

Foundations of Artificial Intelligence

G7. Board Games: Monte-Carlo Tree Search Variants

Malte Helmert

University of Basel

May 18, 2026

Foundations of Artificial Intelligence

May 18, 2026 — G7. Board Games: Monte-Carlo Tree Search Variants

G7.1 Simulation Phase

G7.2 Tree Policy

G7.3 Tree Policy: Examples

G7.4 Comparison of Game Algorithms

G7.5 Summary

Board Games: Overview

chapter overview:

- ▶ G1. Introduction and State of the Art
- ▶ G2. Formal Definition and Minimax Search
- ▶ G3. Evaluation Functions
- ▶ G4. Alpha-Beta Search
- ▶ G5. Stochastic Games
- ▶ G6. Monte-Carlo Tree Search Framework
- ▶ G7. Monte-Carlo Tree Search Variants

Monte-Carlo Tree Search: Pseudo-Code

```
function visit_node(n)
if is_terminal(n.position):
    utility := utility(n.position)
else:
    s := n.get_unvisited_successor()
    if s is none:
        n' := apply_tree_policy(n)
        utility := visit_node(n')
    else:
        utility := simulate_game(s)
        n.add_and_initialize_child_node(s, utility)
n.N := n.N + 1
n. $\hat{v}$  := n. $\hat{v}$  +  $\frac{utility - n.\hat{v}}{n.N}$ 
return utility
```

G7.1 Simulation Phase

Simulation Phase

idea: determine **initial utility estimate** by **simulating game** following a **default policy**

Definition (default policy)

Let $\mathcal{S} = \langle S, A, T, s_I, S_G, utility, player \rangle$ be a game.

A **default policy** for \mathcal{S} is a mapping $\pi_{\text{def}} : S \times A \mapsto [0, 1]$ s.t.

- 1 $\pi_{\text{def}}(s, a) > 0$ implies that move a is applicable in position s
- 2 $\sum_{a \in A} \pi_{\text{def}}(s, a) = 1$ for all $s \in S$

In the call to `simulate_game(s)`,

- ▶ the default policy is applied starting from position s (determining decisions **for both players**)
- ▶ until a terminal position s_G is reached
- ▶ and $utility(s_G)$ is returned.

Implementations

“standard” implementation: Monte-Carlo random walk

- ▶ in each position, select a move uniformly at random
- ▶ until a terminal position is reached
- ▶ policy very cheap to compute
- ▶ uninformed \rightsquigarrow often not sufficient for good results
- ▶ not always cheap to simulate

alternative: game-specific default policy

- ▶ hand-crafted or
- ▶ learned offline

Gelly and Silver, Combining Online and Offline Knowledge in UCT (ICML, 2007)

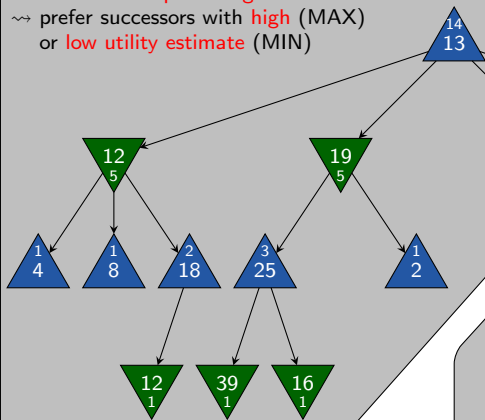
Default Policy vs. Evaluation Function

- ▶ default policy **simulates** a game to obtain utility estimate
~> default policy must be evaluated in many positions
- ▶ if default policy is **expensive to compute** or **poorly informed**, simulations are expensive
- ▶ **observe**: simulating a game to the end is just a **specific implementation** of an **evaluation function**
- ▶ many modern implementations replace default policy with **evaluation function** that **directly** computes a utility estimate
~> MCTS becomes a **heuristic search algorithm**

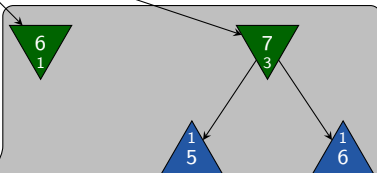
G7.2 Tree Policy

Objective of Tree Policy (1)

exploit collected information to
focus search on **promising areas**
 ~> prefer successors with **high (MAX)**
 or **low utility estimate (MIN)**

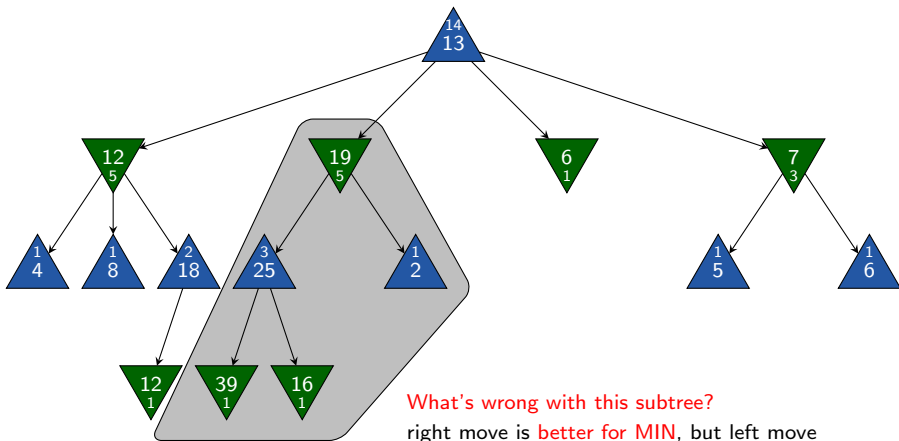


These are **contradictory objectives!**
 ~> **1st central challenge** for tree policy:
balance exploration and exploitation



explore areas that have
 not been **investigated thoroughly**
 ~> also consider other successors,
 in particular with **low visit count**

Objective of Tree Policy (2)



What's wrong with this subtree?

right move is **better for MIN**, but left move has **higher influence** on utility estimate

↪ **2nd central challenge** for tree policy:
exploit much more often than explore
 (in the limit)

Asymptotic Optimality

Definition (asymptotic optimality)

Let \mathcal{S} be a game with set of positions S .

Let $v^*(s)$ denote the (true) utility of position $s \in S$.

Let $n.\hat{v}^k$ denote the utility estimate of a search node n after k trials.

An MCTS algorithm is **asymptotically optimal** if

$$\lim_{k \rightarrow \infty} n.\hat{v}^k = v^*(n.\text{position})$$

for all search nodes n .

Asymptotic Optimality

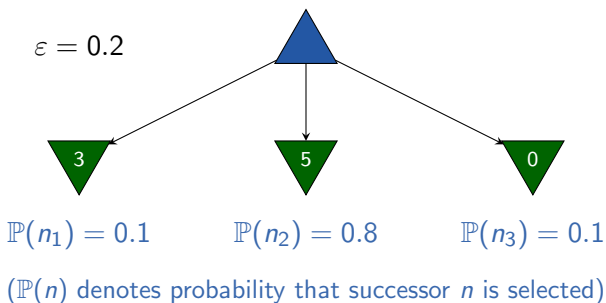
a tree policy is **asymptotically optimal** if

- ▶ it **explores forever**:
 - ▶ every position is **eventually added to the game tree** and **visited infinitely often**
(requires that the game tree is finite)
 - ↔ after a finite number of trials, all trials **end in a terminal position** and the **default policy** is no longer used
- ▶ and it is **greedy in the limit**:
 - ▶ the probability that an optimal move is selected converges to 1
 - ↔ in the limit, backups based on trials where only an **optimal policy** is followed dominate suboptimal backups

G7.3 Tree Policy: Examples

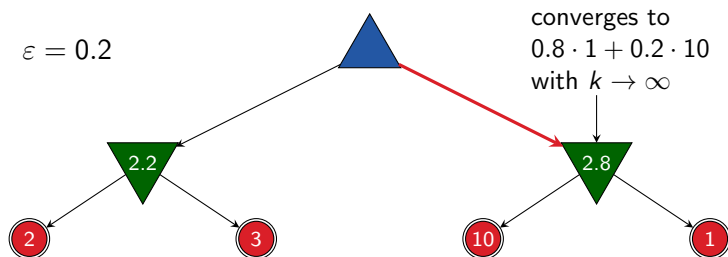
ϵ -greedy: Idea and Example

- ▶ tree policy with constant parameter ϵ
- ▶ with probability $1 - \epsilon$, pick a **greedy move** which leads to:
 - ▶ a successor with **highest utility estimate** (for MAX)
 - ▶ a successor with **lowest utility estimate** (for MIN)
- ▶ otherwise, pick a non-greedy successor **uniformly at random**



ϵ -greedy: Optimality

ϵ -greedy is **not asymptotically optimal**:

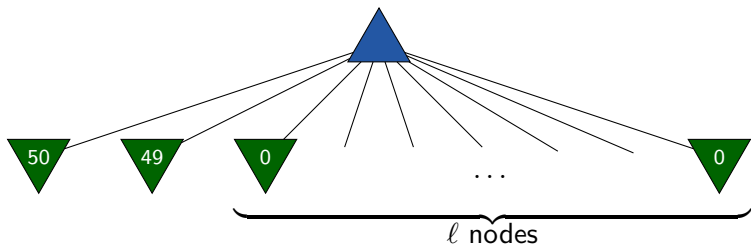


variants that are asymptotically optimal exist
 (e.g., **decaying ϵ** , **minimax backups**)

ϵ -greedy: Weakness

problem:

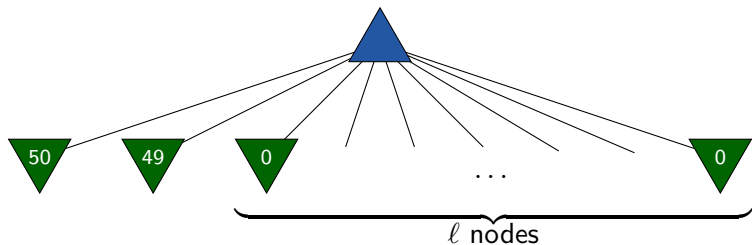
when ϵ -greedy explores, all non-greedy moves are treated **equally**



e.g., $\epsilon = 0.2, l = 9: \mathbb{P}(n_1) = 0.8, \mathbb{P}(n_2) = 0.02$

Softmax: Idea and Example

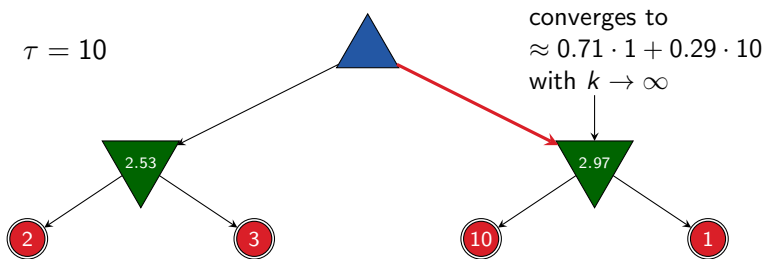
- ▶ tree policy with constant parameter $\tau > 0$
- ▶ select moves with a frequency that **directly relates** to their utility estimate
- ▶ **Boltzmann exploration** selects moves proportionally to $\mathbb{P}(n) \propto e^{\frac{n \cdot \hat{v}}{\tau}}$ for MAX and to $\mathbb{P}(n) \propto e^{\frac{-n \cdot \hat{v}}{\tau}}$ for MIN



e.g., $\tau = 10, \ell = 9$: $\mathbb{P}(n_1) \approx 0.51$, $\mathbb{P}(n_2) \approx 0.46$

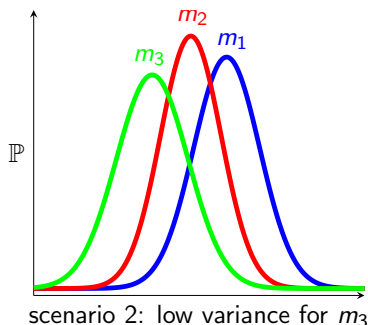
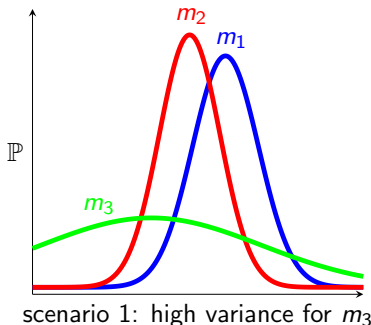
Boltzmann exploration: Optimality

Boltzmann exploration is **not asymptotically optimal**:



variants that are asymptotically optimal exist
 (e.g., **decaying τ** , **minimax backups**)

Boltzmann Exploration: Weakness



- ▶ Boltzmann exploration only considers **mean** of sampled utilities for the given moves
- ▶ as we sample the same node many times, we can also gather information about variance (how **reliable** the information is)
- ▶ Boltzmann exploration ignores the variance, treating the two scenarios equally

Upper Confidence Bounds: Idea

balance **exploration** and **exploitation** by preferring moves that

- ▶ have been **successful in earlier iterations** (exploit)
- ▶ have been **selected rarely** (explore)

Upper Confidence Bounds: Idea

upper confidence bound for MAX:

- ▶ select successor n' of n that maximizes $n'.\hat{v} + B(n')$
- ▶ based on **utility estimate** $n'.\hat{v}$
- ▶ and a **bonus term** $B(n')$
- ▶ select $B(n')$ such that $v^*(n'.\text{position}) \leq n'.\hat{v} + B(n')$ with high probability
- ▶ idea: $n'.\hat{v} + B(n')$ is an **upper confidence bound** on $n'.\hat{v}$ under the collected information

(for MIN: maximize $-n'.\hat{v} + B(n')$)

Upper Confidence Bounds: UCB1

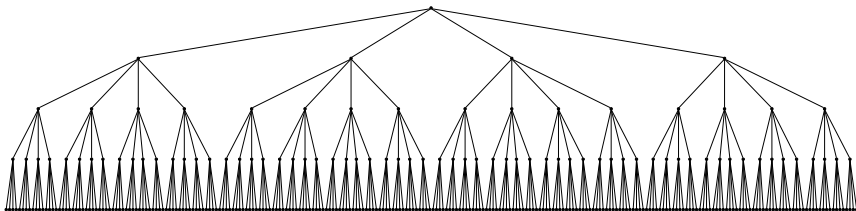
- ▶ use $B(n') = \sqrt{\frac{2 \cdot \ln n \cdot N}{n' \cdot N}}$ as bonus term
- ▶ bonus term is derived from **Chernoff-Hoeffding bound**, which
 - ▶ gives the probability that a **sampled value** (here: $n' \cdot \hat{v}$)
 - ▶ is far from its **true expected value** (here: $v^*(n'.\text{position})$)
 - ▶ in dependence of the **number of samples** (here: $n' \cdot N$)
- ▶ picks an optimal move **exponentially** more often in the limit

UCB1 is **asymptotically optimal**.

G7.4 Comparison of Game Algorithms

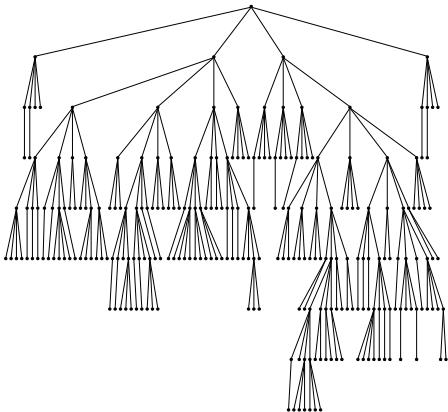
Minimax Tree

full tree up to depth 4



alpha-beta search with same effort:
↪ depth 6–8 with good move ordering

MCTS Tree



G7.5 Summary

Summary

- ▶ tree policy is crucial for MCTS
 - ▶ ϵ -greedy favors greedy moves and treats all others equally
 - ▶ Boltzmann exploration selects moves proportionally to an exponential function of their utility estimates
 - ▶ UCB1 favors moves that were successful in the past or have been explored rarely
- ▶ for each, there are applications where they perform best
- ▶ good default policies are domain-dependent and hand-crafted or learned offline
- ▶ using evaluation functions instead of a default policy often pays off