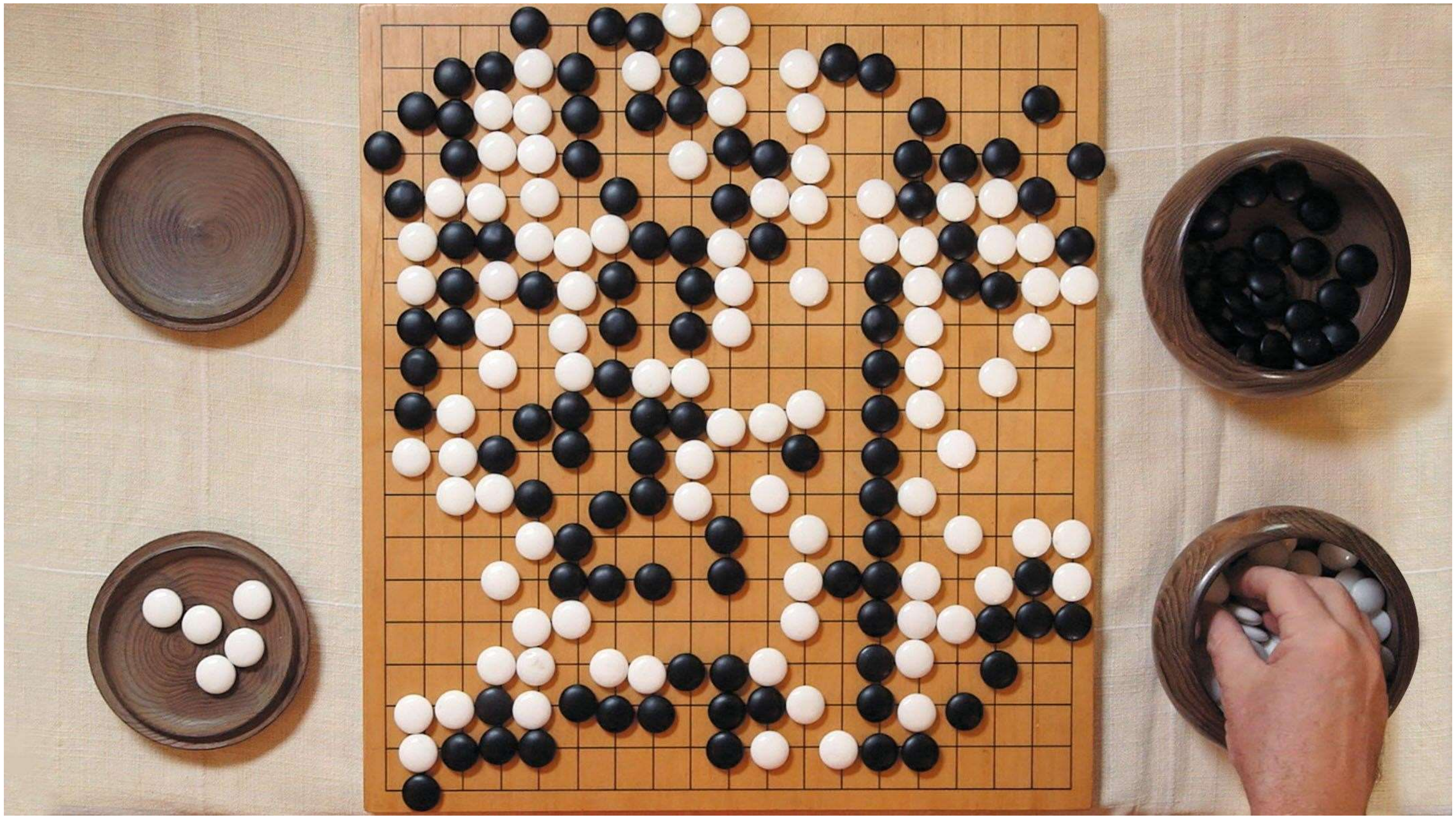
Prunes away branches that cannot influence the final decision

ALPHA - BETA

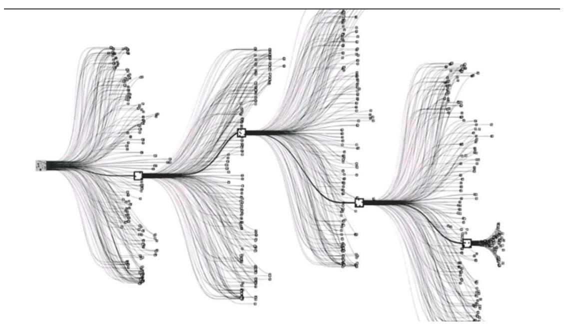


Computer Turn
0.3
Human Turn
0.3
α = -2
β = 0.3
≥0.7
Computer Turn
α = 0.7
β = 0.3
0.7
don't need to look here

2^4 VS. 2^{250}



GO



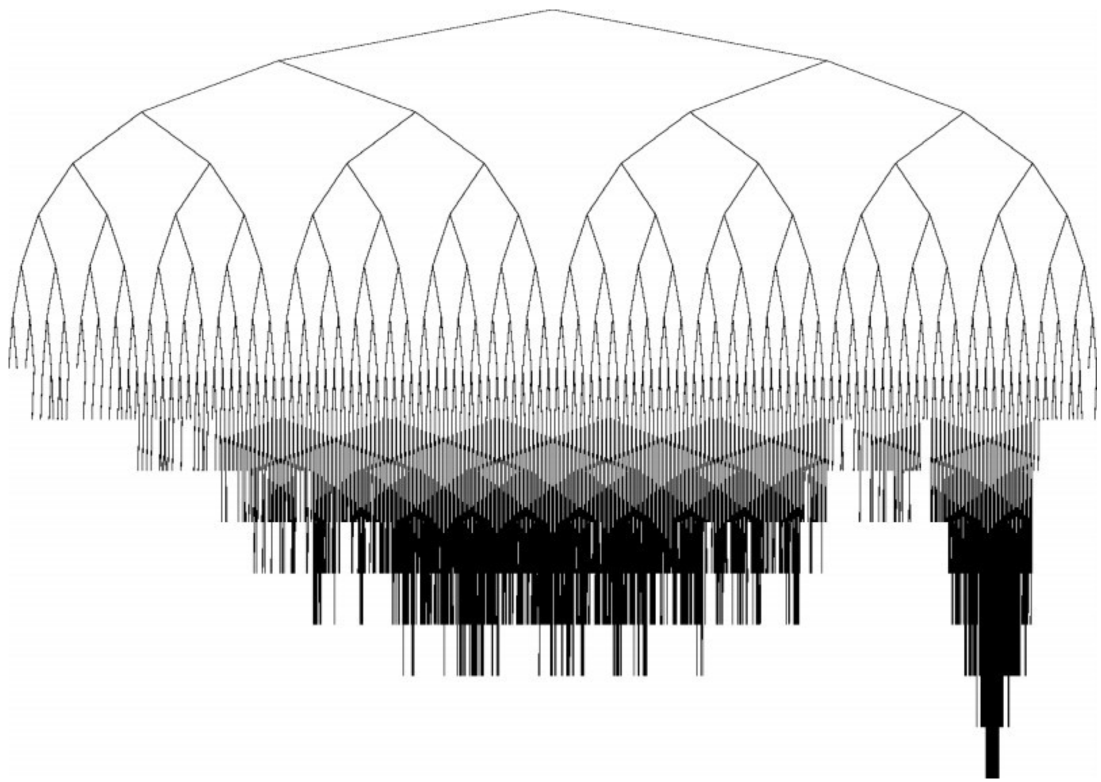
- Level weak to intermediate with alpha-beta
- Branching factor of Go is very large
  - 250 moves on average, game length > 200 moves
  - Order of magnitude greater than the branching factor of 20 for Chess
- Lack of a good evaluation function
  - Too subtle to model: similar looking positions can have completely different outcomes

MONTE CARLO!

- Revolutionized the world of computer Go
  - Application to deterministic games pretty recent (less than 10 years)
  - Explosion in interest, applications far beyond games
    - Planning, motion planning, optimization, finance, energy management

ALPHAGO AND MONTE CARLO TREE SEARCH

ASYMMETRIC TREE EXPLORATION

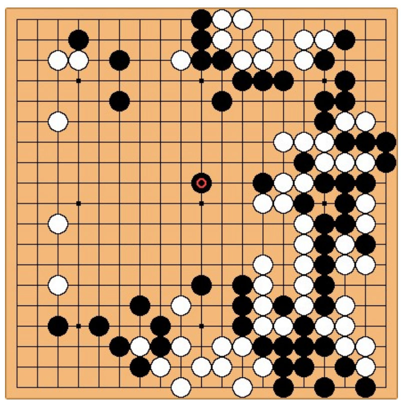


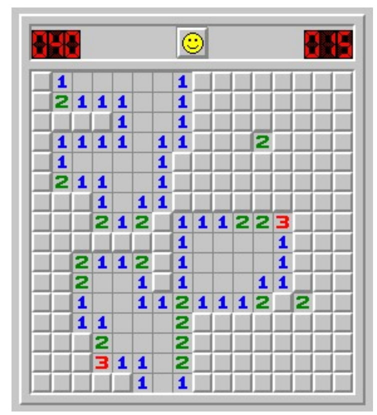
From *Bandit Algorithms for Tree Search*, Coquelin and Munos, 2007

MCTS FOR COMPUTER GO AND MINESWEEPER

Go: deterministic transitions

MineSweeper: probabilistic transitions





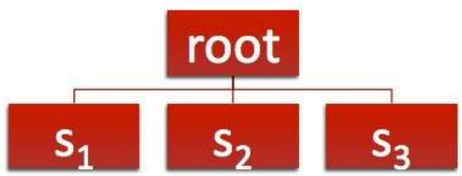
BASIC MONTE CARLO SIMULATION

- No evaluation function?

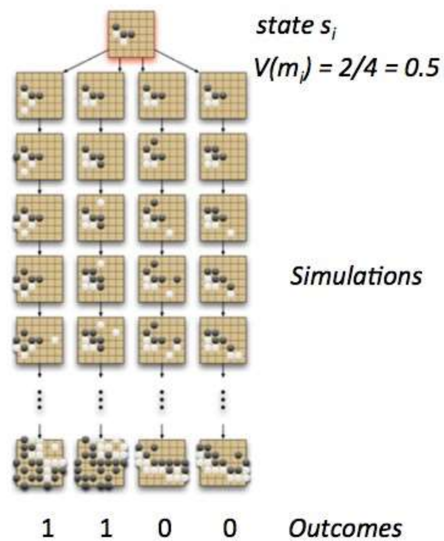
→ Simulate game using random moves
→ Score game at the end, keep winning statistics
→ Play move with best winning percentage
→ Repeat

Use this as the evaluation function, hopefully it will preserve *some* difference between a good position and a bad position

BASIC MONTE CARLO SEARCH



1 ply tree
root = current position
s_1 = state after move m_1
s_2 = ...

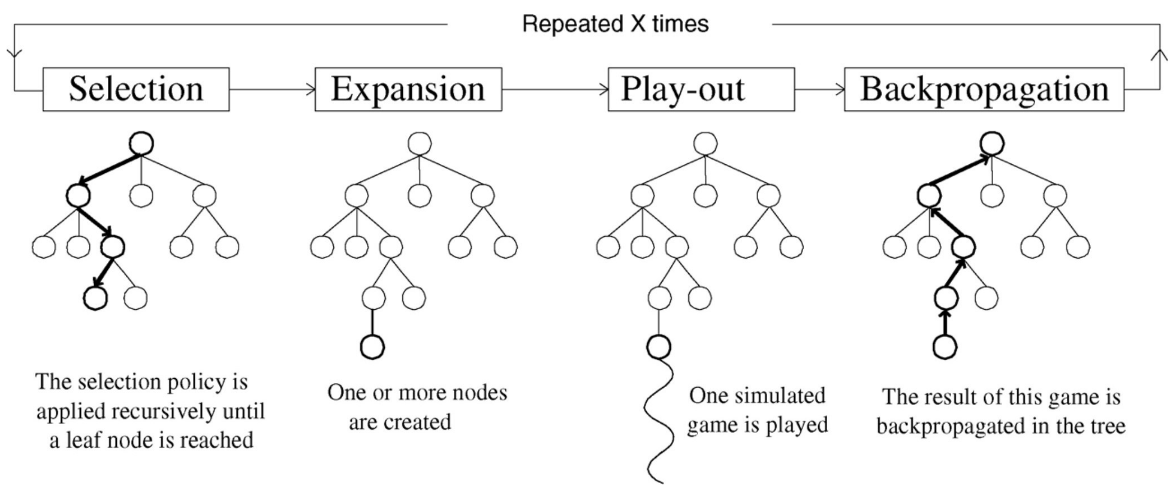


MONTE CARLO TREE SEARCH

- MCTS builds a *statistics tree* (detailing value of nodes) that partially maps onto the entire game tree
- Statistics tree guides the AI to focus on **most interesting nodes** in the game tree
- Value of nodes determined by simulations

MONTE CARLO TREE SEARCH

- Builds and searches an asymmetric game tree to make each move



Real-Time Monte Carlo Tree Search in Ms Pac-Man. Pepels et al. IEEE Transactions on Computational Intelligence and AI in Games (

MONTE CARLO TREE SEARCH

Use results of simulations to guide growth of the game tree

Exploitation: focus on promising moves

Exploration: focus on moves where uncertainty about evaluation is high

Seems like two contradictory goals

- Theory of **bandits** can help

MULTI-ARMED BANDIT PROBLEM

- Assumptions:
  - Choice of several arms
  - Each arm pull is independent of other pulls
  - Each arm has fixed, unknown average payoff
- Which arm has the best average payoff?

CONSIDER A ROW OF THREE SLOT MACHINES

A

P(A wins) = 60%

B

P(B wins) = 55%

C

P(C wins) = 40%

Each pull of an arm is either

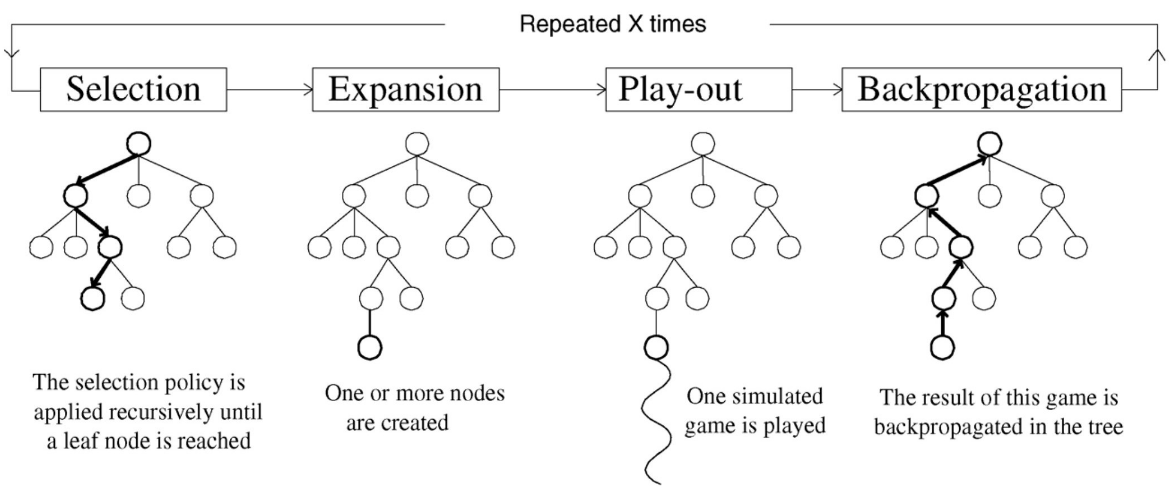
- A win(payoff 1) or
- A loss (payoff 0)
- A is the best arm → **but we don't know that**

EXPLORATION VS. EXPLOITATION

- Want to **explore** all arms
  - BUT, if we explore too much, may sacrifice reward we could have gotten
- Want to **exploit** promising arms more often
  - BUT, if we exploit too much, can get stuck with sub-optimal values

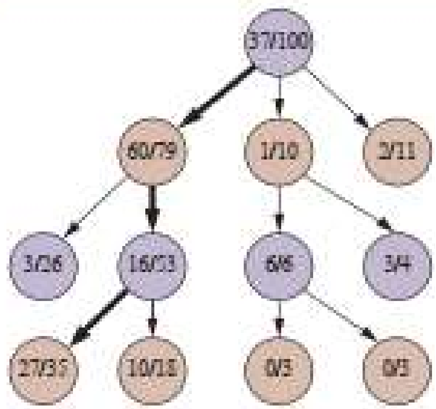
- Want to minimize regret = loss from playing non-optimal arm
- Need to balance between exploration and exploitation

MONTE CARLO TREE SEARCH



1) SELECTION

Selection policy is applied recursively until a leaf node is reached



(a) Selection

Figure 6.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b)

2) EXPANSION & SIMULATION

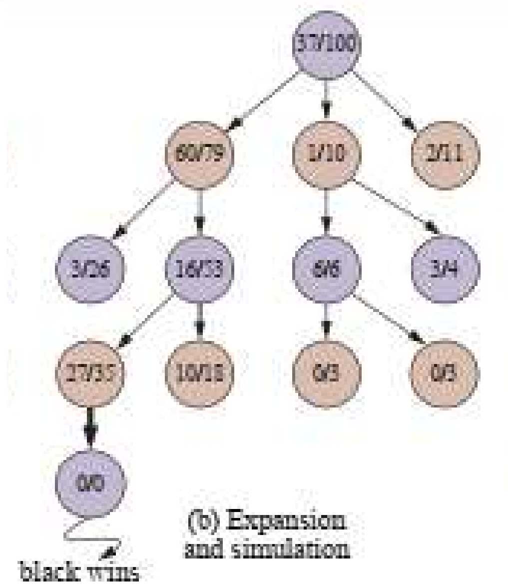
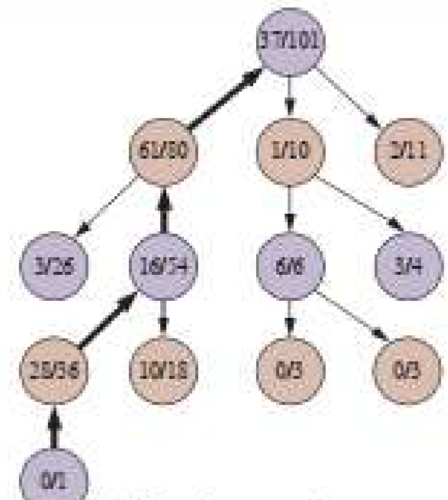


Figure 6.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b) we expand the selected node and do a simulation (playout), which ends in a win for black. In

4) BACKPROPAGATION

Result is backpropagated up the t



(c) Backpropagation

Figure 6.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b) we expand the selected node and do a simulation (playout), which ends in a win for black. In (c), the results of the simulation are back-propagated up the tree.

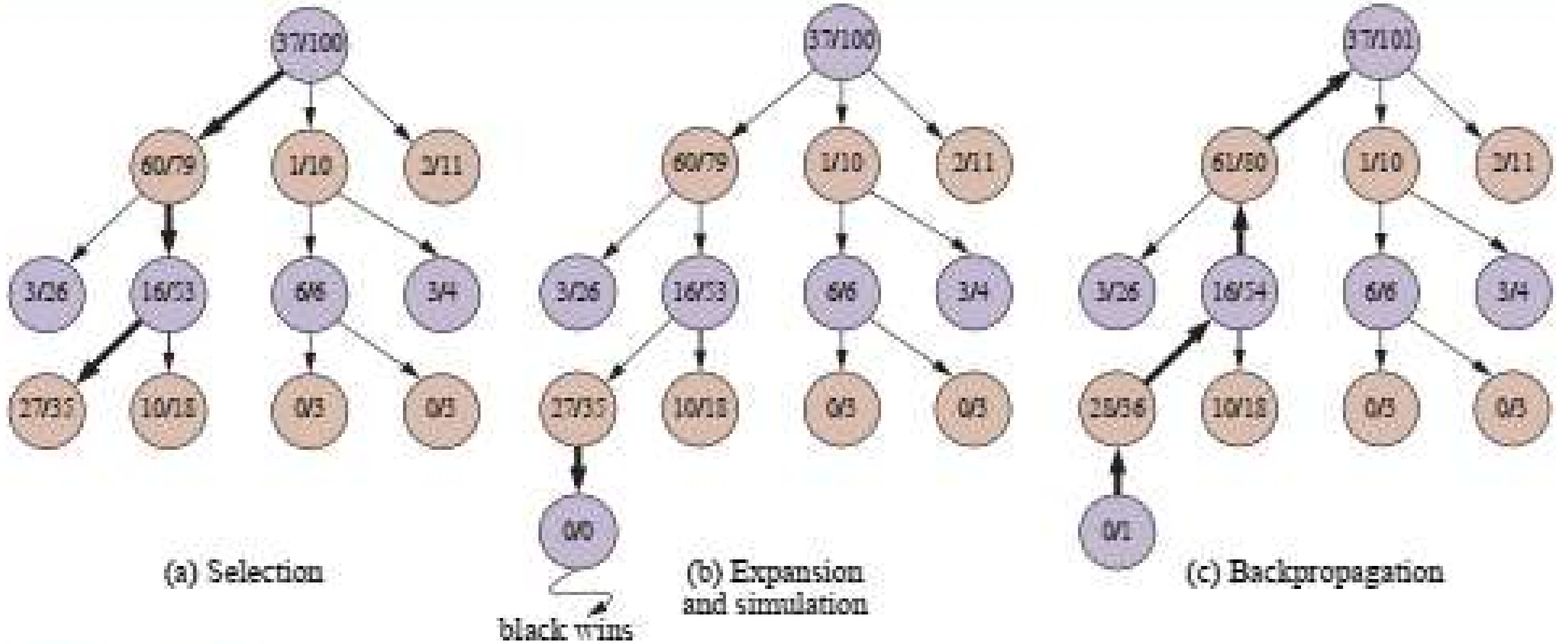
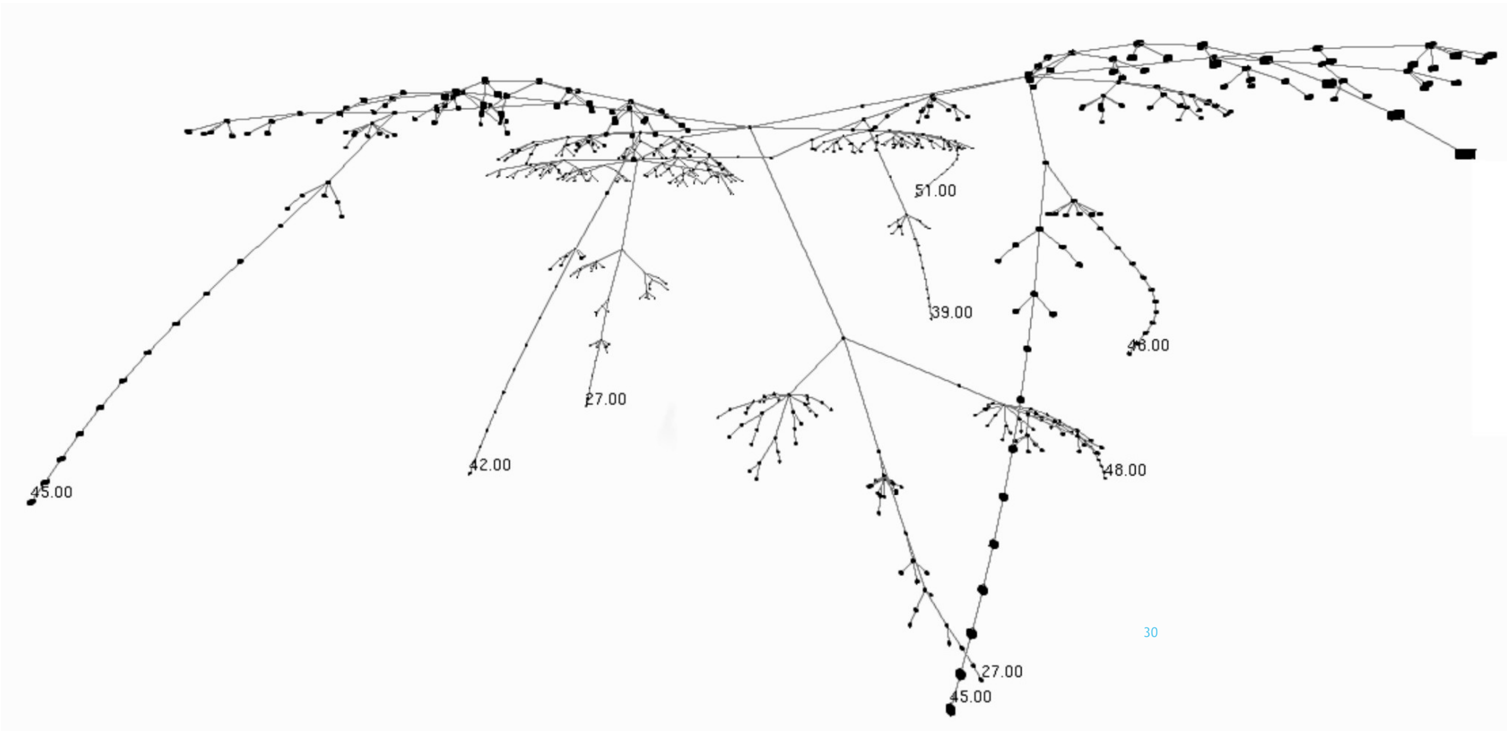


Figure 6.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b)

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

Figure 6.11 The Monte Carlo tree search algorithm. A game tree, *tree*, is initialized, and then we repeat a cycle of SELECT / EXPAND / SIMULATE / BACK-PROPAGATE until we run out of time, and return the move that led to the node with the highest number of playouts.

SAMPLE MCTS TREE



(fig from CadiaPlayer, Bjornsson and Finsson, IEE T-CIAIG 2009)

IMPACT – STRENGTHS OF MCTS

- Very general algorithm for decision making
- Works with very little domain-specific knowledge
  - Need simulator of the domain
- Can take advantage of knowledge when present
- Anytime algorithm
  - Can stop the algorithm and provide answer immediately, though improves answer with more time

IMPROVING SIMULATIONS

- Default roll-out policy is to make uniform random moves
- Goal is to find strong correlations between initial position and result of a simulation
- Game independent techniques
- If there is an immediate win, take it
- Last Good Reply
- Using prior knowledge

LAST GOOD REPLY

- Last Good Reply (Drake 2009), Last Good Reply with Forgetting (Baier et al 2010)
- Idea: after winning simulation, store (opponent move, our answer) move pairs
  - Try same reply in future simulations
  - Forgetting: delete move pair if it fails
- Evaluation: worked well for Go program with simpler playout policy
  - Trouble reproducing success with stronger Go programs
  - Simple form of adaptive simulations

USING PRIOR KNOWLEDGE

- (Silver 2009) machine-learned 3x3 pattern values
- Mogo and Fuego: hand-crafted features
  - Weights and interaction weights trained by Latent Feature Ranking
- AlphaGo
  - Neural networks trained over human expert games

Learn better knowledge

- E.g. patterns, features of a domain