

Foundations of Artificial Intelligence

G6. Board Games: Monte-Carlo Tree Search Framework

Malte Helmert

University of Basel

May 18, 2026

Foundations of Artificial Intelligence

May 18, 2026 — G6. Board Games: Monte-Carlo Tree Search Framework

G6.1 Introduction

G6.2 Monte-Carlo Tree Search

G6.3 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

G6.1 Introduction

Monte-Carlo Tree Search

algorithms considered previously:

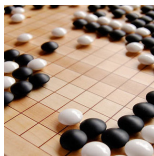
13	2	3	12
9	11	1	10
	6	4	14
15	8	7	5

systematic search:

- ▶ **systematic exploration** of search space
- ▶ **computation** of (state) quality follows **performance metric**



algorithms considered today:



search based on **Monte-Carlo methods**:

- ▶ **sampling of game simulations**
- ▶ **estimation** of (state) quality by **averaging** over simulation results



Game Applications

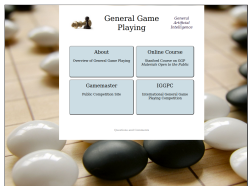
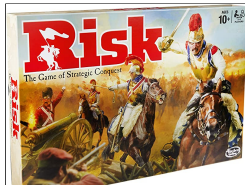
board games



hidden information games



stochastic games



general game playing



real-time strategy games



dynamic difficulty adjustment

Świechowski et al., Monte Carlo Tree Search: a review of recent modifications and applications (2023)

Applications Beyond Games

story generation



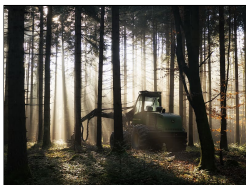
chemical synthesis



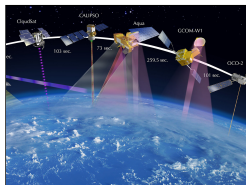
UAV routing



coast security



forest harvesting



Earth observation

Świechowski et al., Monte Carlo Tree Search: a review of recent modifications and applications (2023)

MCTS Environments

MCTS environments cover **entire spectrum of properties**.

We study MCTS under the **same restrictions** as before, i.e.,

- ▶ environment classification,
- ▶ problem solving method,
- ▶ objective of the agent and
- ▶ performance measure

are identical to Chapters G1–G4.

MCTS extensions exist that allow us to **drop most restrictions**.

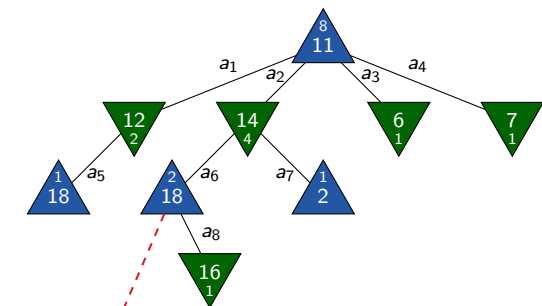
G6.2 Monte-Carlo Tree Search

Data Structures

Monte-Carlo tree search


- ▶ is a **tree search** variant
 - ↪ **no closed list**
 - ▶ iteratively performs **game simulations** from the initial position (called **trial** or **rollout**)
 - ↪ **no (explicit) open list**
- ↪ **MCTS nodes** are the only used data structure

Data Structure: MCTS Nodes



MCTS nodes store

- ▶ a reached **position**
- ▶ **how** it was reached
- ▶ its **successors**
- ▶ a **utility estimate** (\hat{v})
- ▶ a **visit counter** (N)
- ▶ possibly additional information

<i>position:</i>	not displayed
<i>move:</i>	a_6
<i>successors:</i>	[none , ]
\hat{v} :	18
N :	2
....:	...

Monte-Carlo Tree Search: Idea

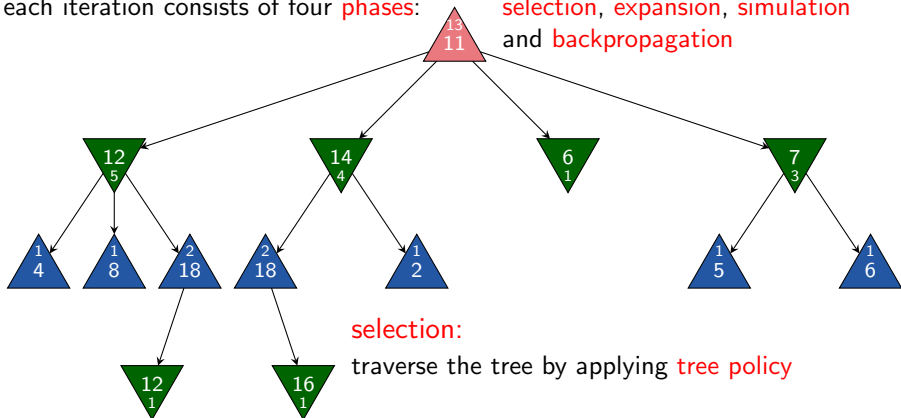
Monte-Carlo Tree Search (MCTS) ideas:

- ▶ build a partial game tree
- ▶ by performing trials as long as resources (deliberation time, memory) allow
- ▶ initially, the tree contains only the root node
- ▶ each trial adds (at most) one node to the tree

after termination, play the associated move of a successor of the root node with highest utility estimate

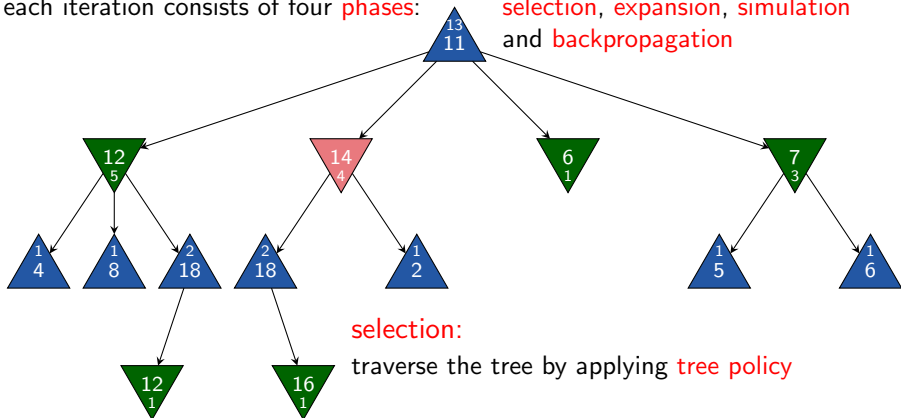
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



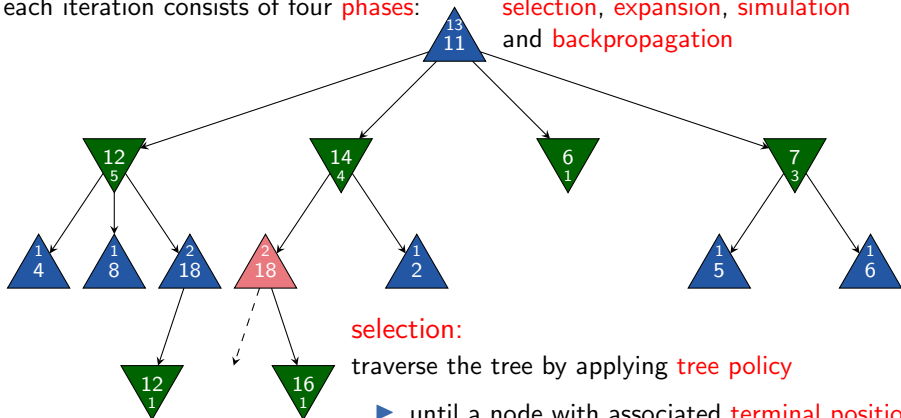
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



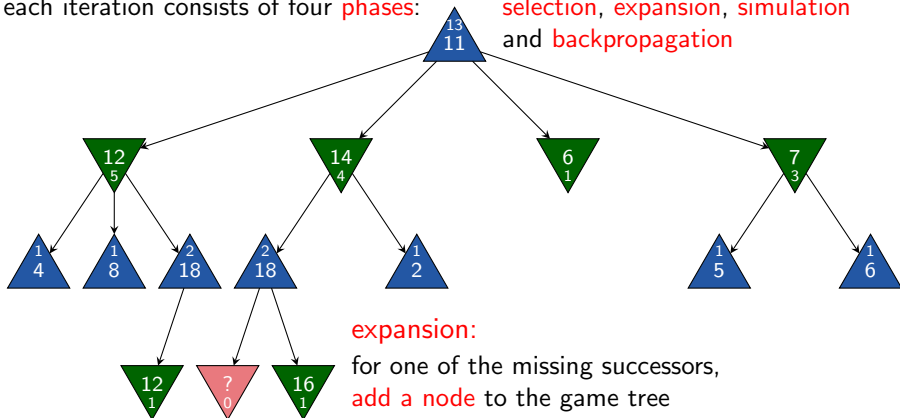
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



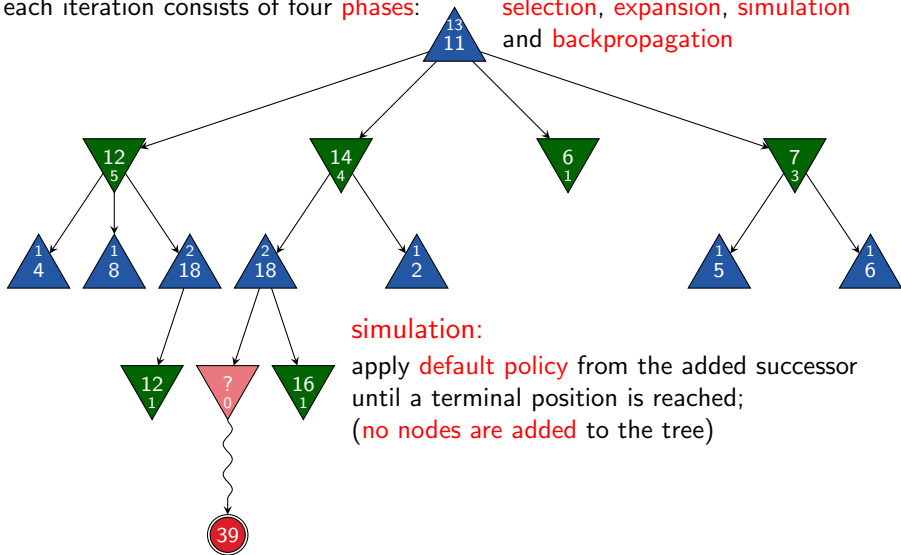
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



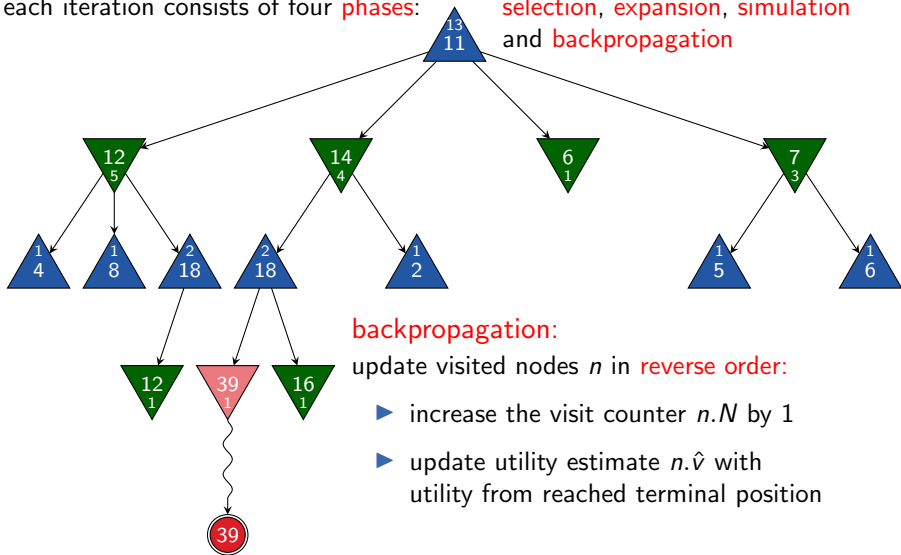
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



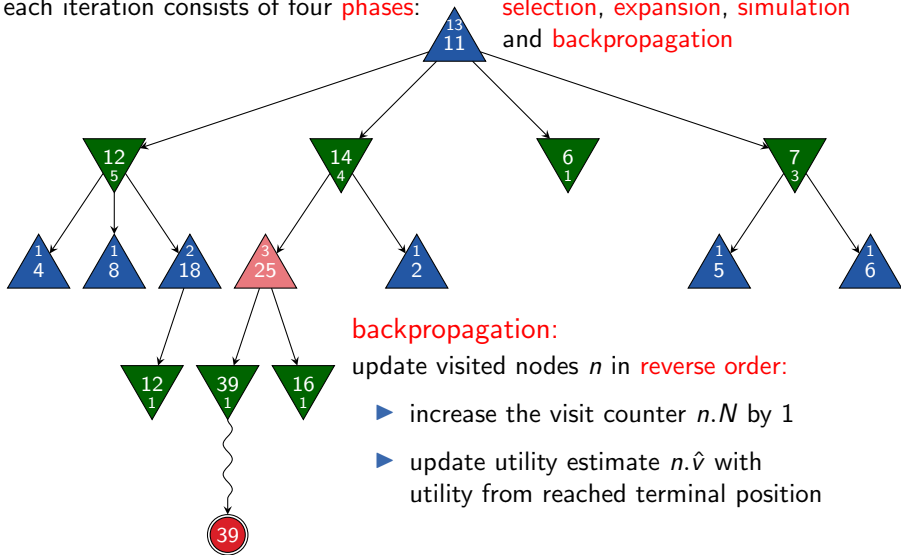
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



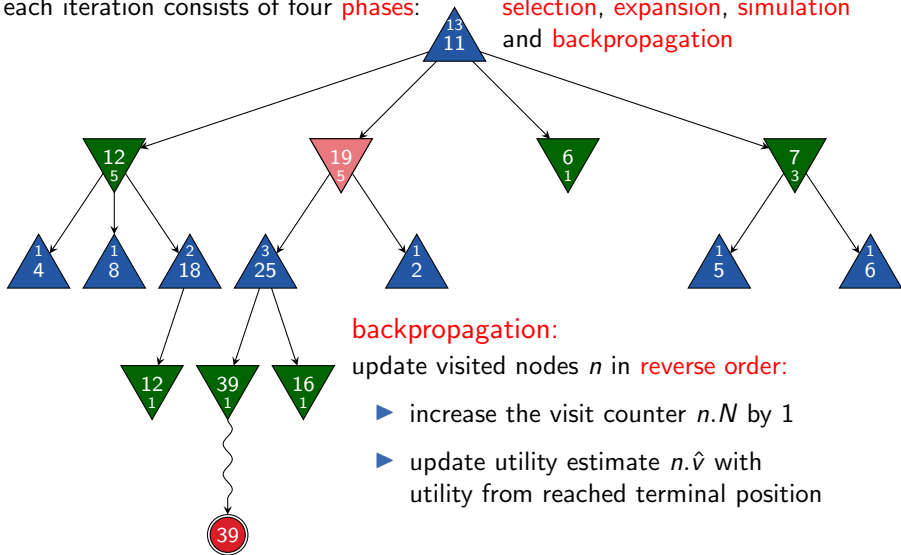
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



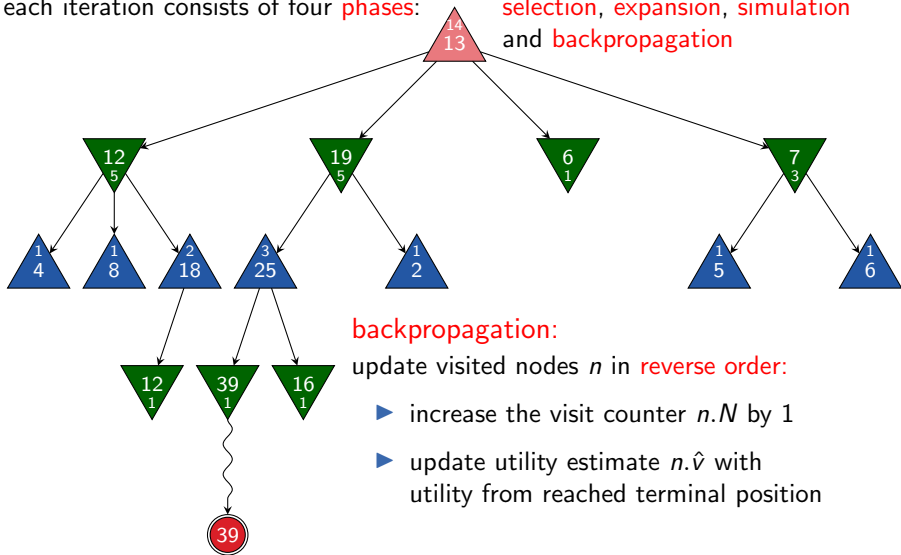
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



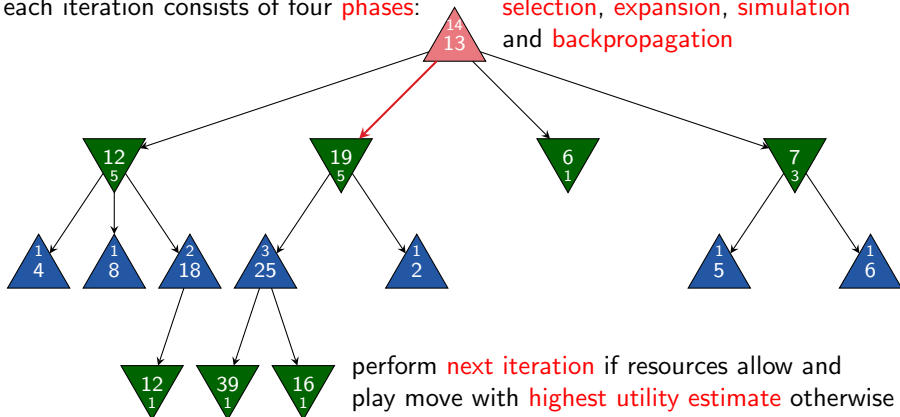
Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



Idea and Example

each iteration consists of four **phases**: **selection**, **expansion**, **simulation** and **backpropagation**



Monte-Carlo Tree Search: Pseudo-Code

Monte-Carlo Tree Search

$n_0 := \text{create_root_node}()$

while $\text{time_allows}()$:

$\text{visit_node}(n_0)$

$n_{\text{best}} := \arg \max_{n \in \text{succ}(n_0)} n.\hat{v}$

return $n_{\text{best}}.\text{move}$

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
  
```

G6.3 Summary

Summary

- ▶ Monte-Carlo methods compute **averages** over a number of random **samples**.
- ▶ **Monte-Carlo Tree Search (MCTS)** algorithms **simulate** a playout of the game
- ▶ and iteratively build a search tree, adding (at most) one node in each iteration.
- ▶ MCTS is parameterized by a **tree policy** and a **default policy**.