Foundations of Artificial Intelligence

44. Board Games: Monte-Carlo Tree Search Framework

Thomas Keller and Florian Pommerening

University of Basel

May 24, 2023

Board Games: Overview

chapter overview:

- 40. Introduction and State of the Art
- 41. Minimax Search and Evaluation Functions
- 42. Alpha-Beta Search
- 43. Stochastic Games
- 44. Monte-Carlo Tree Search Framework
- 45. Monte-Carlo Tree Search Configurations

44.1 Introduction

Monte-Carlo Tree Search

algorithms considered previously:

systematic search:

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

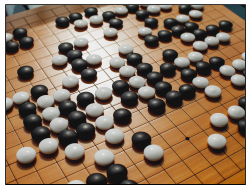
algorithms considered today:

search based on Monte-Carlo methods:

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

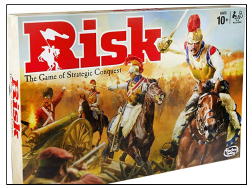
Game Applications

board games



hidden information games

stochastic games



general game playing

real-time strategy games

dynamic difficulty adjustment

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

Applications Beyond Games

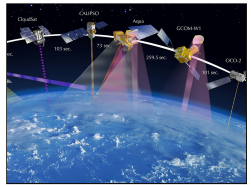
story generation

chemical synthesis

UAV routing

coast security

forest harvesting



Earth oberservation

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

MCTS Environments

MCTS environments cover entire spectrum of properties
$\rightsquigarrow$ need some restrictions

we study MCTS under the same restrictions as last week, i.e.,

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

are identical to last week

MCTS extensions exist that allow to drop most restrictions

44.2 Monte-Carlo Tree Search

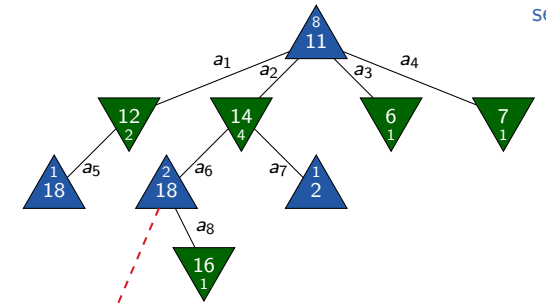
Data Structures

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

⇝ search nodes are the only used data structure

Data Structure: Search Nodes



search nodes store

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

position:	not displayed
move:	a_6
successors:	[**none**, 16 1]
$\hat{v}$:	18
N:	2
...:	...

Monte-Carlo Tree Search: Idea

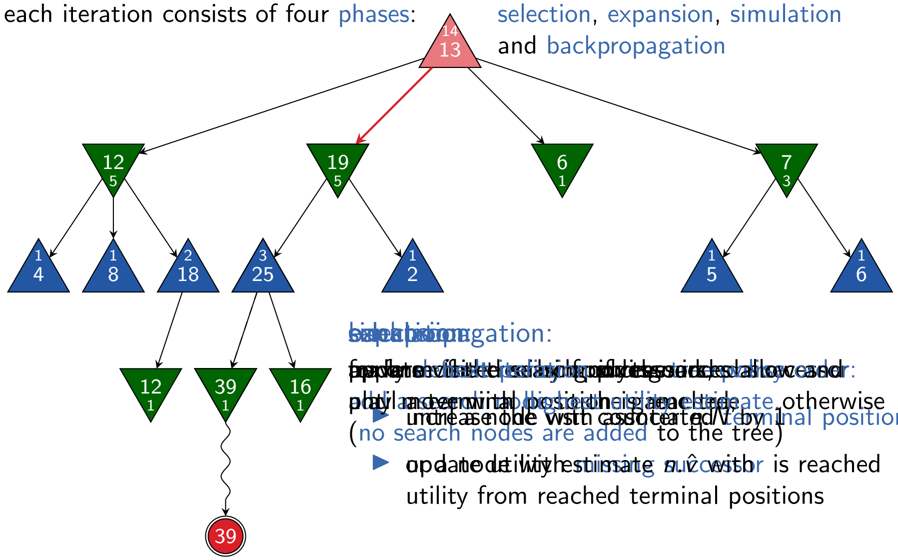
Monte-Carlo Tree Search (MCTS) ideas:

- 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 search 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



[illegible] utility from reached terminal positions

Monte-Carlo Tree Search: Pseudo-Code

Monte-Carlo Tree Search

n_0 := create_root_node()
while time_allows():
  visit_node(n_0)
$n_{\text{best}} := \arg\max_{n \in \text{succ}(n_0)} n.\hat{v}$
return n_{best}.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*

44.3 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.