

06 Monte Carlo Tree Search (MCTS)

Table of contents

- Planning with a Model
- Simulation-Based Search
- Example: The Game of Go

Planning with a Model

Planning with a Model

Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$

- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$

Using favourite planning algorithm

- Tree search
- ...

Sample-Based Planning

A simple but powerful approach to planning is to use the model *only* to generate samples

Sample experience from model

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

Apply *model-free* RL to samples, e.g.:

- Q-learning
- Monte-Carlo control or Sarsa

Sample-based planning methods are often more efficient

Back to the AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

Real experience

$A, 0, B, 0$

$B, 1$

$B, 1$

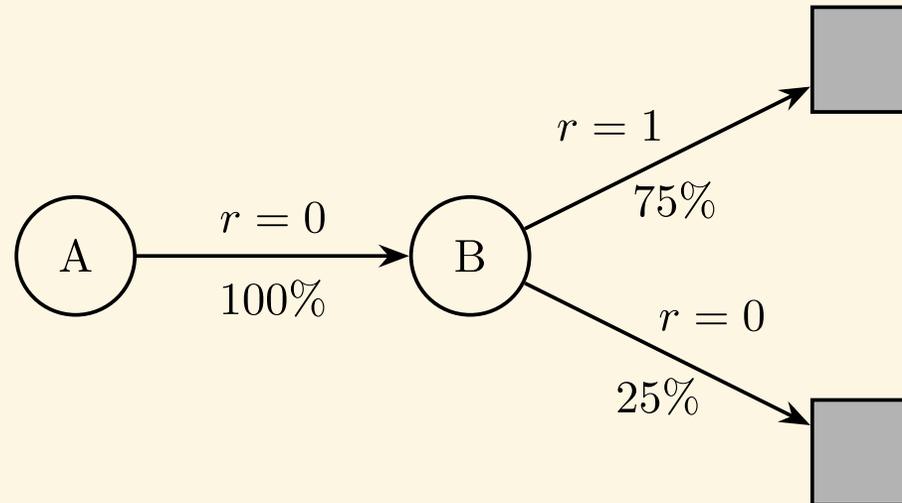
$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 0$



Sampled experience

$B, 1$

$B, 0$

$B, 1$

$A, 0, B, 1$

$B, 1$

$A, 0, B, 1$

$B, 1$

$B, 0$

e.g. Monte-Carlo learning: $V(A) = 1; V(B) = 0.75$

We can sample as many trajectories as we want from the model,
unlike from the environment

- we essentially have infinite data

Planning with an Inaccurate Model

Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$

- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- i.e. Model-based RL is only as good as the estimated model

When the model is *inaccurate*, planning process will compute a sub-optimal policy

- **Solution 1:** when model is wrong, use model-free RL
- **Solution 2:** reason explicitly about model uncertainty (e.g. Bayesian approach)

Real and Simulated Experience

We consider two sources of experience

Real experience Sampled from environment (true MDP)

$$S' \sim \mathcal{P}_{s,s'}^a$$
$$R = \mathcal{R}_s^a$$

Simulated experience Sampled from model (approximate MDP)

$$S' \sim \mathcal{P}_\eta(S' | S, A)$$
$$R = \mathcal{R}_\eta(R | S, A)$$

Integrating Learning and Planning

Model-Free RL

- No model
- **Learn** value function (and/or policy) from **real experience**

Model-Based RL (using Sample-Based Planning)

- Learn a model from real experience
- **Plan** value function (and/or policy) from **simulated experience**

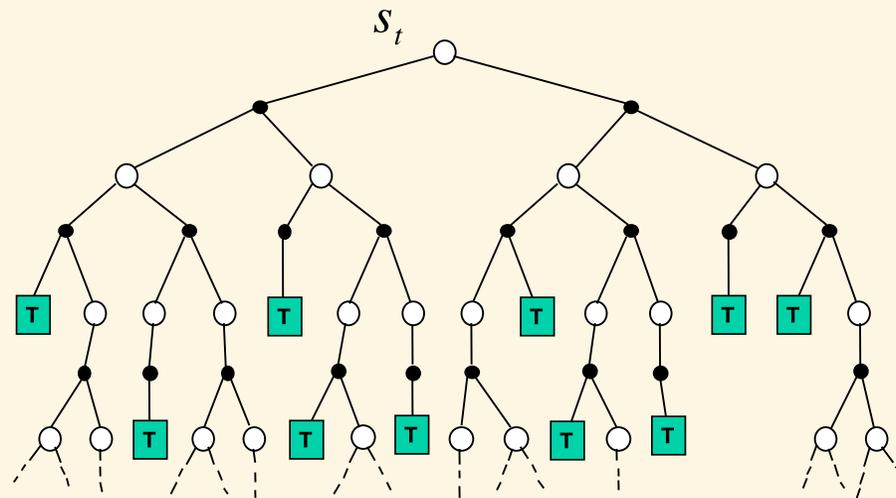
Integrated Architectures

- Learn a model from real experience
- **Learn and plan** value function (and/or policy) from **real and simulated** experience

Simulation-Based Search

Simulation-Based Search (1)

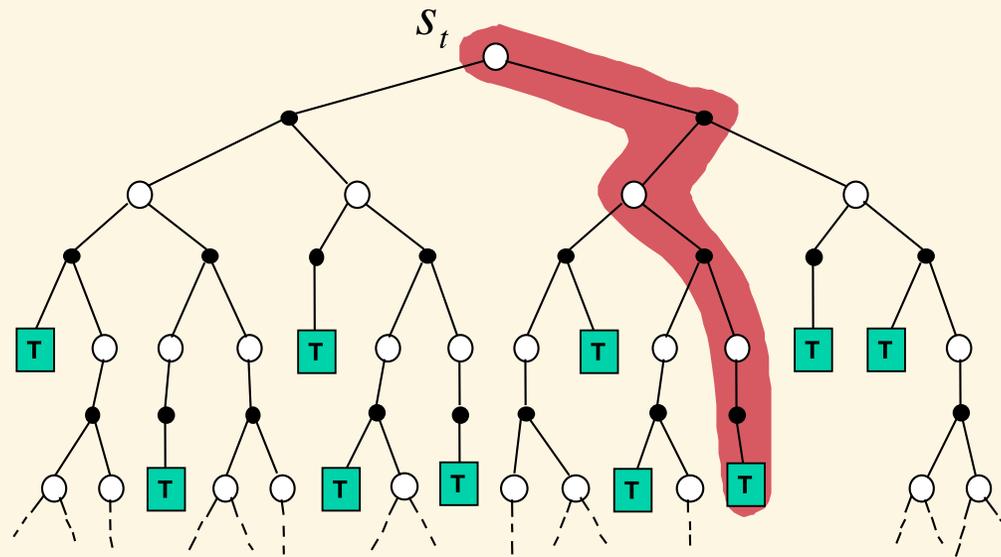
- Forward search algorithms select the best action by **look-ahead**
- They build a **search tree** with the current state s_t at the root
- Using a **model** of the MDP to look ahead



No need to solve whole MDP, just sub-MDP starting from **now**

Simulation-Based Search (2)

- Forward search paradigm using sample-based planning
- Simulate episodes of *experience* from now with the model
- Apply model-free RL to simulated episodes



Simulation-Based Search (3)

Simulate episodes of experience from **now** with the model

$$\{s_t^k, A_t^k, R_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu$$

Apply **model-free** RL to simulated episodes

- Monte-Carlo control \rightarrow Monte-Carlo search
- Sarsa \rightarrow TD search

Simple Monte-Carlo Search

Given a model \mathcal{M}_ν , and a **simulation policy** π

- For each action $a \in \mathcal{A}$
 - Simulate K episodes from current (real) state s_t

$$\{s_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Evaluate actions by mean return (**Monte-Carlo evaluation**)

$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^K G_t \xrightarrow{P} q_\pi(s_t, a)$$

- Select current (real) action with maximum value $a_t = \arg \max_{a \in \mathcal{A}} Q(s_t, a)$

Monte-Carlo Tree Search (Evaluation)

Given a model \mathcal{M}_ν , simulate K episodes from current state s_t using current simulation policy π

$$\{s_t, A_t^k, R_{t+1}^k, S_{t+1}^k, \dots, S_T^k\}_{k=1}^K \sim \mathcal{M}_\nu, \pi$$

- Build a **search tree** containing visited states and actions
- **Evaluate** states $Q(s, a)$ by *mean* return of episodes from each pair s, a

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{k=1}^K \sum_{u=t}^T \mathbf{1}(S_u, A_u = s, a) G_u \xrightarrow{P} q_\pi(s, a)$$

After search is finished, select current (real) action with maximum value in search tree $a_t = \arg \max_{a \in \mathcal{A}} Q(s_t, a)$

Monte-Carlo Tree Search (Simulation)

In MCTS, the simulation policy π **improves**

Each simulation consists of two phases (in-tree, out-of-tree)

- **Tree policy** (improves): pick actions to maximise $Q(S, A)$
- **Default policy** (fixed): pick actions randomly

Monte-Carlo Tree Search

Repeat (each simulation)

Evaluate states $Q(S, A)$ by Monte-Carlo evaluation

Improve tree policy, e.g. by ϵ -greedy(Q)

Monte-Carlo control applied to **simulated experience**

- Converges on the optimal search tree,
 $Q(S, A) \rightarrow q_*(S, A)$

Example: The Game of Go

Example: The Game of Go

The ancient oriental game of Go is 2,500 years old

Considered to be the hardest classic board game

Considered a grand challenge task for AI (*John McCarthy*)

Traditional game-tree search has failed in Go

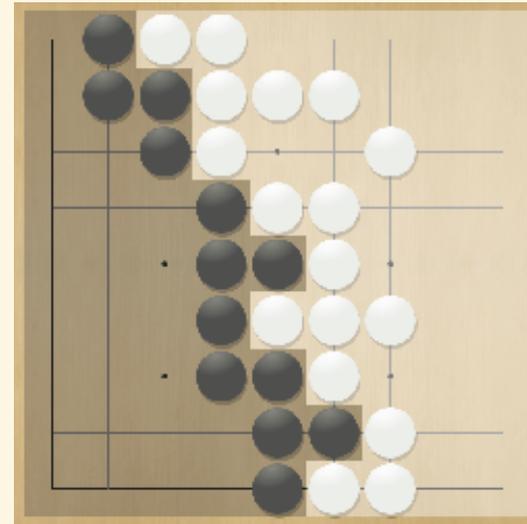
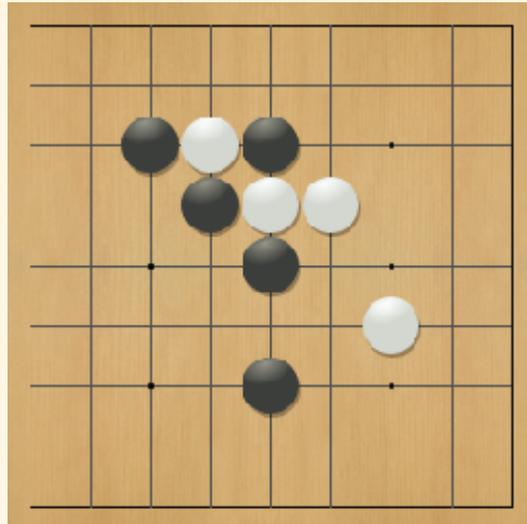


Rules of Go

Usually played on 19x19, also 13x13 or 9x9 board

Simple rules, complex strategy

- Black and white place down stones alternately
- Surrounded stones are captured and removed
- The player with more territory wins the game



Position Evaluation in Go

How good is a position s ?

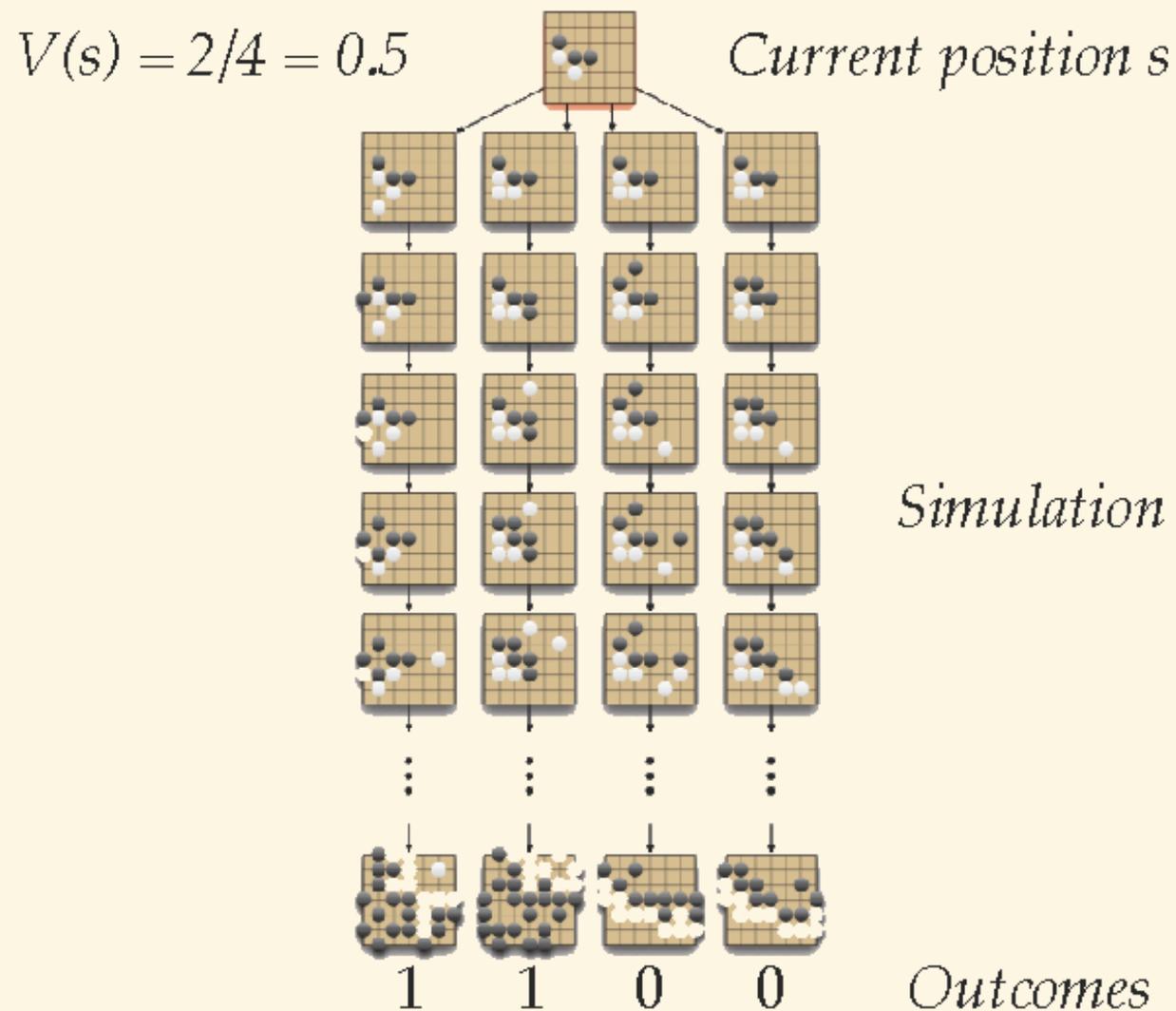
- Reward function (undiscounted):

$$R_t = 0 \quad \text{for all non-terminal steps } t < T$$
$$R_T = \begin{cases} 1 & \text{if Black wins} \\ 0 & \text{if White wins} \end{cases}$$

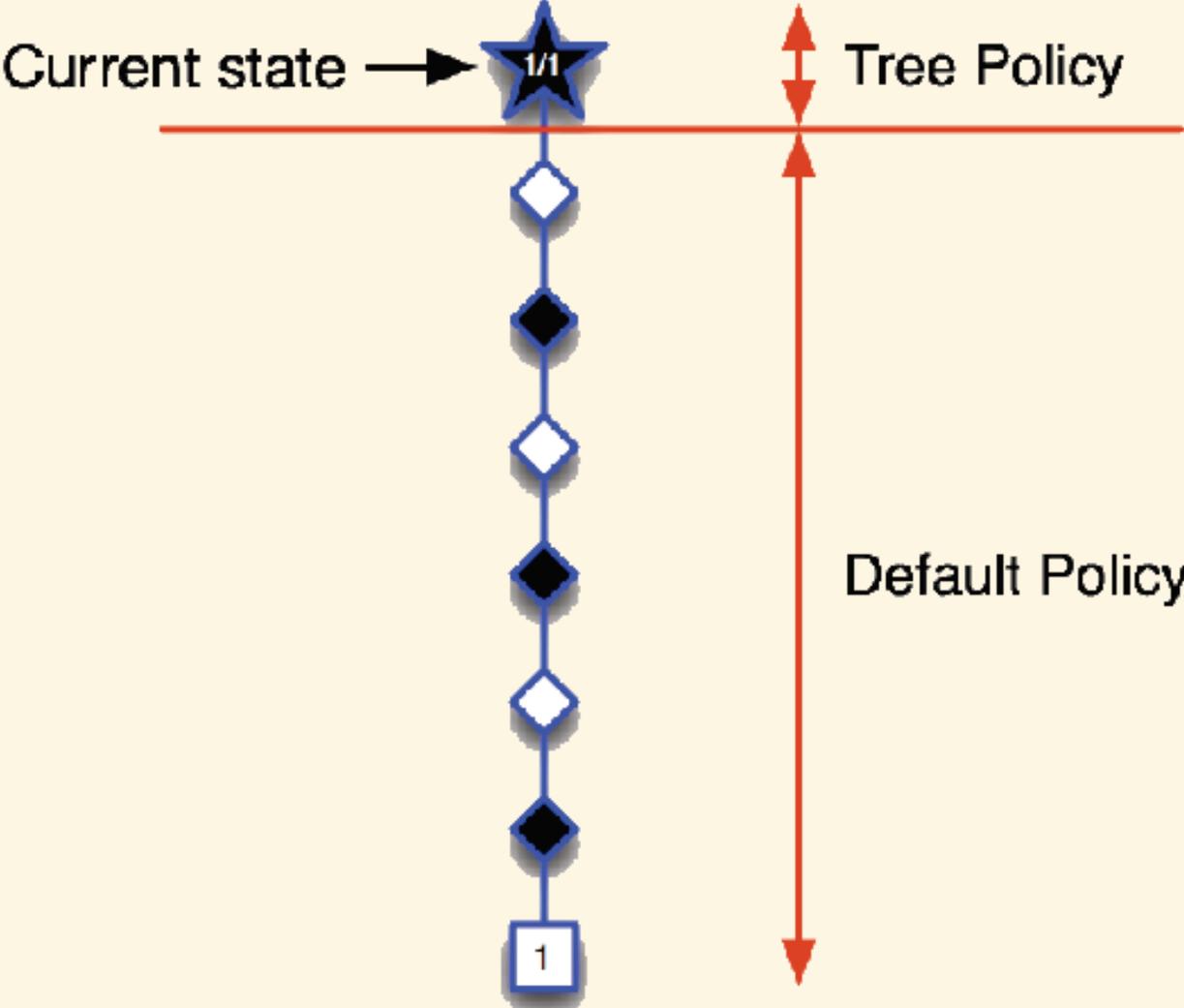
- Policy $\pi = \langle \pi_B, \pi_W \rangle$ selects moves for both players
- Value function (how good is position s):

$$v_\pi(s) = \mathbb{E}_\pi [R_T \mid S = s] = \mathbb{P}[\text{Black wins} \mid S = s]$$
$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

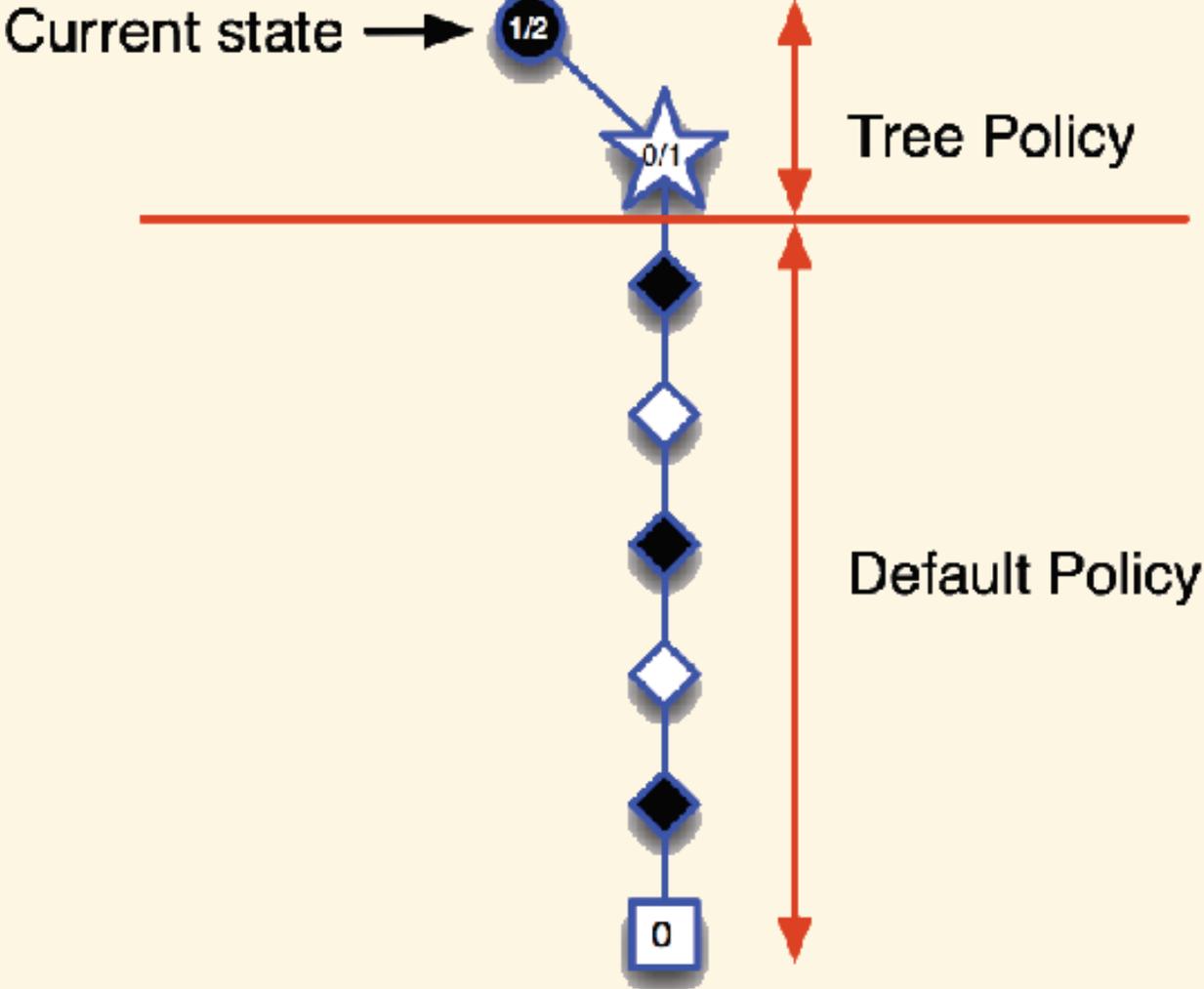
Monte-Carlo Evaluation in Go



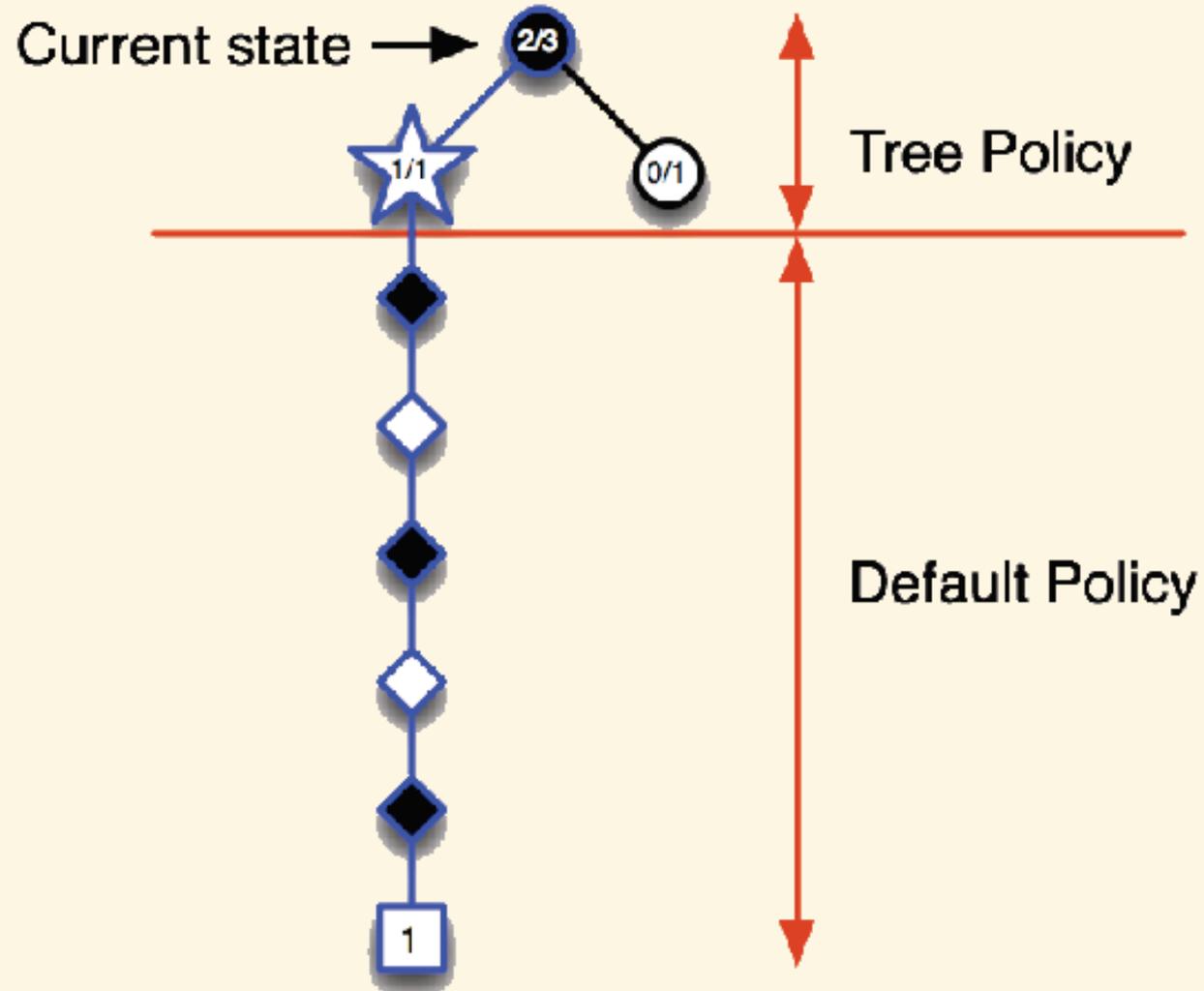
Applying Monte-Carlo Tree Search (1)



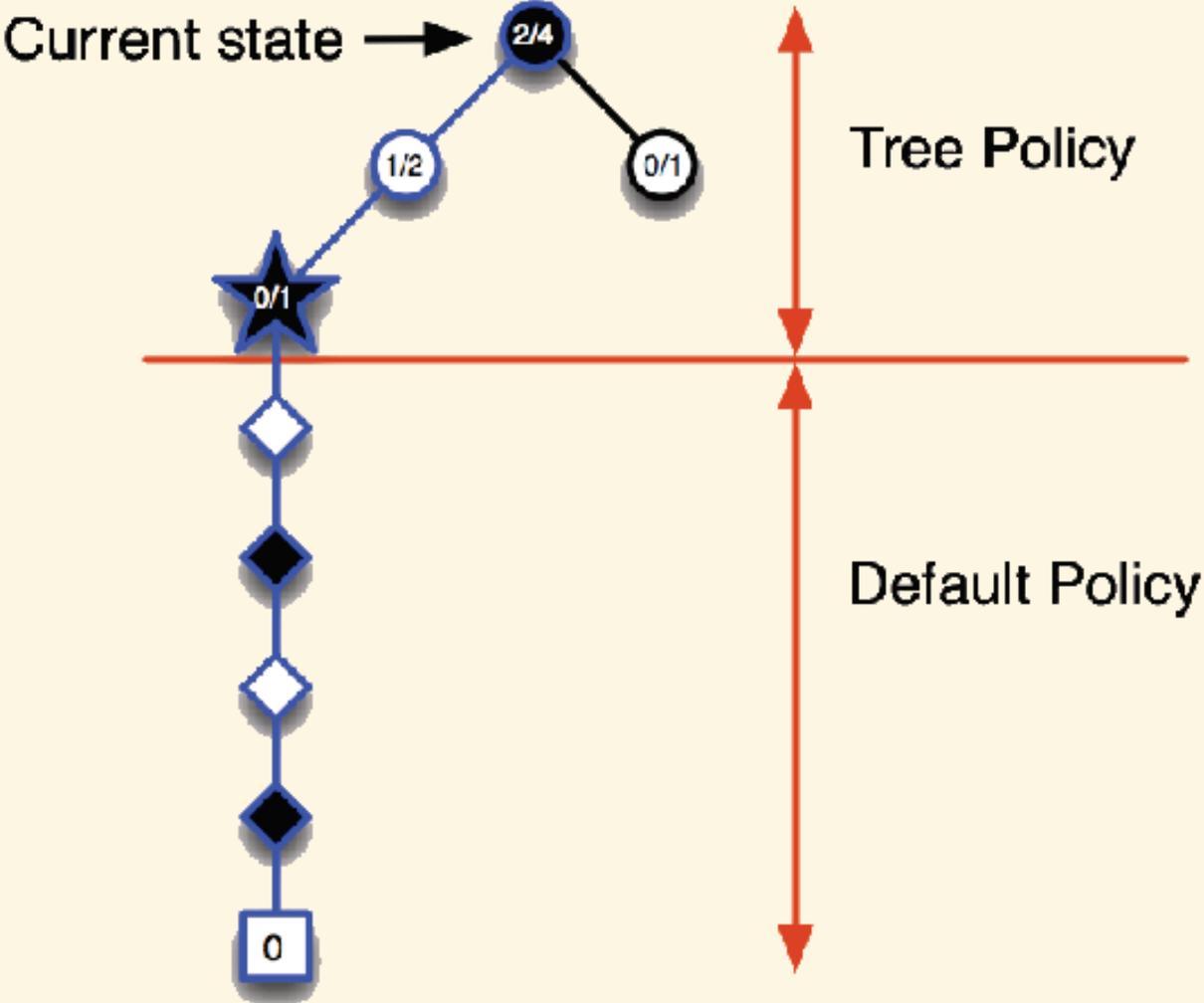
Applying Monte-Carlo Tree Search (2)



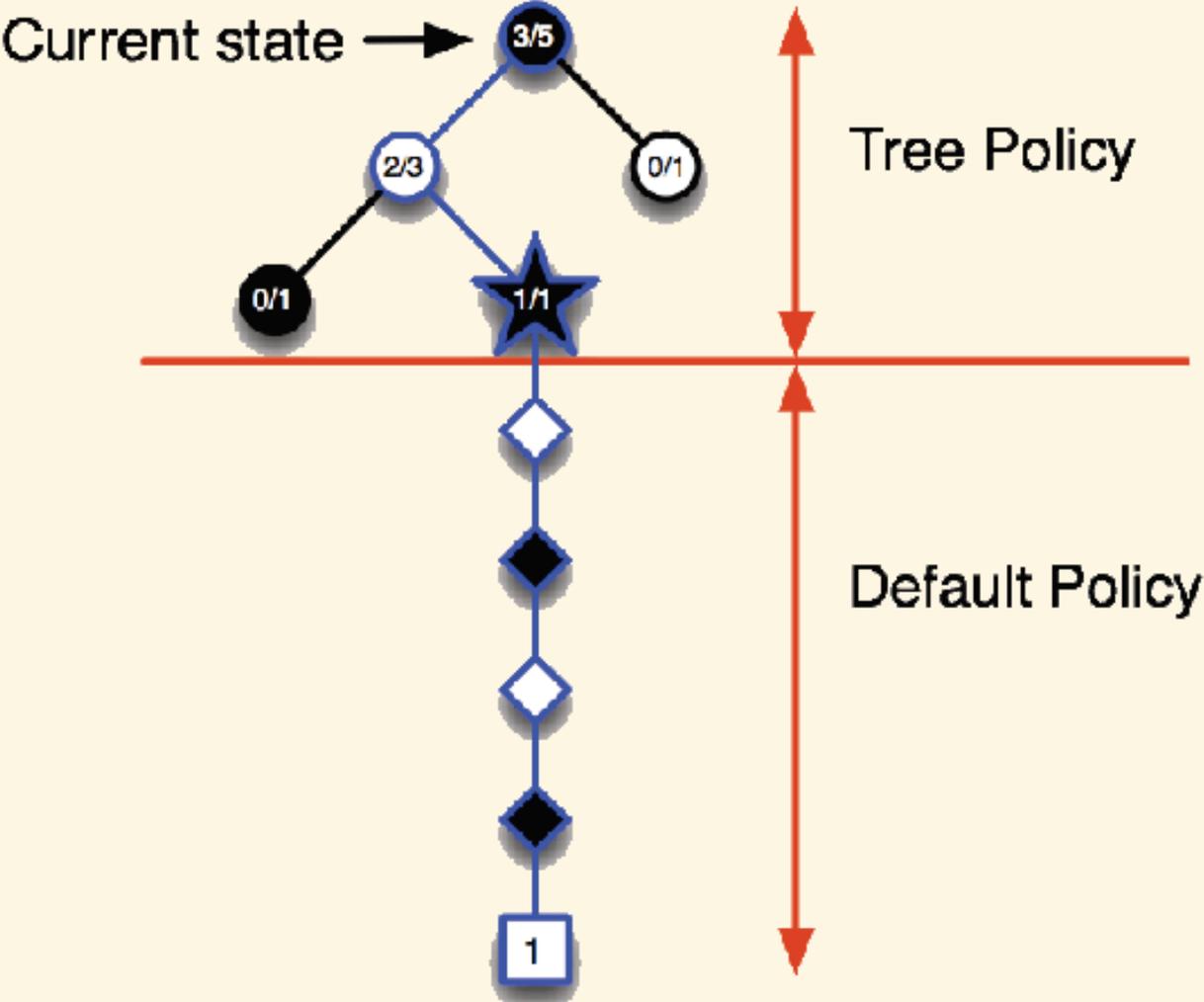
Applying Monte-Carlo Tree Search (3)



Applying Monte-Carlo Tree Search (4)



Applying Monte-Carlo Tree Search (5)



Advantages of MC Tree Search

Highly selective best-first search

- Evaluates states *dynamically* (unlike e.g. dynamic programming which does not use trees)
- Uses **sampling** to break curse of dimensionality

Works for “black-box” models (only requires samples)

- Computationally efficient, anytime, parallelisable