Scalable ML

10605-10805

Monte Carlo Tree Search

Barnabás Póczos

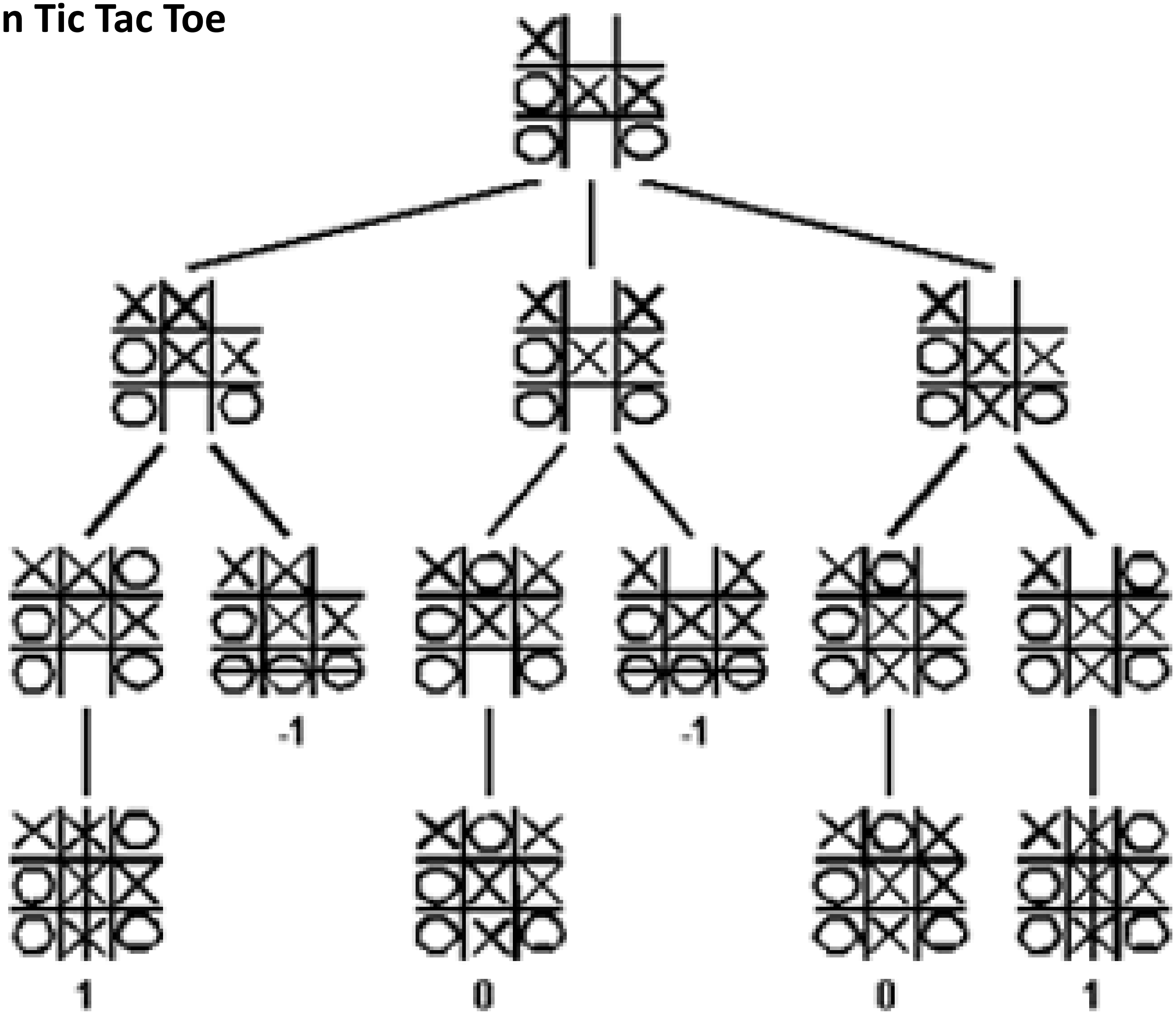
Introduction

Reading material: "Monte Carlo Tree Search and Rapid Action Value Estimation in computer Go" by Sylvain Gelly

- **Monte-Carlo tree search** is a paradigm for search in board games.
- It has revolutionized the performance of computer Go programs.
- MCTS in Go: MoGo, **first program that achieved dan (master) level at 9 × 9 Go.**
- **Game tree in 19x19 Go games:** Breadth ~ 250, Depth ~ 150

Game Tree

Game tree in Tic Tac Toe



Monte-Carlo Tree Search

- The key idea of Monte-Carlo Tree Search is to **simulate many thousands of random games from the current position ONLINE during the game**, using self-play.

- **From any starting board position we create a search tree. New positions are added into a search tree**, and each node of the **tree contains a value that predicts who will win** from that position assuming perfect play.

- The search tree is used to guide simulations along promising paths by selecting the child node [i.e. action] with highest potential value

Simulation-based search

- **Two player games.**
- Black and White alternate turns.

Actions: $a_t \in \mathcal{A}(s_t)$

Policies:

$$\pi(s,a) = [\pi_B(s,a), \pi_W(s,a)]$$

$$\pi_B(s,a) = Pr_B(a|s)$$

$$\pi_W(s,a) = Pr_W(a|s)$$

- The game finishes upon reaching a terminal state with outcome z.
- Black's goal is to maximize z; White's goal is to minimize z.

Simulation Policy

Each simulated game, which we call a simulation, starts from a root state s_0, and sequentially samples states and actions, without backtracking, **until the game terminates**.

$$\text{Simulation policy } \pi(s,a)\text{: to select } a_t \sim \pi(s_t,\cdot).$$

Simulation:

$$s_t \Rightarrow s_{t+1} \Rightarrow s_{t+2} \cdots s_{T-1} \Rightarrow s_T \Rightarrow \text{Outcome } z$$

Terminal state

$$a_t \sim \pi(s_t,\cdot) \quad a_{t+1} \sim \pi(s_{t+1},\cdot) \quad a_{T-1} \sim \pi(s_{T-1},\cdot)$$

Action-value function

Definition [action-value function]: the expected outcome after playing action a in state s, and then following policy π_B, π_W until termination:

$$Q^{\pi}(s,a) = Q^{\pi_B,\pi_W}(s,a) = \mathbb{E}_{\pi_B,\pi_W}[z|s_t = s, a_t = a]$$

Notation

$N(s)$ complete games are simulated with policy π from state s.

z_i is the outcome of the ith simulation;

$\mathbb{I}_i(s,a)$ is an indicator function returning 1 if action a was selected in state s during the ith simulation, and 0 otherwise;

$N(s,a) = \sum_{i=1}^{N(s)} \mathbb{I}_i(s,a)$ counts the total number of simulations in which action a was selected in state s.

Monte-Carlo Simulation: Evaluate (s,a)

Monte-Carlo simulation provides a method for estimating $Q^\pi(s_0, a)$.

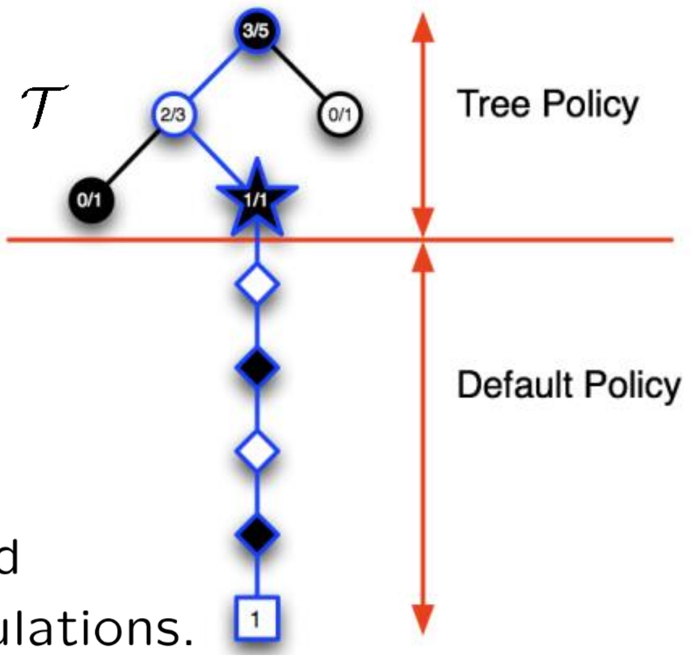
The estimated value of $Q^\pi(s, a)$ is the mean outcome of all simulations in which action a was selected in state s:

$$\hat{Q}^\pi(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{N(s)} \mathbb{I}_i(s, a) z_i$$

In its most basic form, Monte-Carlo simulation is only used to evaluate actions, but not to improve the simulation policy

However, the basic algorithm can be extended by progressively favoring the most successful actions, or by progressively pruning away the least successful actions

Monte-Carlo Tree Search



$n(s)$: node of state s in the search tree $\mathcal{T}$

Each node in the tree has:

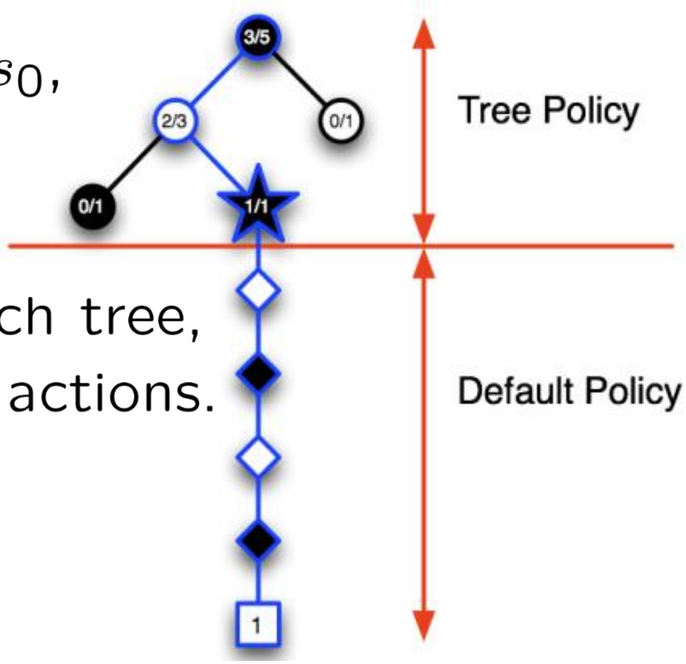
$N(s)$: state s was visited $N(s)$ times during the simulations.

$N(s,a)$: state s with action a was visited and chosen $N(s,a)$ times during the simulations.

$Q(s,a)$: estimated action value function

Monte-Carlo Tree Search

Simulations start from the root state s_0, and are divided into two stages.



When state s_t is represented in the search tree, $s_t \in \mathcal{T}$, a tree policy is used to select actions.

Otherwise, a default policy is used to roll out simulations to completion.

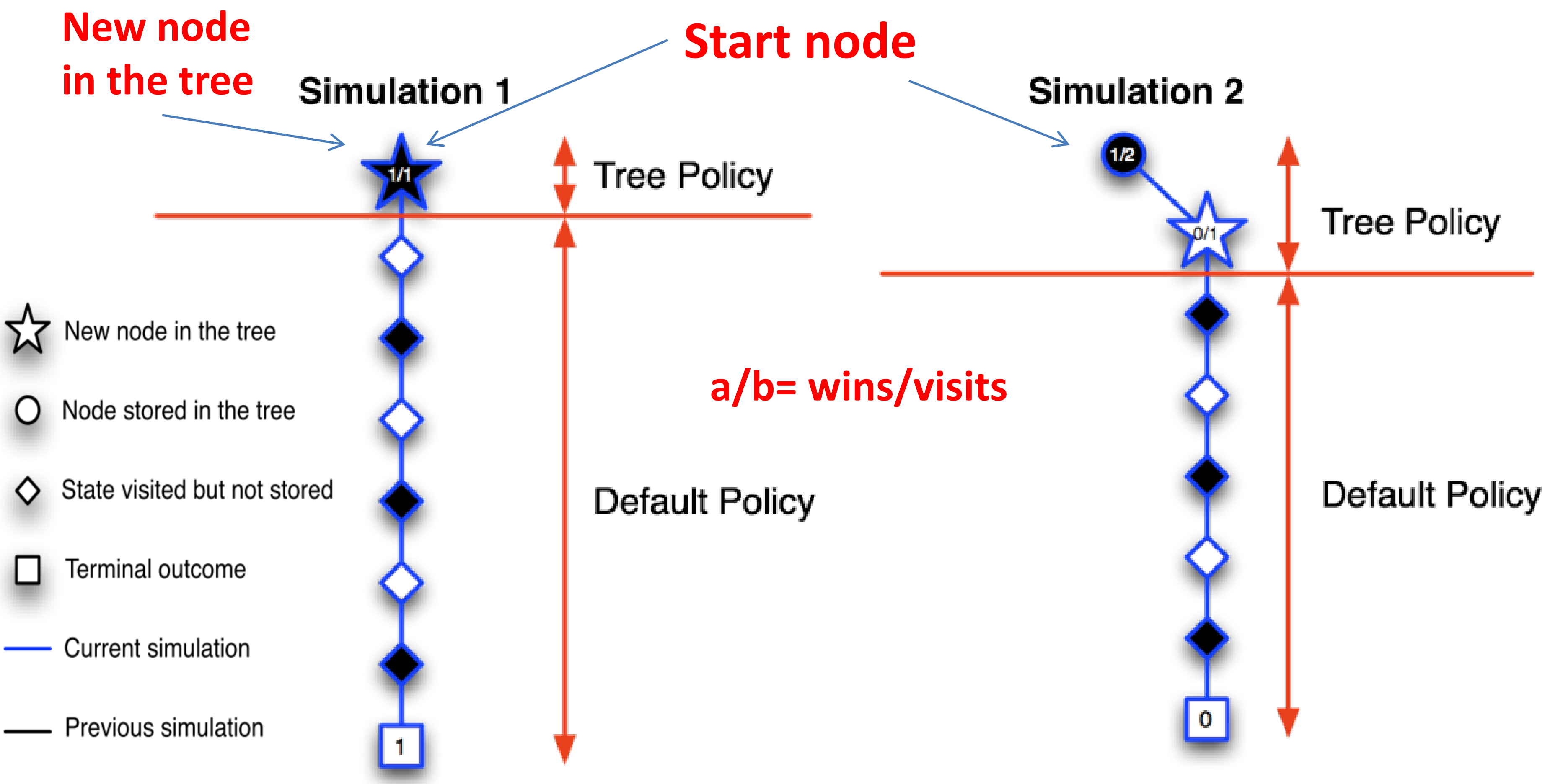
Tree policy: E.g. greedy: $\arg\max_a Q(s_t, a)$ if s_t is in the tree $\mathcal{T}$.

Default policy: E.g. uniformly random among all legal actions

Notation

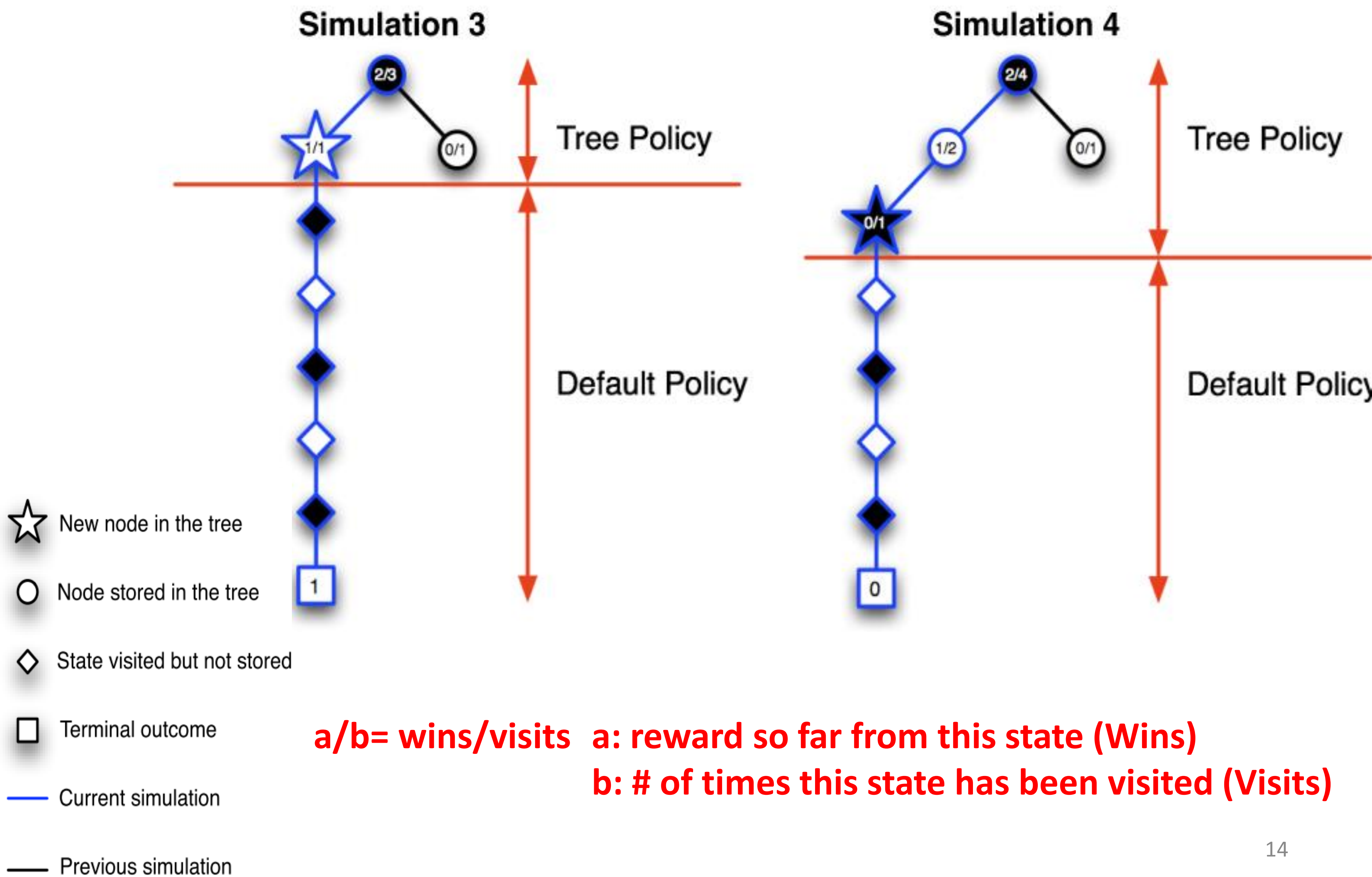
- New node in the tree
- Node stored in the tree
- State visited but not stored
- Terminal outcome
- Current simulation
- Previous simulation

Monte-Carlo Tree Search



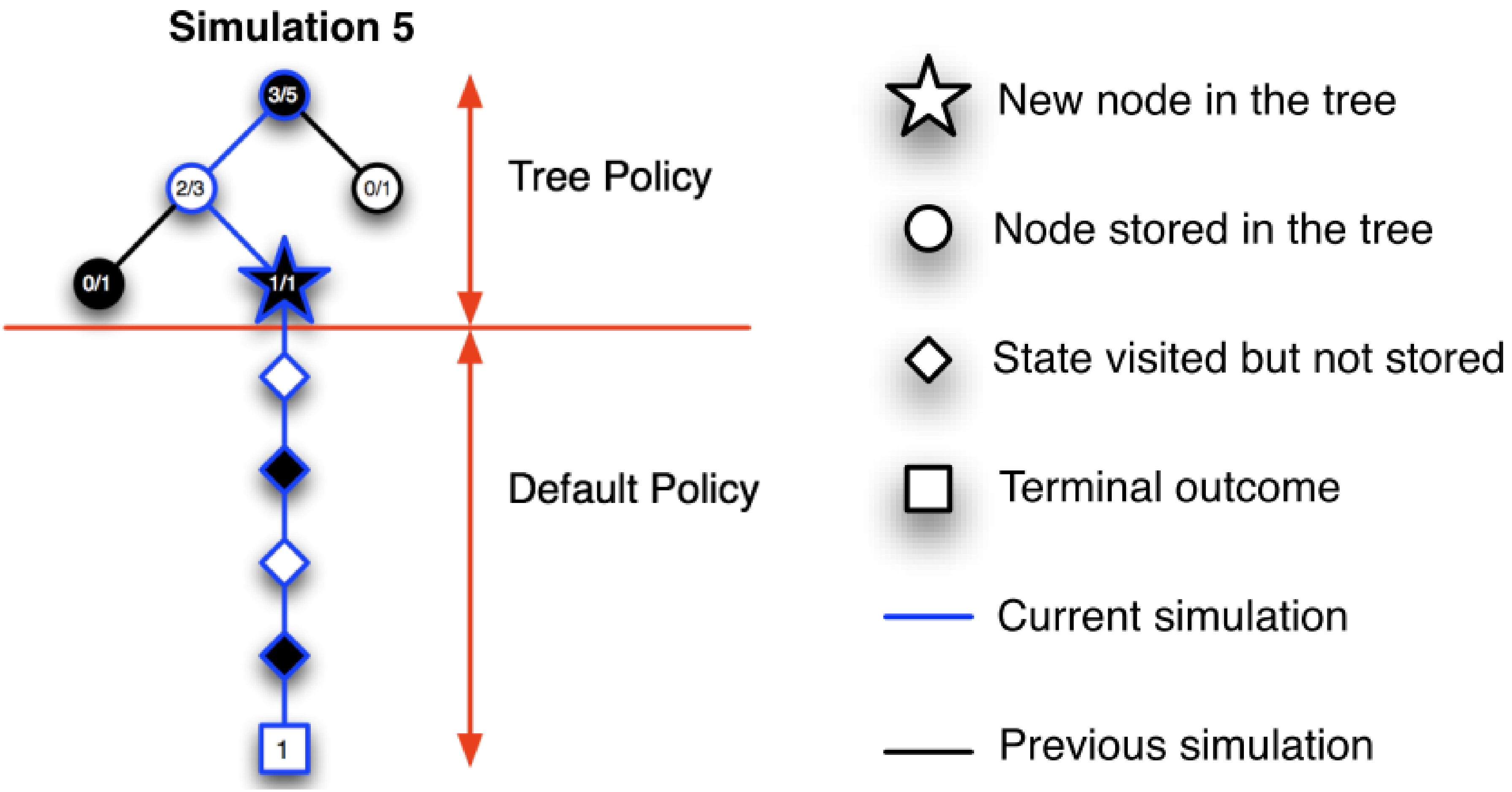
- Each simulation has an outcome of 1 for a black win or 0 for a white win (square).
- At each simulation, a new node (star) is added into the search tree.
- The value of each node in the search tree (circles and star) is then updated to count the number of black wins, and the total number of visits (wins/visits).

Monte-Carlo Tree Search



a/b= wins/visits **a: reward so far from this state (Wins)**
b: # of times this state has been visited (Visits)

Monte-Carlo Tree Search



Search Tree Update

Every state and action in the search tree is evaluated by its mean outcome during simulations.

After each simulation $s_0, a_0, s_1, a_1, \ldots, s_T$ with outcome z, each node in the search tree, $n(s_t)|s_t \in \mathcal{T}$, updates its count, and updates its action value $Q(s_t, a_t)$ to the new MC value:

$$N(s_t) \Leftarrow N(s_t) + 1$$

$$N(s_t, a_t) \Leftarrow N(s_t, a_t) + 1$$

$$Q(s_t, a_t) \Leftarrow Q(s_t, a_t) + \frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$$

because

$[N(s_t, a_t) + 1]Q(s_t, a_t)$ should be updated to $N(s_t, a_t)Q(s_t, a_t) + z$

Upper Confidence Bound applied to Trees (UCT)

- Greedy action selection can often be an inefficient as it will typically avoid searching actions after one or more poor outcomes, even if there is significant uncertainty about the value of those actions.

- To explore the search tree more efficiently, the principle of **optimism in the face of uncertainty** can be applied, which favors the actions with the greatest potential value.

- To implement this principle, each action value receives a bonus that corresponds to the amount of uncertainty in the current value of that state and action.

- The UCT algorithm applies this principle to Monte-Carlo tree search

Upper Confidence Bound applied to Trees (UCT)

The action value is augmented by an exploration bonus.

The exploration bonus is highest for rarely visited state-action pairs, and the tree policy selects the action by maximizing the augmented value

$$Q(s_t, a_t)^{\oplus} = Q(s_t, a_t) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$$

$$a^* = \text{argmax}_a Q^{\oplus}(s, a)$$

Exploration bonus.
The smaller the N(s,a),
the bigger it is

Here c is a scalar exploration constant

Algorithm 1 Two Player UCT

procedure UCTSEARCH(s_0)
1 **while** time available **do**
SIMULATE($board, s_0$)
end while
$board.SetPosition(s_0)$
return SELECTMOVE($board, s_0, 0$)
end procedure

procedure SIMULATE($board, s_0$)
2 $board.SetPosition(s_0)$
$[s_0, ..., s_T]$ = SIMTREE($board$)
z = SIMDEFAULT($board$)
BACKUP($[s_0, ..., s_T], z$)
end procedure

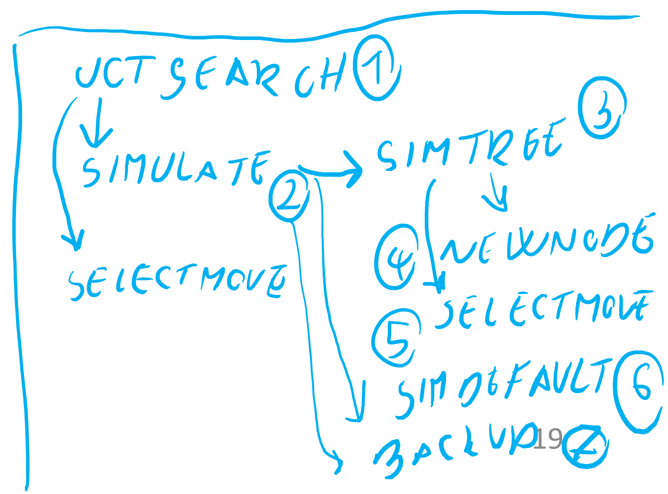
procedure SIMTREE($board$)
3 $c = exploration\ constant$
$t = 0$
while not $board.GameOver()$ **do**
$s_t = board.GetPosition()$
if $s_t \notin tree$ **then**
NEWNODE(s_t)
return $[s_0, ..., s_t]$
end if
a = SELECTMOVE($board, s_t, c$)
$board.Play(a)$
$t = t + 1$
end while
return $[s_0, ..., s_{t-1}]$
end procedure

procedure SIMDEFAULT($board$) 6
while not $board.GameOver()$ **do**
a = DEFAULTPOLICY($board$)
$board.Play(a)$
end while
return $board.BlackWins()$
end procedure

procedure SELECTMOVE($board, s, c$)
$legal = board.Legal()$
if $board.BlackToPlay()$ **then** 5
$$a^* = \operatorname*{argmax}_{a \in legal} \left(Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \right)$$
else
$$a^* = \operatorname*{argmin}_{a \in legal} \left(Q(s,a) - c\sqrt{\frac{\log N(s)}{N(s,a)}} \right)$$
end if
return a^*
end procedure

procedure BACKUP($[s_0, ..., s_T], z$)
for $t = 0$ **to** T **do** 7
$N(s_t) = N(s_t) + 1$
$N(s_t, a_t)$ ++
$Q(s_t, a_t)$ += $\frac{z - Q(s_t,a_t)}{N(s_t,a_t)}$
end for
end procedure

procedure NEWNODE(s)
4 $tree.Insert(s)$
$N(s) = 0$
for all $a \in \mathcal{A}$ **do**
$N(s,a) = 0$
$Q(s,a) = 0$
end for
end procedure



Algorithm: Two Player UCT

procedure $\text{UCTSEARCH}(s_0)$
  while time available **do**
    $\text{SIMULATE}(board, s_0)$
  end while
  $board.SetPosition(s_0)$
  return $\text{SELECTMOVE}(board, s_0, 0)$
end procedure

Suggest a move from s_0

Simulate complete games from s_0 while have thinking time and meanwhile build a search tree from s_0

When we have no more time, move greedily according to the tree policy with exploration constant c=0

procedure $\text{SIMULATE}(board, s_0)$
  $board.SetPosition(s_0)$
  $[s_0, a_0, s_1, a_1, \ldots, s_T] = SimTree(board)$
  $z = \text{SIMDEFAULT}(board)$
  $Backup([s_0, a_0, s_1, a_1, \ldots, s_T], z)$
end procedure

Simulate one game from s_0 while have time

Simulate a game with tree policy as long as possible and add one new node to the tree

Then complete the game with default policy

UCT update equations in the tree along the states in the tree policy part of the simulated game

Algorithm: Two Player UCT

procedure SIMTREE($board$) — Simulate a game with tree policy as long as possible

```
procedure SimTree(board)
    c = exploration constant
    t = 0
    while not board.GameOver() do
        s_t = board.GetPosition()
        if s_t ∉ tree then
            NewNode(s_t)
            return [s_0, a_0, s_1, a_1, ..., s_t]
        end if
        a = SelectMove(board, s_t, c)
        board.Play(a)
        t = t + 1
    end while
    return [s_0, a_0, s_1, a_1, ..., s_{t-1}]
end procedure
```

- NEWNODE(s_t): Add node to the tree if this is a new state that is not in the tree
- $a = $ SELECTMOVE($board, s_t, c$): UCT tree policy move from s_t.

procedure SIMDEFAULT($board$) — Finish a simulated game with default policy

```
procedure SimDefault(board)
    while not board.GameOver() do
        a = DefaultPolicy(board)
        board.Play(a)
    end while
    return board.BlackWins()
end procedure
```

Algorithm: Two Player UCT

procedure SELECTMOVE($board, s, c$)
 $legal = board.Legal()$
 if $board.BlackToPlay()$ **then**
  $$a^* = \underset{a \in legal}{\operatorname{argmax}} \left(Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \right)$$
 else
  $$a^* = \underset{a \in legal}{\operatorname{argmin}} \left(Q(s,a) - c\sqrt{\frac{\log N(s)}{N(s,a)}} \right)$$
 end if
 return a^*
end procedure

Select the UCT tree policy move both for simulations and for the actual move during the game. For the actual move, c is set to c=0.

Move for Black

Move for White

procedure $Backup([s_0, a_0, s_1, a_1, \ldots, s_T], z)$ UCT update rules
 for $t = 0$ **to** T **do**
  $N(s_t) = N(s_t) + 1$
  $N(s_t, a_t) ++$
  $Q(s_t, a_t) += \frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$
 end for
end procedure

procedure NEWNODE(s)
 $tree.Insert(s)$
 $N(s) = 0$
 for all $a \in \mathcal{A}$ **do**
  $N(s,a) = 0$
  $Q(s,a) = 0$
 end for
end procedure

If it is a new node, insert it into the tree

Monte Carlo Tree Search in Go

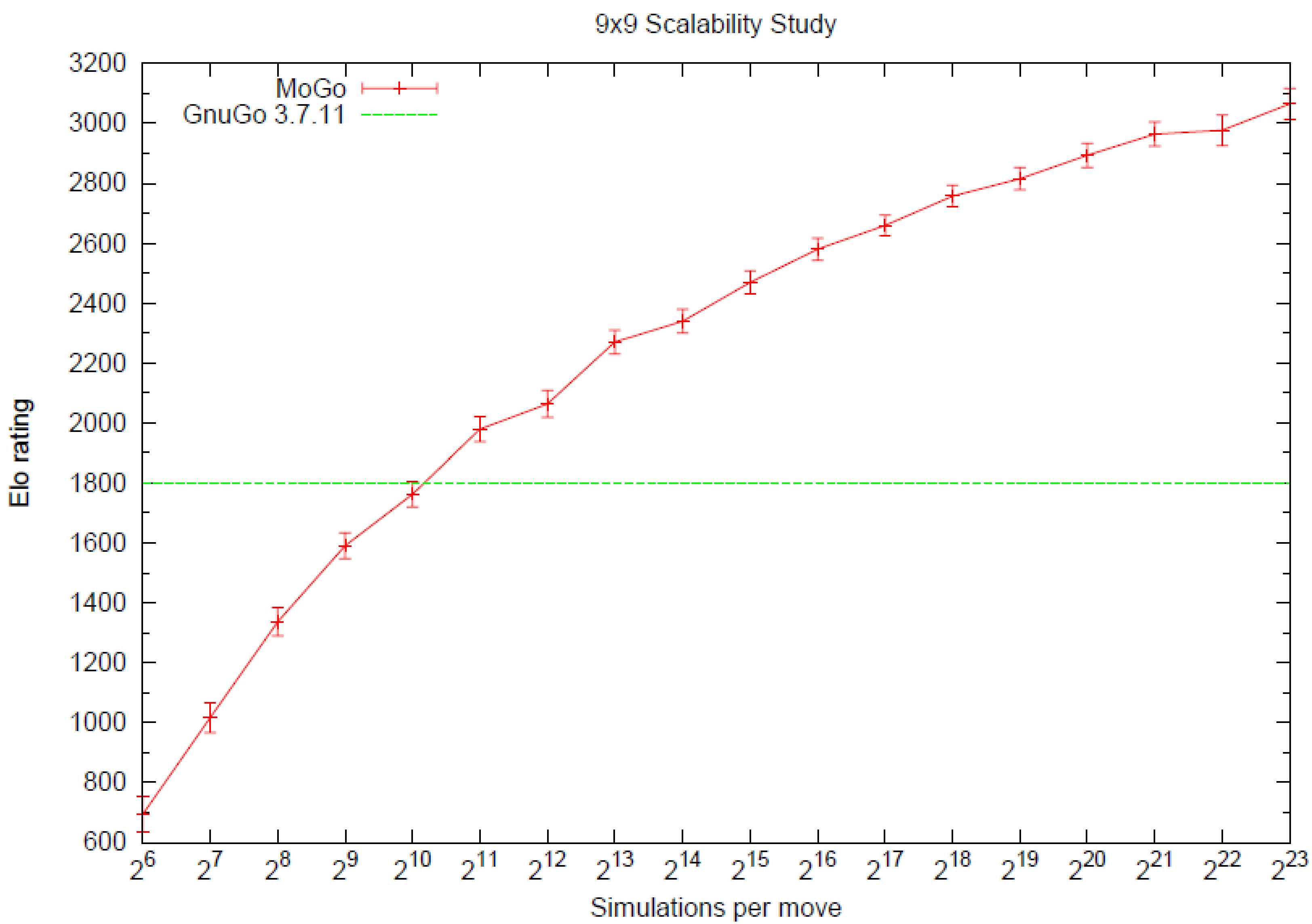
- First Monte Carlo Tree Search in Go: **Crazy Stone**.
  - 2006: Gold medal in 9x9 Go Olympiad
  - First version used Normal approximation on uncertainty instead of UCT

- First UCT in Go: MoGo.
  - A new era started in Go
  - 2007: ~2500 Elo scores on 9x9 Go

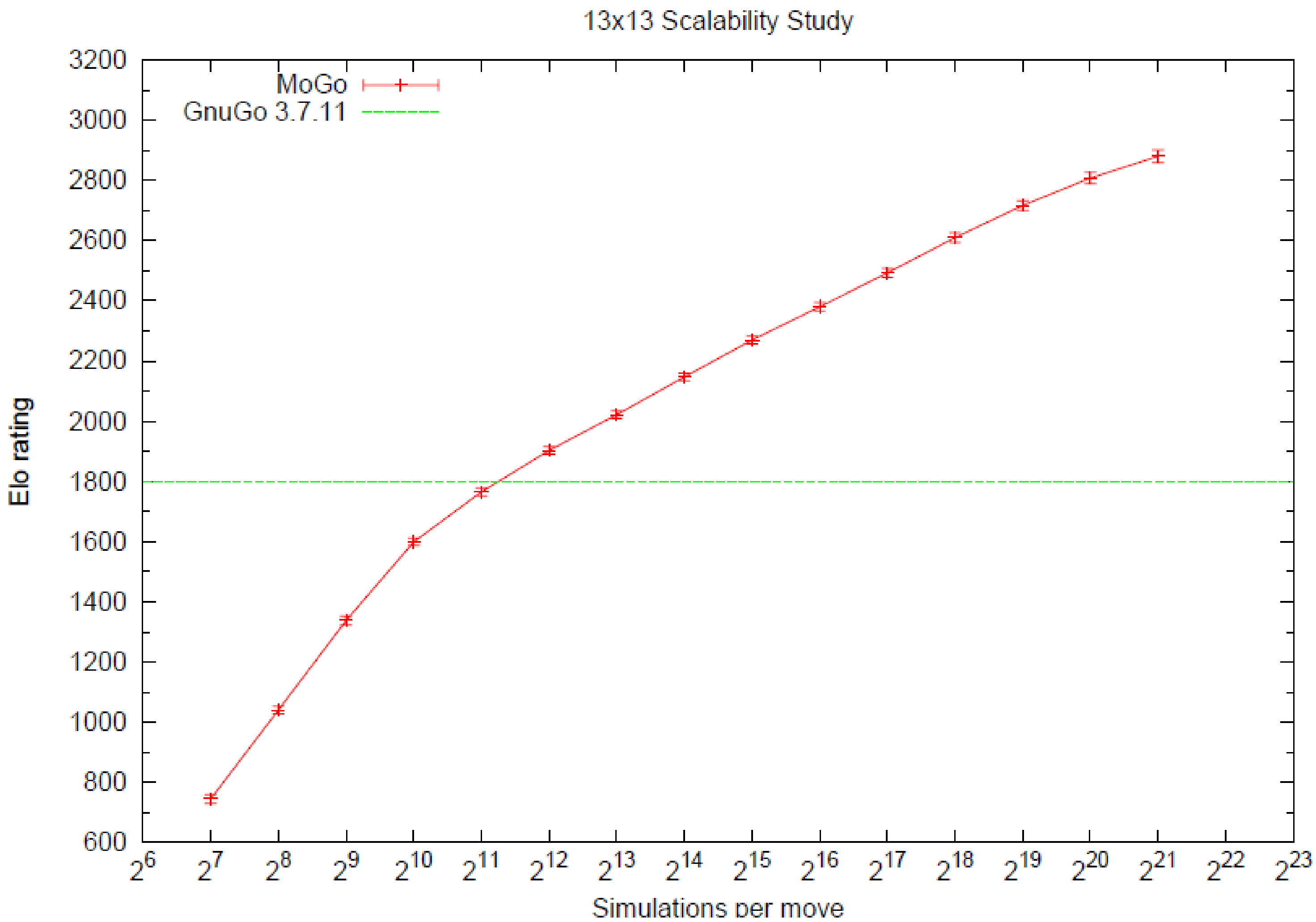
Year	*Program*	*Description*	*Elo*
2006	*Indigo*	Pattern database, Monte-Carlo simulation	1400
2006	*GnuGo*	Pattern database, alpha-beta search	1800
2006	*Many Faces*		1800
2006	*NeuroGo*	Temporal-difference learning, neural network	1850
2007	*RLGO*	Temporal-difference search	2100
2007	*MoGo*	Variants of heuristic MC–RAVE	2500
2007	*Crazy Stone*		2500
2009	*Fuego*		2700
2010	*Many Faces*		2700
2010	*Zen*		2700

Table 2: Approximate Elo ratings, on the Computer Go Server, of 9×9 Go programs discussed in the text.

Monte Carlo Tree Search in Go



Monte Carlo Tree Search in Go



Thanks for your Attention! ☺