Monte Carlo Tree Search & Stochastic Games

Outline

I. Monte Carlo tree search (MCTS)

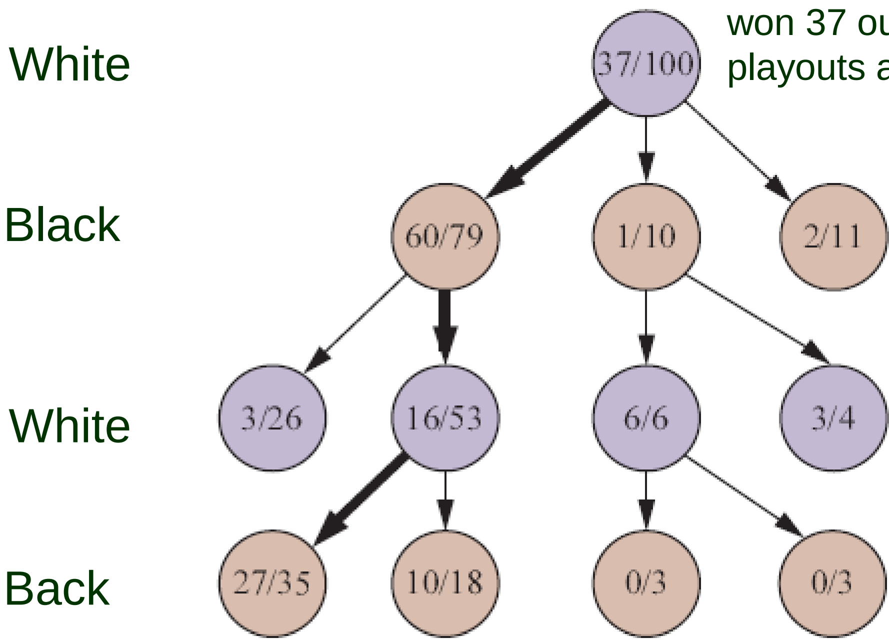
II. Stochastic games

* Figures/images are from the textbook site (or by the instructor).

I. An Iteration of MCTS – Step 1: Selection

Which move should Black make (at the root)?

Root: state just after the move by white, who has won 37 out of the 100 playouts at the node so far.



(a) Selection

- Unlike in the minimax game tree, an edge coming into a node now represents a move by the player represented by the node.

I. An Iteration of MCTS – Step 1: Selection

Which move should Black make (at the root)?

Root: state just after the move by white, who has won 37 out of the 100 playouts at the node so far.

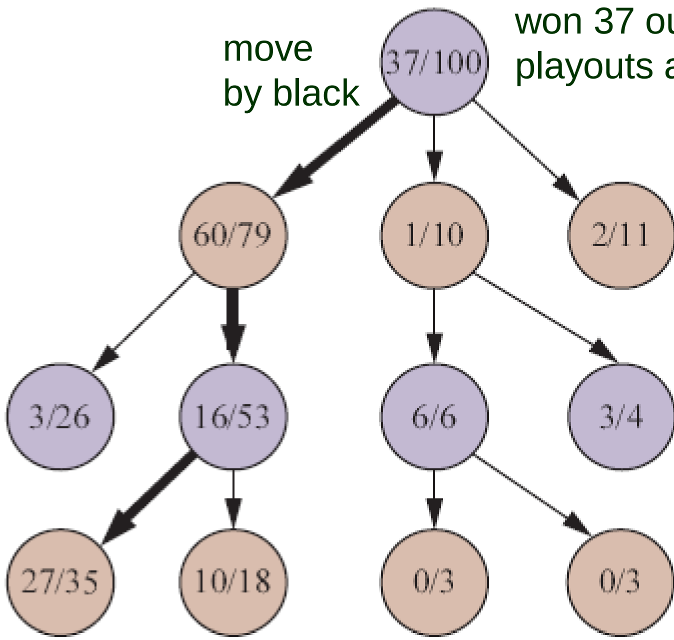
White

move
by black

Black

White

Back



(a) Selection

- Unlike in the minimax game tree, an edge coming into a node now represents a move by the player represented by the node.

I. An Iteration of MCTS – Step 1: Selection

Which move should Black make (at the root)?

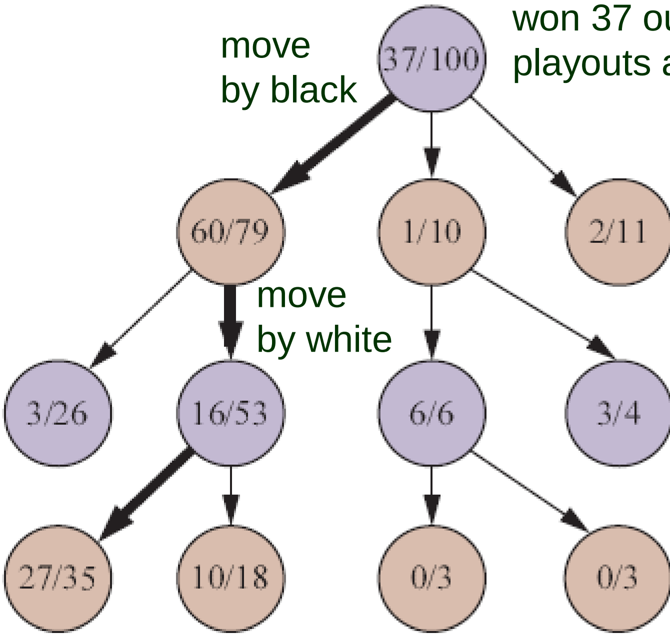
Root: state just after the move by white, who has won 37 out of the 100 playouts at the node so far.

White

Black

White

Back

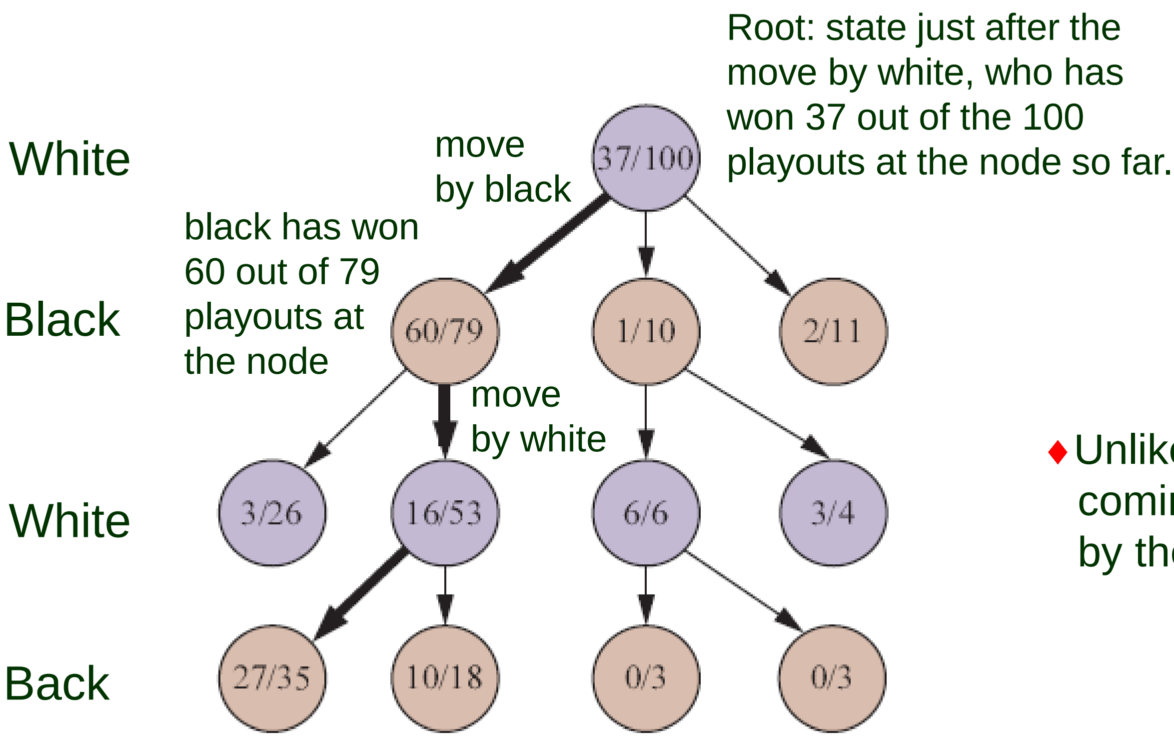


(a) Selection

- Unlike in the minimax game tree, an edge coming into a node now represents a move by the player represented by the node.

I. An Iteration of MCTS – Step 1: Selection

Which move should Black make (at the root)?



(a) Selection

- ♦ Unlike in the minimax game tree, an edge coming into a node now represents a move by the player represented by the node.

I. An Iteration of MCTS – Step 1: Selection

Which move should Black make (at the root)?

Root: state just after the move by white, who has won 37 out of the 100 playouts at the node so far.

White

move by black

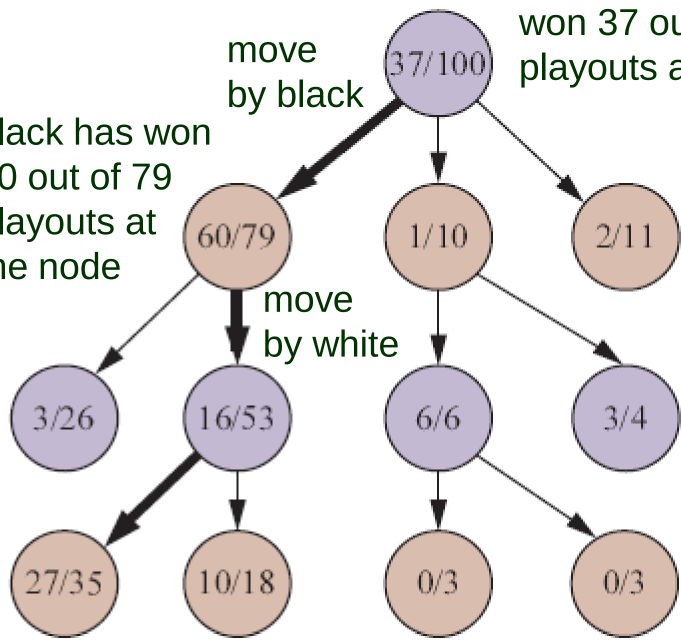
black has won 60 out of 79 playouts at the node

Black

move by white

White

Back

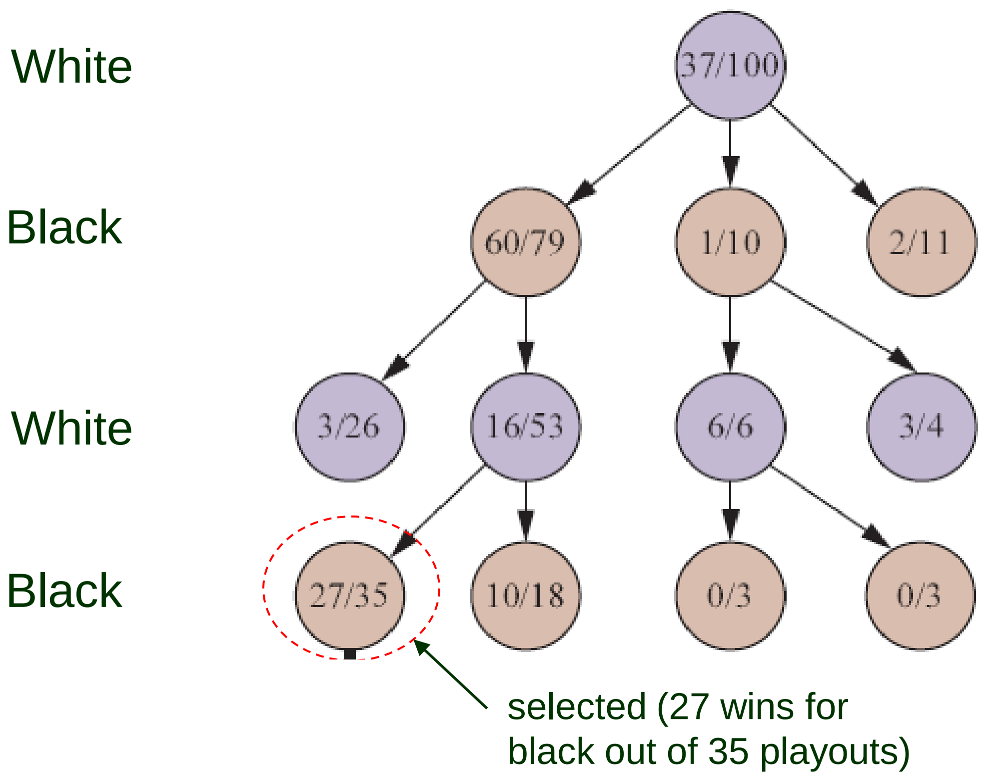


(a) Selection

Also reasonable to select for the purpose of exploration

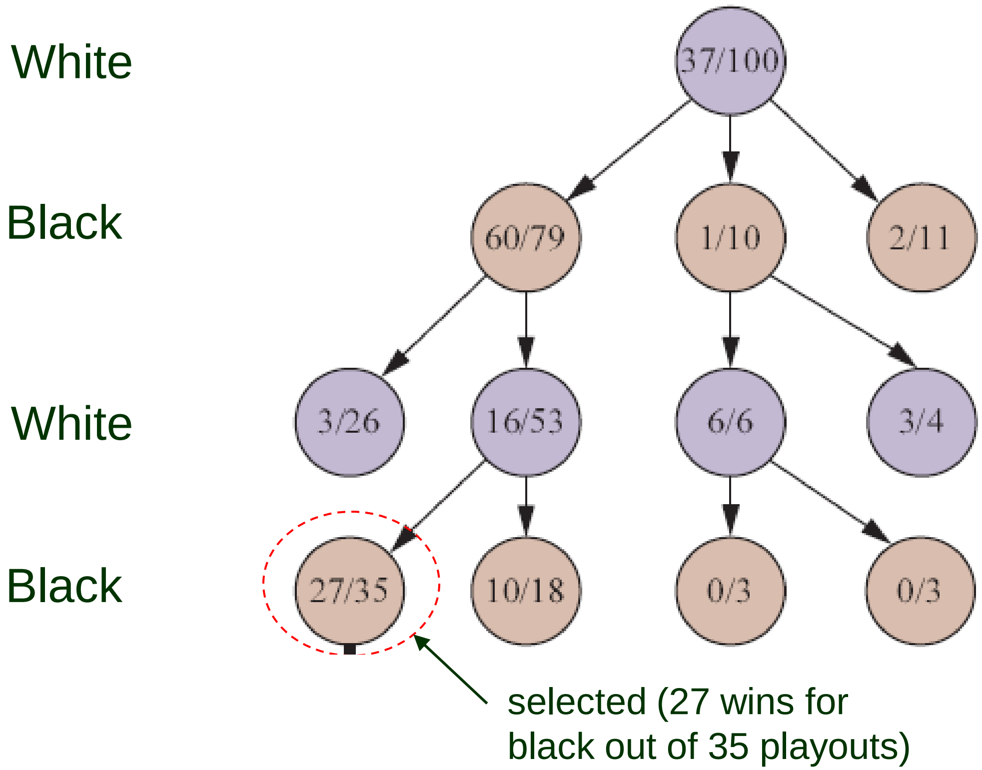
- Unlike in the minimax game tree, an edge coming into a node now represents a move by the player represented by the node.

Steps 2 & 3: Expansion & Simulation



(b) Expansion and simulation

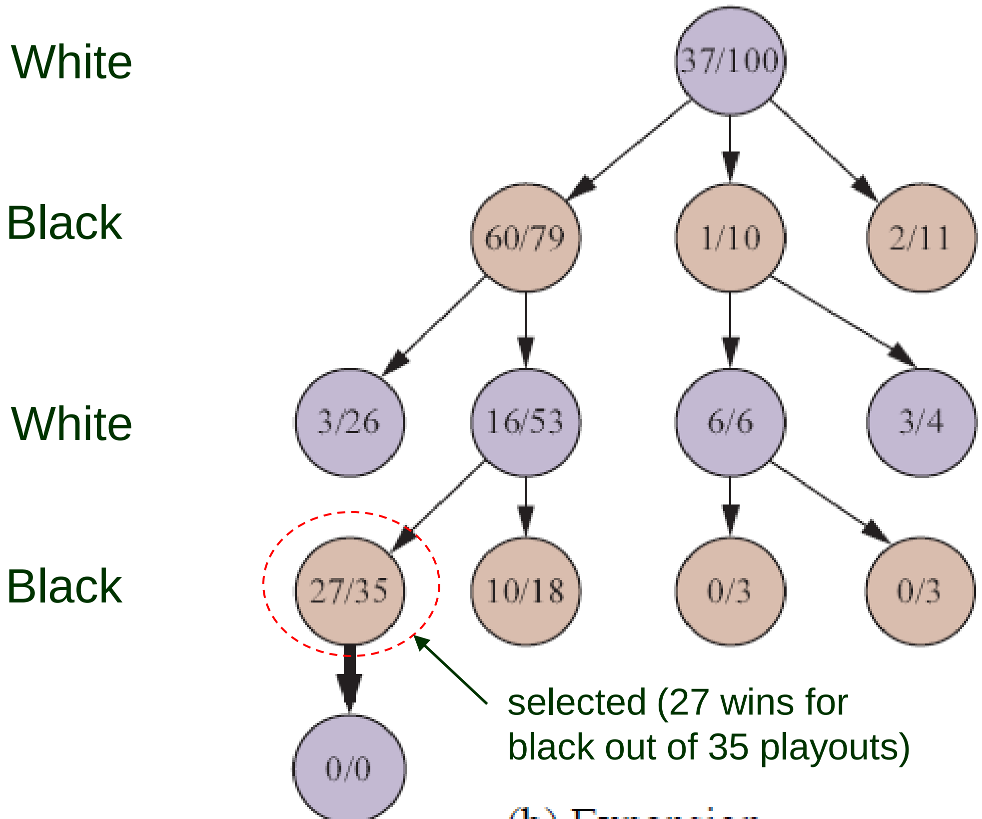
Steps 2 & 3: Expansion & Simulation



(b) Expansion and simulation

- Generate a new child of the selected node.

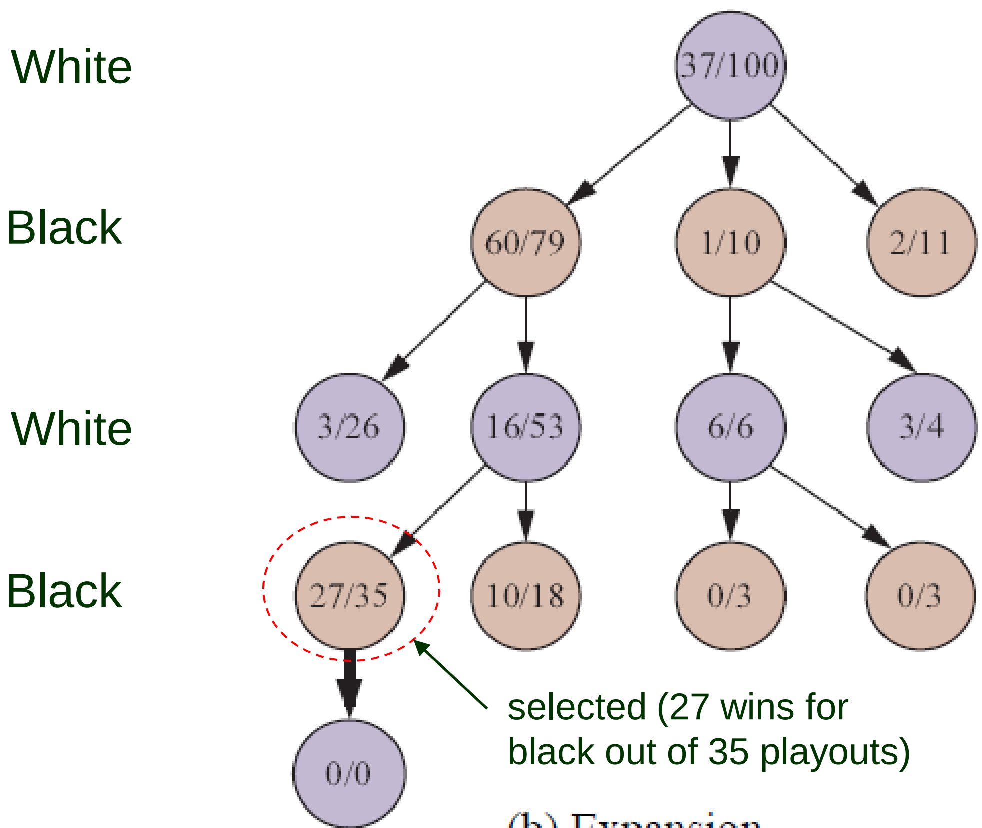
Steps 2 & 3: Expansion & Simulation



(b) Expansion and simulation

- Generate a new child of the selected node.

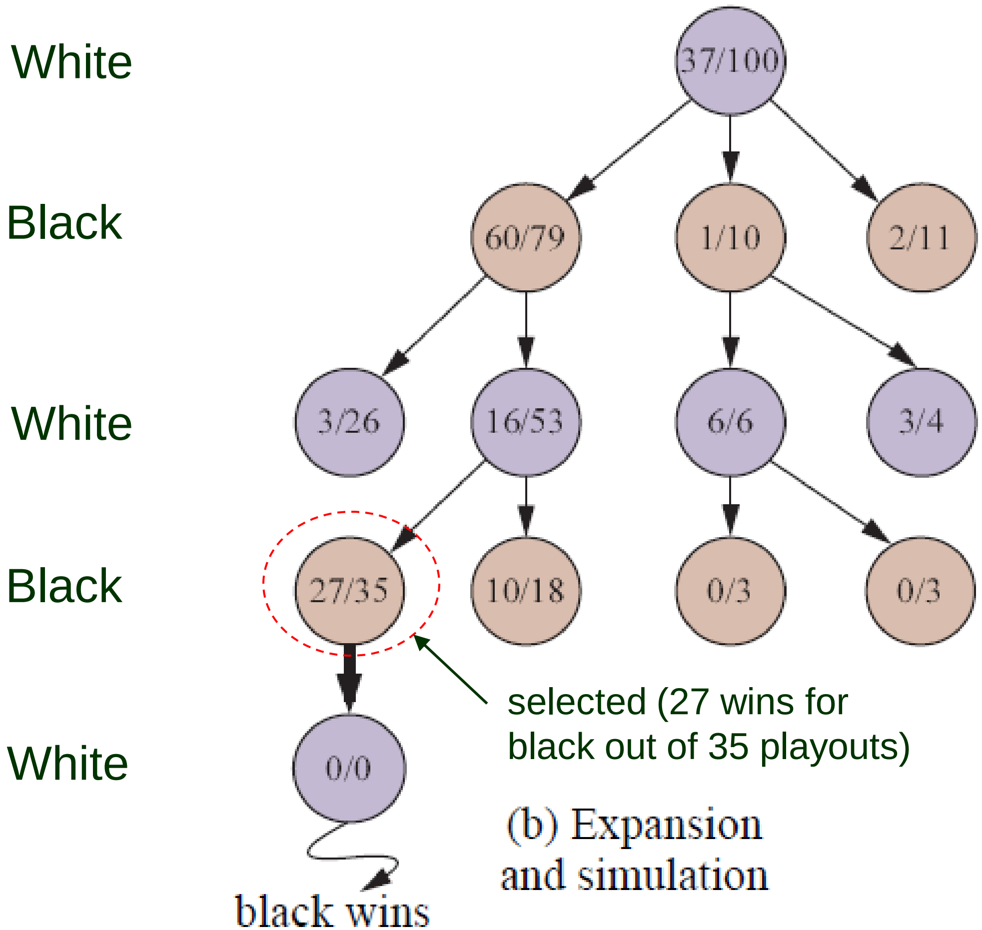
Steps 2 & 3: Expansion & Simulation



(b) Expansion and simulation

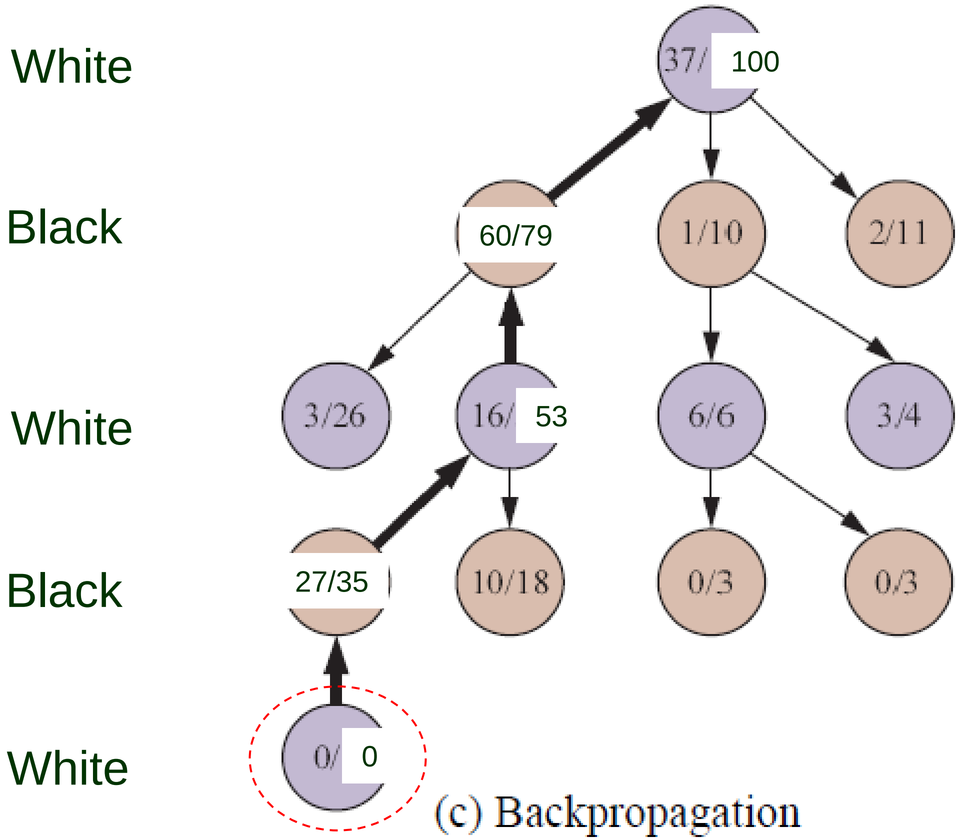
- Generate a new child of the selected node.
- Perform a playout from the newly generated child node.

Steps 2 & 3: Expansion & Simulation



- Generate a new child of the selected node.
- Perform a playout from the newly generated child node.

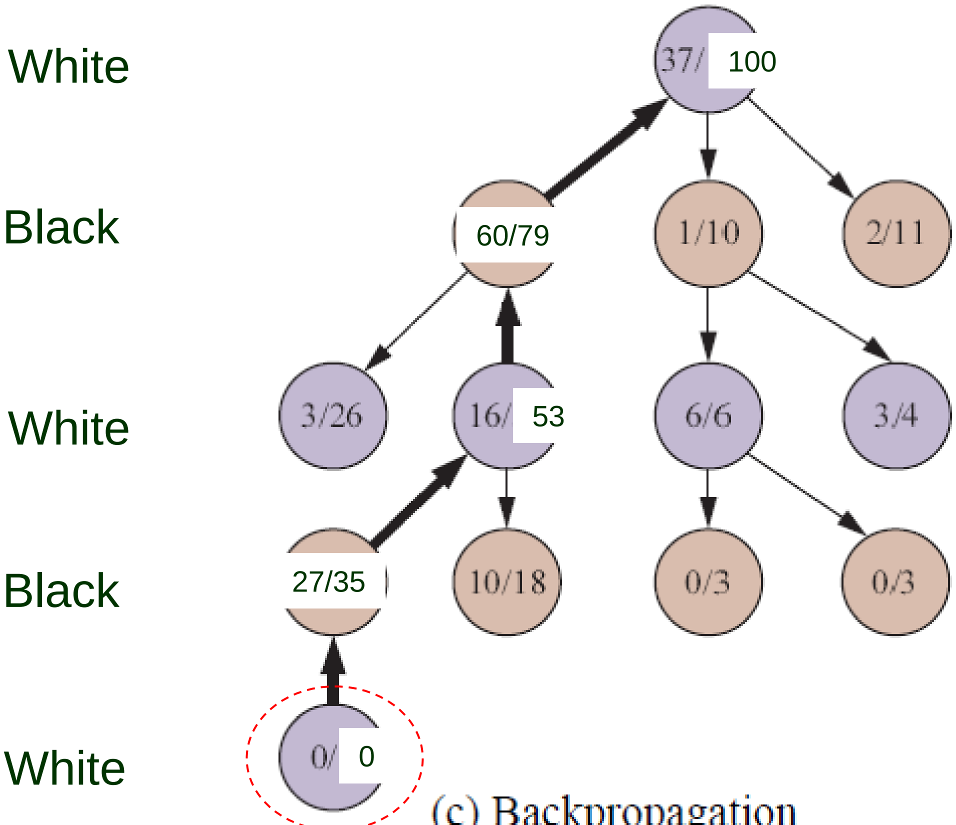
Step 4: Back Propagation



(c) Backpropagation

- Update all the nodes upward along the path until the root.

Step 4: Back Propagation

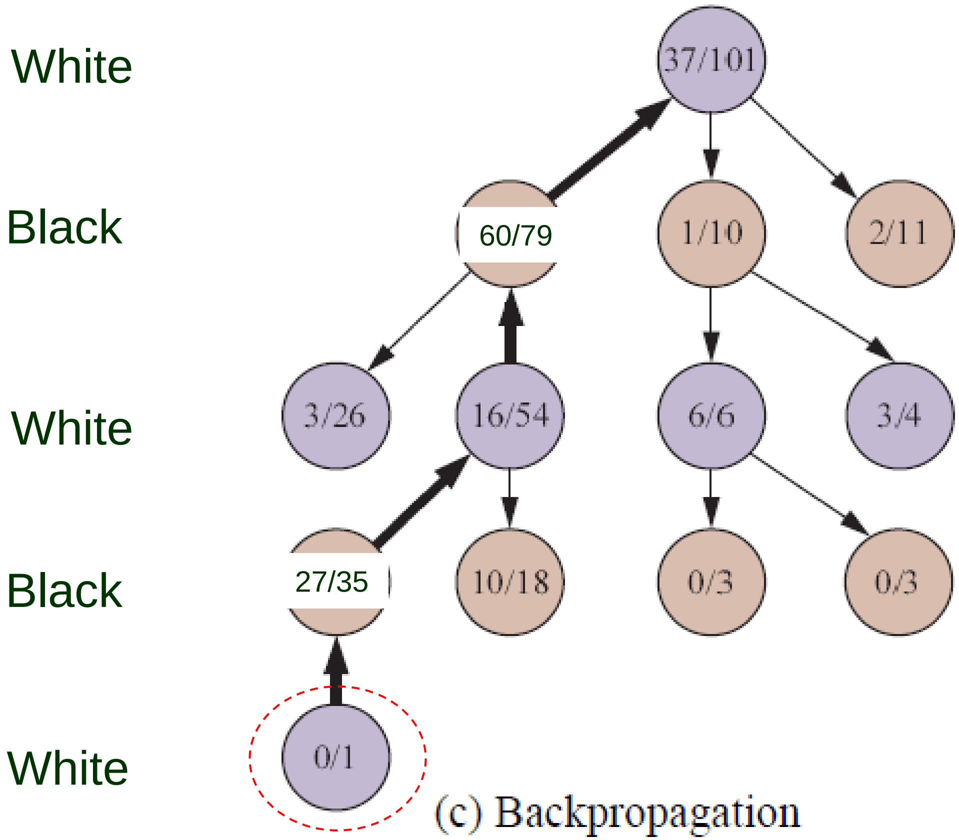


(c) Backpropagation

- Update all the nodes upward along the path until the root.

Black wins this playout:

Step 4: Back Propagation



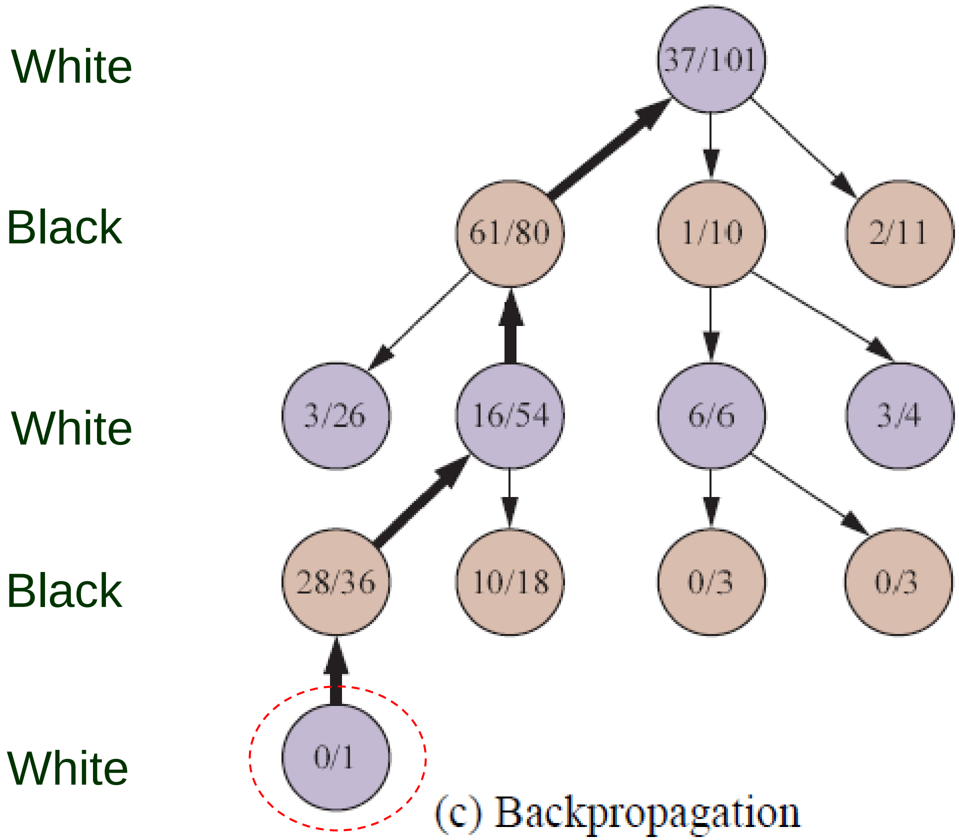
(c) Backpropagation

- Update all the nodes upward along the path until the root.

Black wins this playout:

- ♦ At a white node, increment #playouts only.

Step 4: Back Propagation



(c) Backpropagation

- Update all the nodes upward along the path until the root.

Black wins this playout:

- ♦ At a white node, increment #playouts only.
- ♦ At a black node, increment #wins and #playouts.

Termination

- MCTS repeats the four steps (selection, expansion, simulation, back-propagation) in order until
  - a set number N of iterations have been performed, or
  - the allotted time has expired.
- It returns the move with the highest number of playouts.

Termination

- MCTS repeats the four steps (selection, expansion, simulation, back-propagation) in order until
  - a set number N of iterations have been performed, or
  - the allotted time has expired.
- It returns the move with the highest number of playouts.

Why not the highest ratio?

Termination

- MCTS repeats the four steps (selection, expansion, simulation, back-propagation) in order until
  - a set number N of iterations have been performed, or
  - the allotted time has expired.
- It returns the move with the highest number of playouts.

Why not the highest ratio?

- Since better moves are more likely to be chosen, the most promising move is expected to have the highest number of playouts.

Termination

- MCTS repeats the four steps (selection, expansion, simulation, back-propagation) in order until

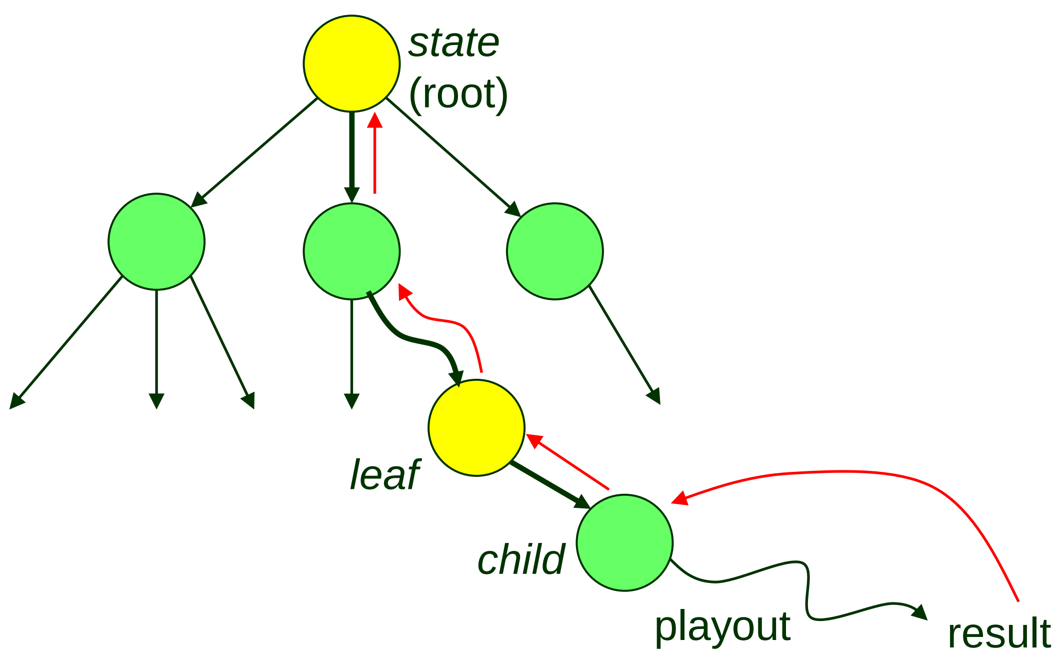
  - a set number N of iterations have been performed, or
  - the allotted time has expired.

- It returns the move with the $\underbrace{\text{highest number of playouts}}_{\text{Why not the highest ratio?}}$.

  - Since better moves are more likely to be chosen, the most promising move is expected to have the highest number of playouts.
  - A node with 65/100 wins is better than one with 2/3 wins (which has a lot of uncertainty).

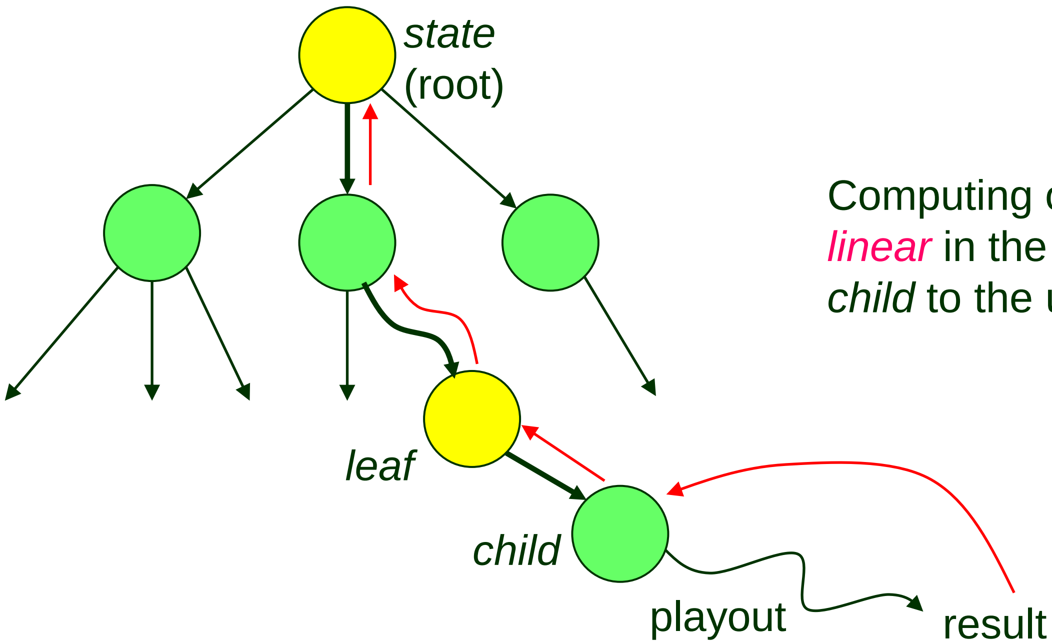
Monte Carlo Tree Search Algorithm

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action  // decide a move at state.
  tree ← NODE(state)    // initialize the tree with state at the root
  while IS-TIME-REMAINING() do  // each iteration expands the tree by one node.
    leaf ← SELECT(tree)               // the node to be expanded must be a leaf.
    child ← EXPAND(leaf)              // tree is expanded to the node child as a child of leaf.
    result ← SIMULATE(child)          // playout: moves are not recorded in the three.
    BACK-PROPAGATE(result, child)  // update nodes on the path upward to the root.
  return the move in ACTIONS(state) whose node has highest number of playouts
```



Monte Carlo Tree Search Algorithm

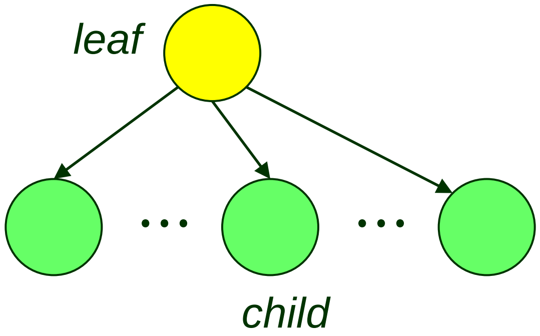
function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action* // decide a move at *state.*
    tree ← NODE(*state*) // initialize the tree with *state* at the root
    while IS-TIME-REMAINING() **do** // each iteration expands the tree by one node.
        leaf ← SELECT(*tree*) // the node to be expanded must be a leaf.
        child ← EXPAND(*leaf*) // tree is expanded to the node *child* as a child of *leaf.*
        result ← SIMULATE(*child*) // playout: moves are not recorded in the three.
        BACK-PROPAGATE(*result*, *child*) // update nodes on the path upward to the root.
    return the move in ACTIONS(*state*) whose node has highest number of playouts



Computing one playout takes time *linear* in the length of the path from *child* to the utility node (result).

Three Issues

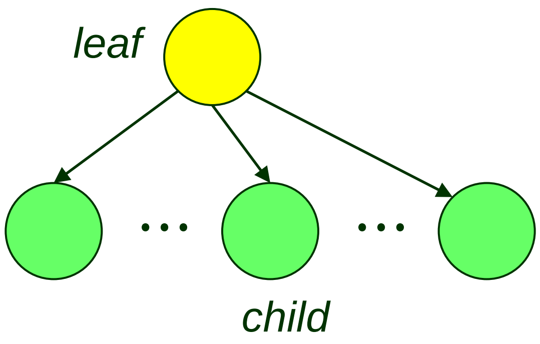
♦ *child* ← EXPAND(*leaf*)



Create one or more child nodes and choose node *child* from one of them.

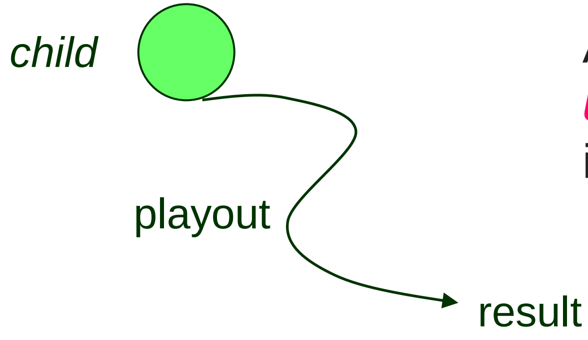
Three Issues

- *child* ← EXPAND(*leaf*)



Create one or more child nodes and choose node *child* from one of them.

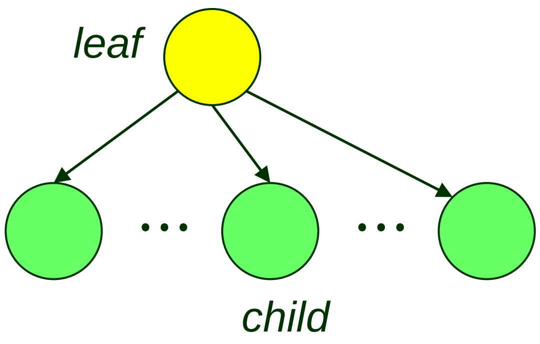
- *result* ← SIMULATE(*child*)



A playout may be as simple as choosing *uniformly random* moves until the game is decided.

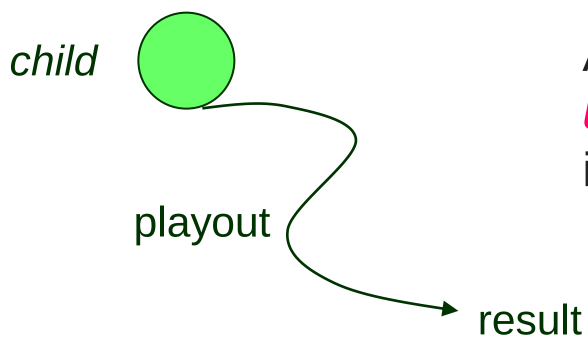
Three Issues

- *child* ← EXPAND(*leaf*)



Create one or more child nodes and choose node *child* from one of them.

- *result* ← SIMULATE(*child*)



A playout may be as simple as choosing *uniformly random* moves until the game is decided.

- IS-TIME-REMAINING()

Pure Monte Carlo search does *N* simulations instead.

Ranking of Possible Moves

Upper confidence bound formula:

$$\mathrm{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player
making a move at n
out of all playouts
through the node

$$\mathrm{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

$$\text{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

#playouts through the node

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player
making a move at n
out of all playouts
through the node

$$\text{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log \boxed{N(PARENT(n))}}{N(n)}}$$

#playouts through
the node

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

#playouts through the parent of n

$$\text{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log \boxed{N(PARENT(n))}}{N(n)}}$$

#playouts through the node

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

#playouts through the parent of n

$$\mathrm{UCB}(n) = \boxed{\frac{U(n)}{N(n)}} + C \times \sqrt{\frac{\log \boxed{N(PARENT(n))}}{N(n)}}$$

#playouts through the node

Exploitation term:
average utility of n

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

#playouts through the parent of n

$$\text{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log \boxed{N(PARENT(n))}}{N(n)}}$$

#playouts through the node

Exploitation term: average utility of n

Exploration term: high value if n has been explored a few times.

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

#playouts through the parent of n

$$\mathrm{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log \boxed{N(PARENT(n))}}{N(n)}}$$

#playouts through the node

balance between exploitation and exploration

Exploitation term: average utility of n

Exploration term: high value if n has been explored a few times.

Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

#playouts through the parent of n

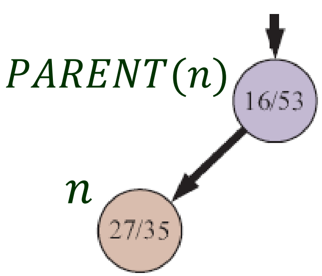
$$\mathrm{UCB}(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

#playouts through the node

balance between exploitation and exploration

Exploitation term: average utility of n

Exploration term: high value if n has been explored a few times.



Ranking of Possible Moves

Upper confidence bound formula:

#wins for the player making a move at n out of all playouts through the node

#playouts through the parent of n

$$\mathrm{UCB}(n) = \boxed{\frac{U(n)}{N(n)}} + C \times \boxed{\sqrt{\frac{\log \boxed{N(PARENT(n))}}{N(n)}}}$$

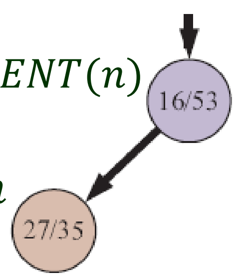
#playouts through the node

balance between exploitation and exploration

Exploitation term: average utility of n

Exploration term: high value if n has been explored a few times.

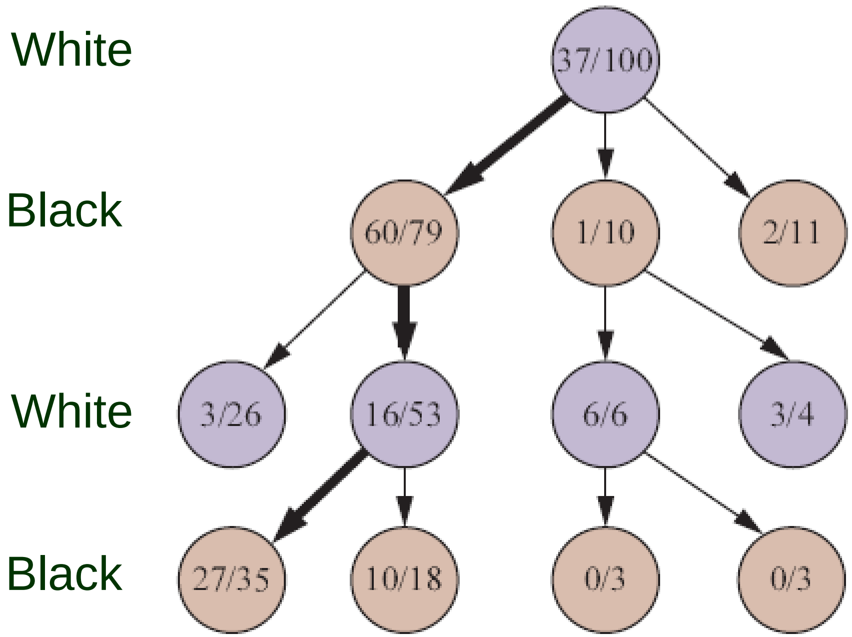
$PARENT(n)$



n

$$U(n) = 27$$
$$N(n) = 35$$
$$N\big(PARENT(n)\big) = 53$$
$$C = \sqrt{2}$$ (choice by a theoretical argument)

Constant C



(a) Selection

- Balances exploitation and exploration.
- Multiple values are tried and the one that performs the best is chosen.
  - $C = 1.4$

    The 60/79 node has the highest score.
  - $C = 1.5$

    The 2/11 node has the highest score.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

  E.g., branching factor b = 32,
  100 plies on average for a game.
  enough computing power to consider 10^9 states

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

  E.g., branching factor b = 32,
  100 plies on average for a game.
  enough computing power to consider 10^9 states

  Minimax can search 6 ply deep: $32^6 \approx 10^9$.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

E.g., branching factor b = 32,
100 plies on average for a game.
enough computing power to consider 10^9 states

Minimax can search 6 ply deep: $32^6 \approx 10^9$.

Alpha-beta can search up to 12 plies.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

E.g., branching factor b = 32,
100 plies on average for a game.
enough computing power to consider 10^9 states

Minimax can search 6 ply deep: $32^6 \approx 10^9$.

Alpha-beta can search up to 12 plies.

Monte Carlo can do 10^7 playouts.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

  E.g., branching factor b = 32,
  100 plies on average for a game.
  enough computing power to consider 10^9 states

  Minimax can search 6 ply deep: $32^6 \approx 10^9$.

  Alpha-beta can search up to 12 plies.

  Monte Carlo can do 10^7 playouts.

- MCTS has advantage over alpha-beta when b is high.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

  E.g., branching factor b = 32,
  100 plies on average for a game.
  enough computing power to consider 10^9 states

  Minimax can search 6 ply deep: $32^6 \approx 10^9$.

  Alpha-beta can search up to 12 plies.

  Monte Carlo can do 10^7 playouts.

- MCTS has advantage over alpha-beta when b is high.

- MCTS is less vulnerable to a single error.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

  E.g., branching factor b = 32,
  100 plies on average for a game.
  enough computing power to consider 10^9 states

  Minimax can search 6 ply deep: $32^6 \approx 10^9$.

  Alpha-beta can search up to 12 plies.

  Monte Carlo can do 10^7 playouts.

- MCTS has advantage over alpha-beta when b is high.
- MCTS is less vulnerable to a single error.
- MCTS can be applied to brand-new games via training by self-play.

More on MCTS

- Time to compute a playout is linear in the depth of the game tree. Compute many playouts before taking one move.

  E.g., branching factor b = 32,
  100 plies on average for a game.
  enough computing power to consider 10^9 states

  Minimax can search 6 ply deep: $32^6 \approx 10^9$.

  Alpha-beta can search up to 12 plies.

  Monte Carlo can do 10^7 playouts.

- MCTS has advantage over alpha-beta when b is high.

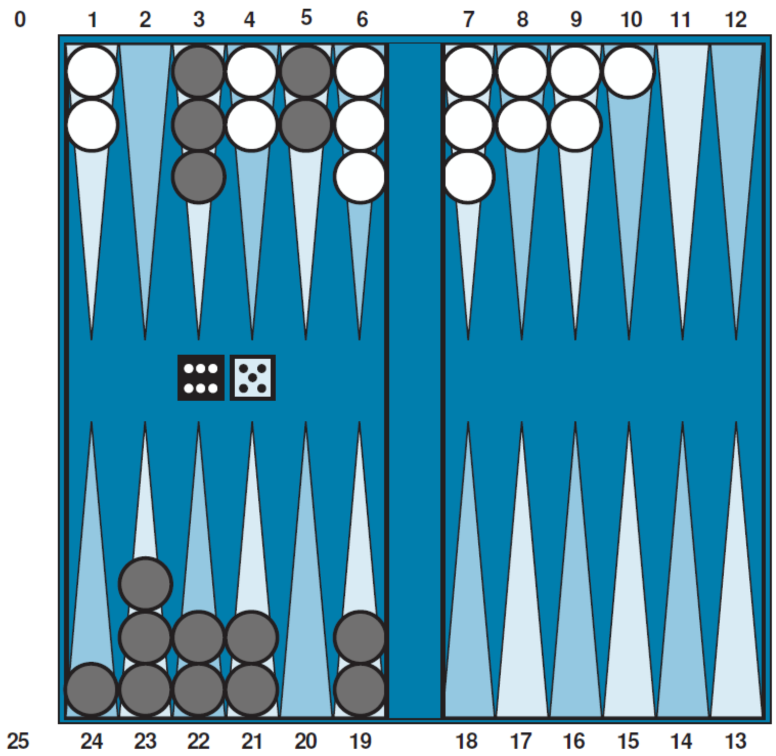
- MCTS is less vulnerable to a single error.

- MCTS can be applied to brand-new games via training by self-play.

- MCTS is less desired than alpha-beta on a game like chess with low b and good evaluation function.

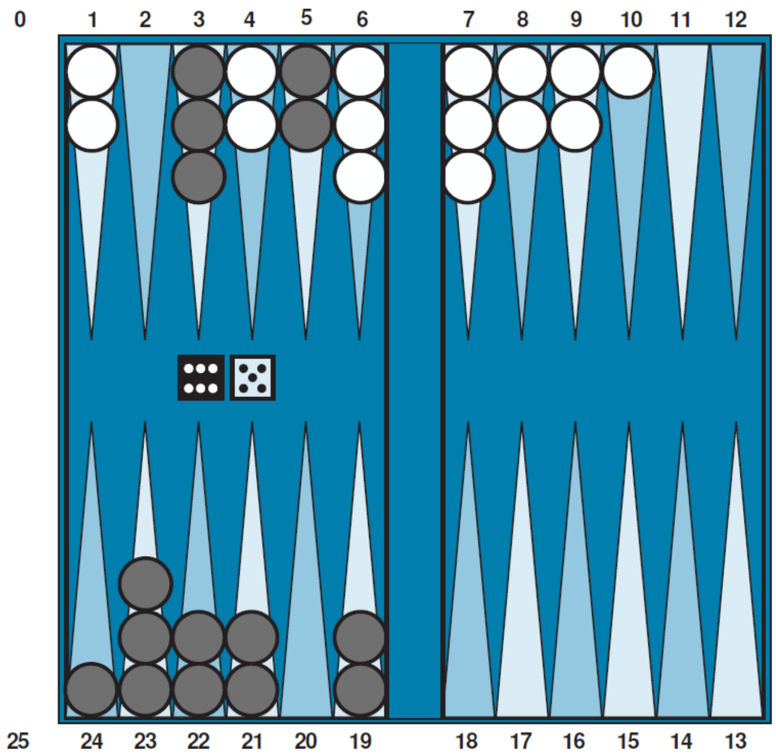
II. Stochastic Games

Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.



II. Stochastic Games

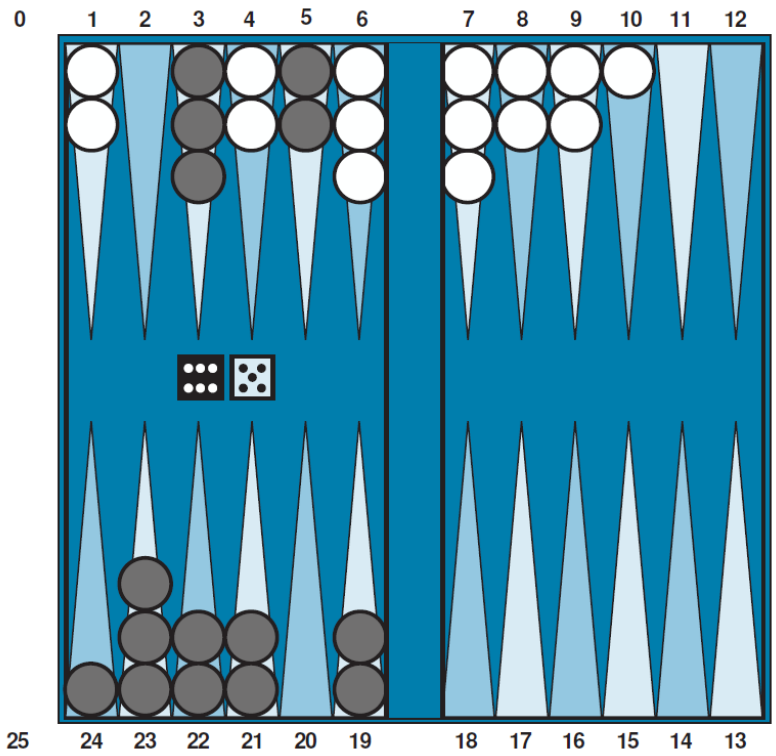
Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.



- Include *chance nodes* in addition to MAX and MIN nodes.

II. Stochastic Games

Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.

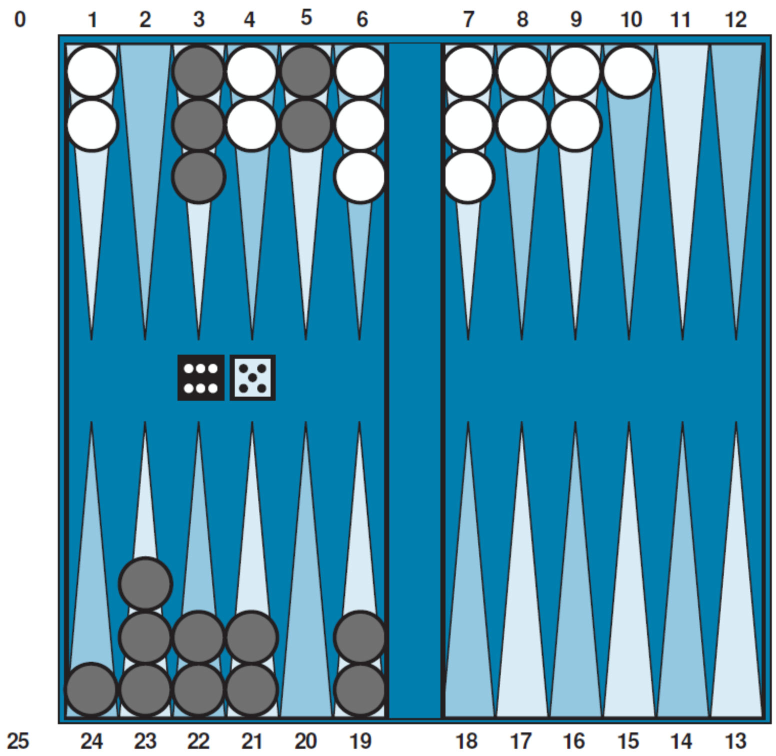


- Include *chance nodes* in addition to MAX and MIN nodes.

Throwing two dice (unordered):

II. Stochastic Games

Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.



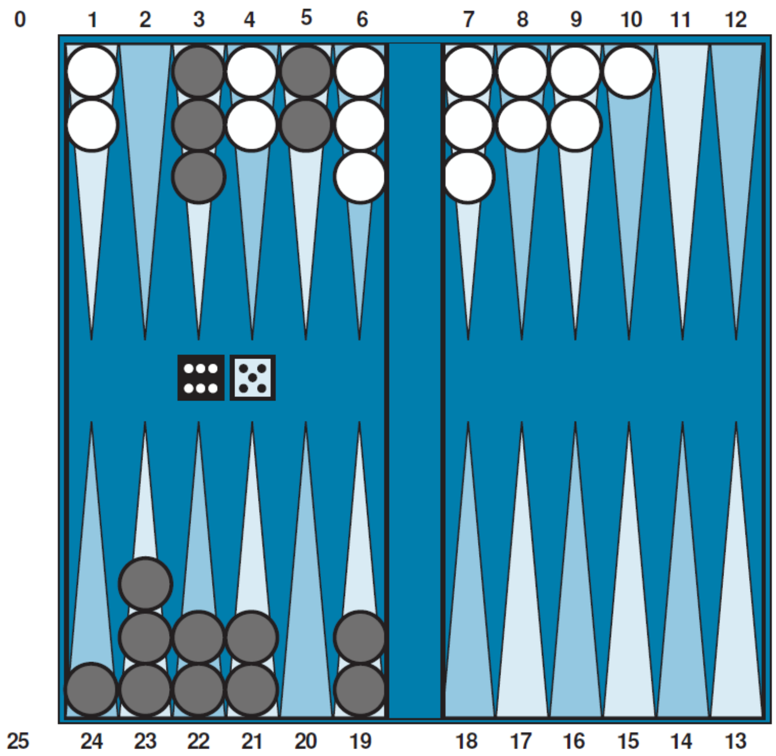
- Include *chance nodes* in addition to MAX and MIN nodes.

Throwing two dice (unordered):

1-1, …, 6-6: probability $1/36$ each

II. Stochastic Games

Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.



- Include *chance nodes* in addition to MAX and MIN nodes.

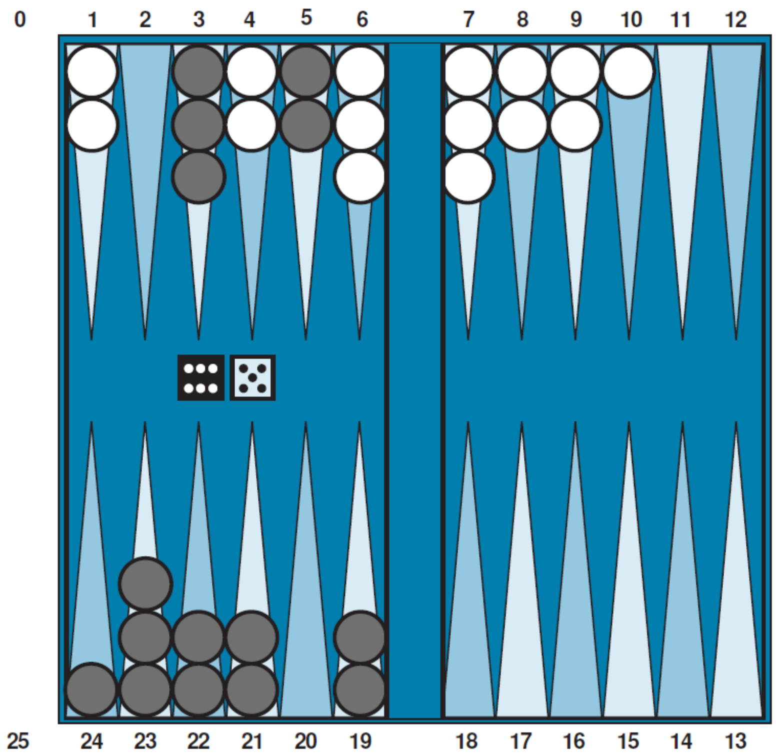
Throwing two dice (unordered):

1-1, …, 6-6: probability $1/36$ each

1-2,…,1-6,2-3,…, 5-6: probability $1/18$ each

II. Stochastic Games

Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.



- Include *chance nodes* in addition to MAX and MIN nodes.

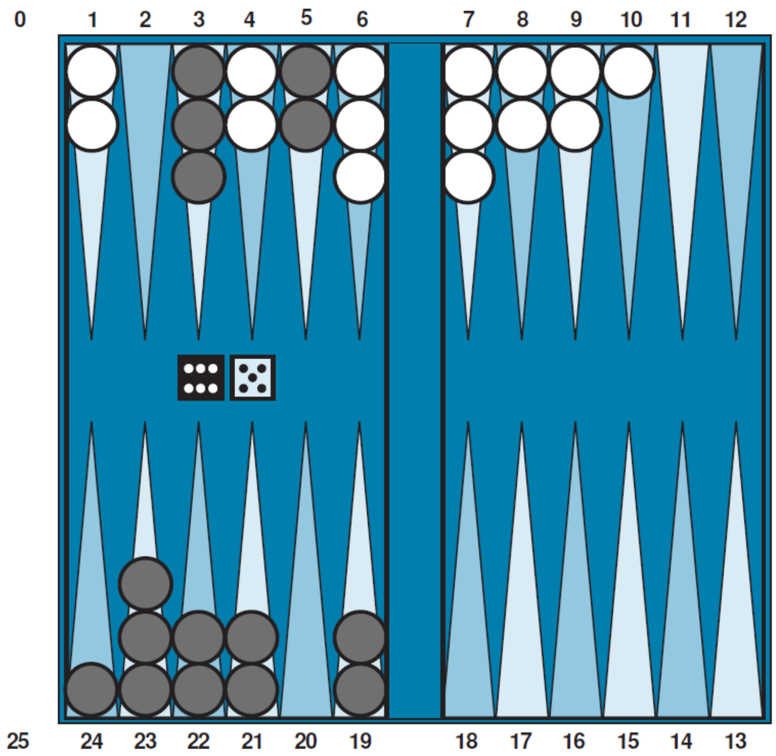
Throwing two dice (unordered):

1-1, …, 6-6: probability $1/36$ each

$\underbrace{\text{1-2,…,1-6,2-3,…, 5-6}}_{15}$: probability $1/18$ each

II. Stochastic Games

Some games (e.g., backgammon) have randomness due to throwing of dice. They combine both luck and skill.



- Include *chance nodes* in addition to MAX and MIN nodes.

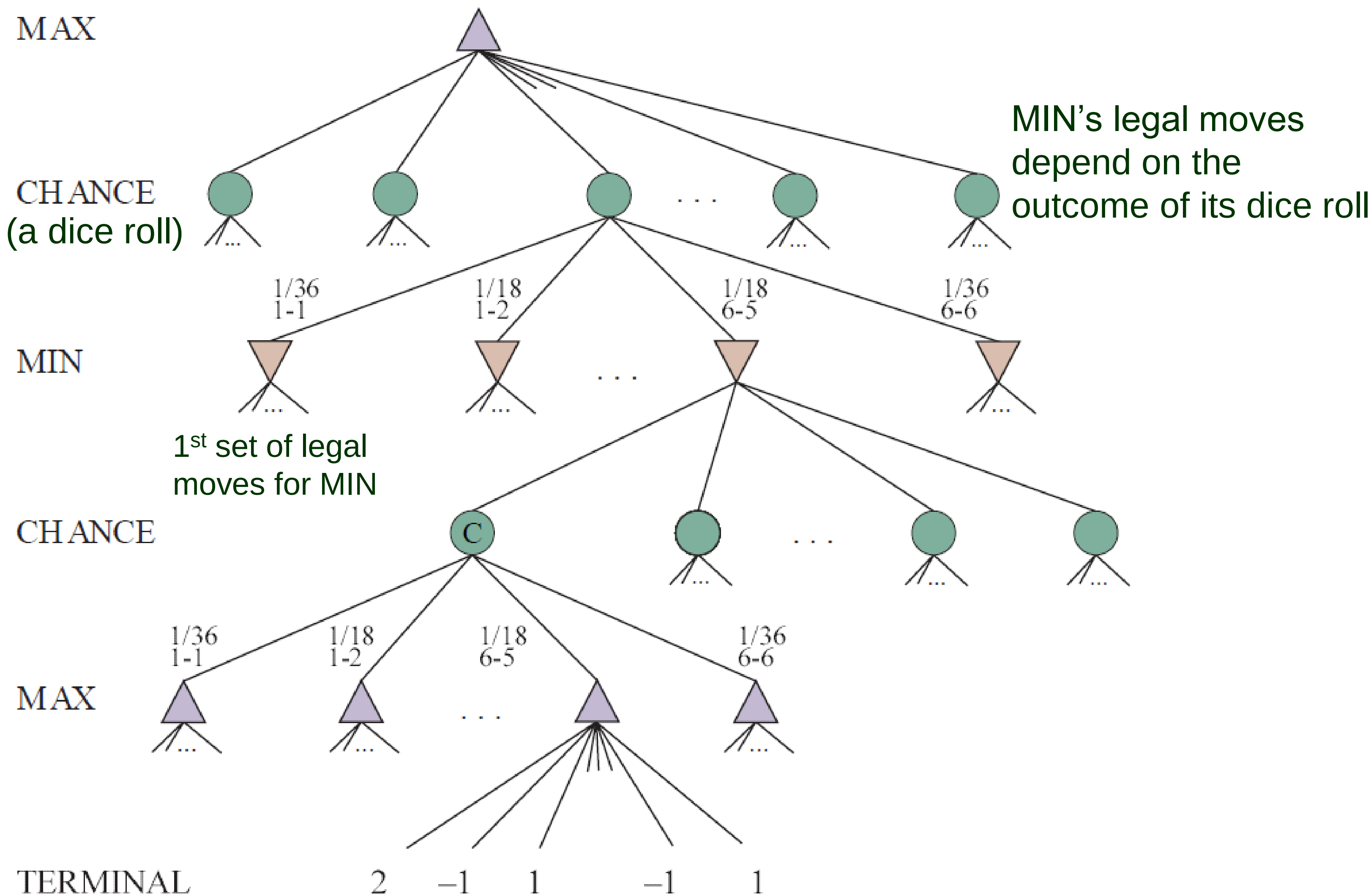
Throwing two dice (unordered):

1-1, …, 6-6: probability $1/36$ each

$\underbrace{\text{1-2,…,1-6,2-3,…, 5-6}}_{15}$: probability $1/18$ each

- Calculate expected value (called *expectiminimax* value) of a position.

Game Tree for a Backgammon Position



Expectiminimax Value

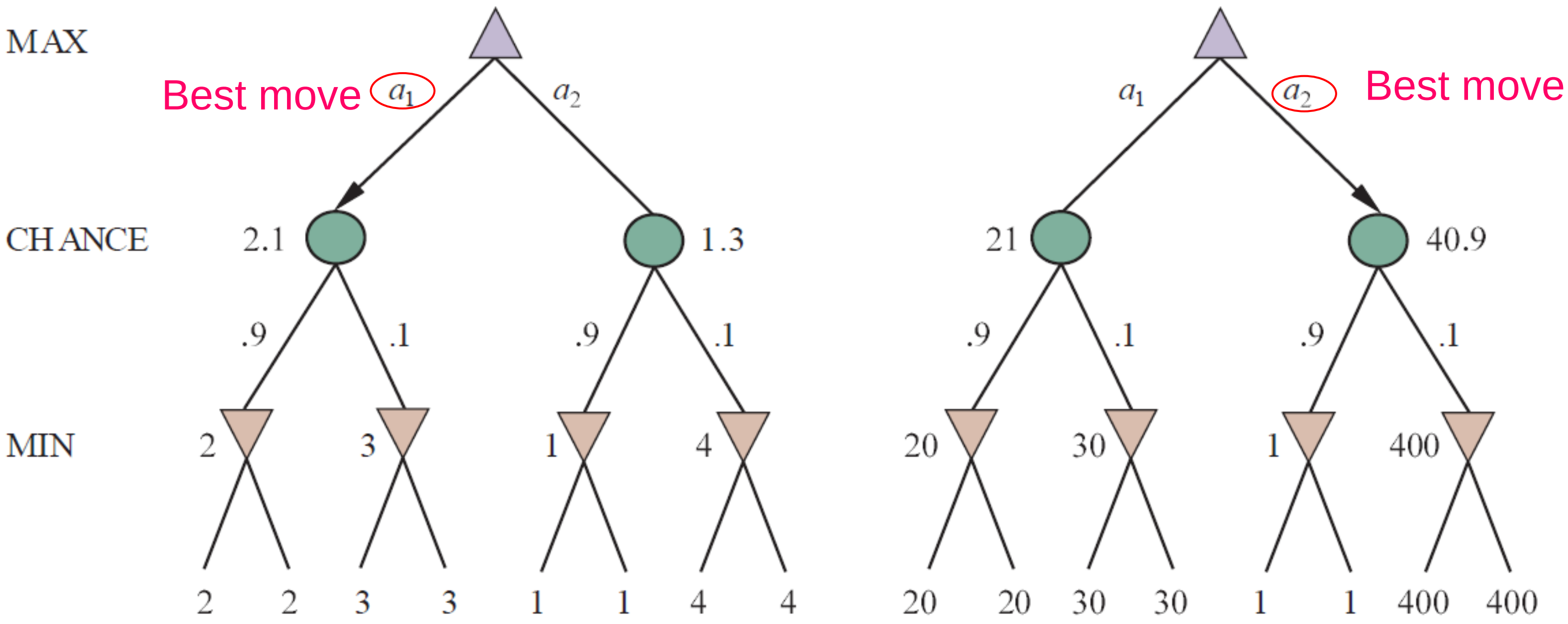
$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \\ \boxed{\sum_r P(r)\ \text{EXPECTIMINIMAX}(\text{RESULT}(s,r))} & \text{if TO-MOVE}(s) = \text{CHANCE} \end{cases}$$

expected value

one possible dice roll

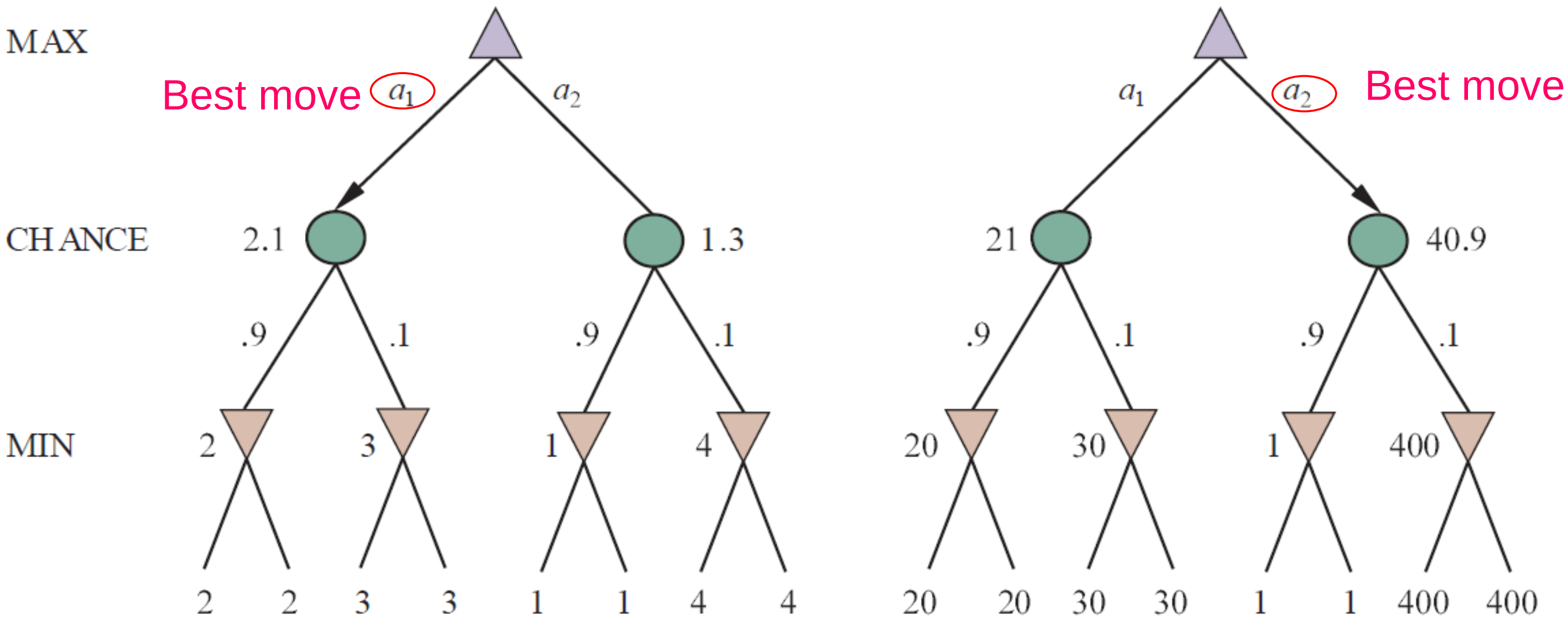
Evaluation Functions

Evaluation functions with the same order of leaf values can yield different move choices at a state.



Evaluation Functions

Evaluation functions with the same order of leaf values can yield different move choices at a state.



Alpha-beta pruning is still applicable if we can bound values on chance nodes.