Foundations of Artificial Intelligence

43. Monte-Carlo Tree Search: Introduction

Malte Helmert and Thomas Keller

University of Basel

May 18, 2020

Board Games: Overview

chapter overview:

- 40. Introduction and State of the Art
- 41. Minimax Search and Evaluation Functions
- 42. Alpha-Beta Search
- 43. Monte-Carlo Tree Search: Introduction
- 44. Monte-Carlo Tree Search: Advanced Topics
- 45. AlphaGo and Outlook

Introduction

Monte-Carlo Tree Search: Brief History

- Starting in the 1930s: first researchers experiment with Monte-Carlo methods
- 1998: Ginsberg's GIB player achieves strong performance playing Bridge
- 2002: Auer et al. present UCB1 action selection for multi-armed bandits
- 2006: Coulom coins the term Monte-Carlo Tree Search (MCTS)
- 2006: Kocsis and Szepesvári combine UCB1 and MCTS into the most famous MCTS variant, UCT

Monte-Carlo Tree Search: Brief History

- Starting in the 1930s: first researchers experiment with Monte-Carlo methods
- 1998: Ginsberg's GIB player achieves strong performance playing Bridge ⇝ this chapter
- 2002: Auer et al. present UCB1 action selection for multi-armed bandits ⇝ Chapter 44
- 2006: Coulom coins the term Monte-Carlo Tree Search (MCTS) ⇝ this chapter
- 2006: Kocsis and Szepesvári combine UCB1 and MCTS into the most famous MCTS variant, UCT ⇝ Chapter 44

Monte-Carlo Tree Search: Applications

Examples for successful applications of MCTS in games:

- board games (e.g., Go ⇝ Chapter 45)
- card games (e.g., Poker)
- AI for computer games
  (e.g., for Real-Time Strategy Games or Civilization)
- Story Generation
  (e.g., for dynamic dialogue generation in computer games)
- General Game Playing

Also many applications in other areas, e.g.,

- MDPs (planning with stochastic effects) or
- POMDPs (MDPs with partial observability)

Monte-Carlo Methods

Monte-Carlo Methods: Idea

- subsume a broad family of algorithms
- decisions are based on random samples
- results of samples are aggregated by computing the average
- apart from these points, algorithms differ significantly

Aside: Hindsight Optimization vs. the Exam

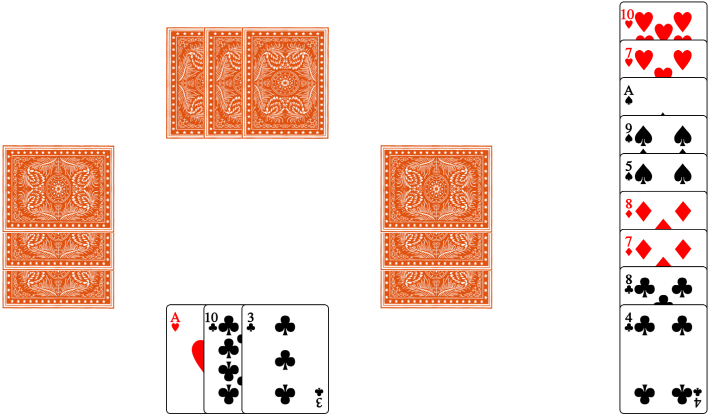
- As a motivating example for Monte-Carlo methods, we now briefly look at hindsight optimization.
- Hindsight optimization is interesting for settings with randomness and partial observability, which we do not otherwise consider in this lecture.
- To keep the discussion short, we do not provide formal details for how to model randomness and partial observability.
- Therefore, the slides on hindsight optimization are not relevant for the exam.

Monte-Carlo Methods: Example

Bridge Player GIB, based on Hindsight Optimization (HOP)

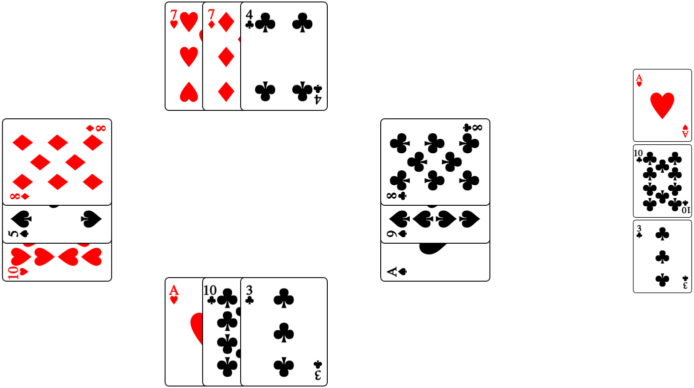
- perform samples as long as resources (deliberation time, memory) allow:
- sample hands for all players that are consistent with current knowledge about the game state
- for each legal move, compute if fully observable game that starts with executing that move is won or lost
- compute win percentage for each move over all samples
- play the card with the highest win percentage

Hindsight Optimization: Example



South to play, three tricks to win, trump suit ♣

Hindsight Optimization: Example



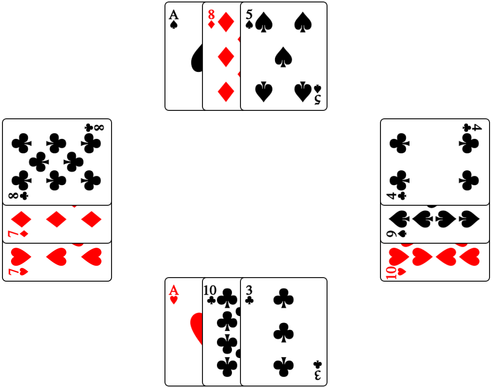
0% (0/1)

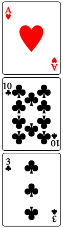
100% (1/1)

0% (0/1)

South to play, three tricks to win, trump suit ♣

Hindsight Optimization: Example





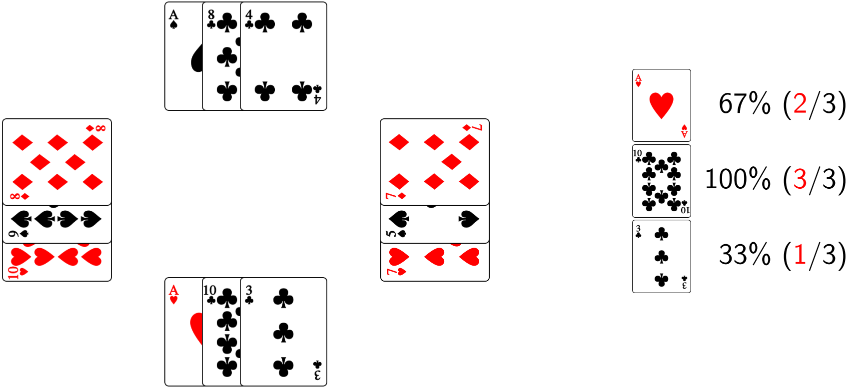
50% (1/2)

100% (2/2)

0% (0/2)

South to play, three tricks to win, trump suit ♣

Hindsight Optimization: Example

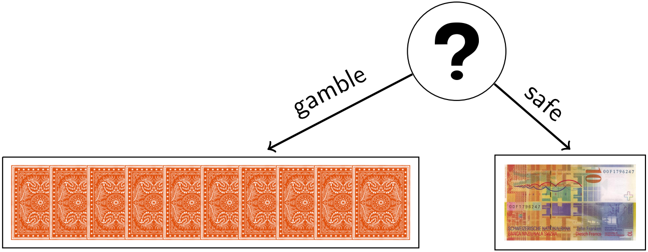


South to play, three tricks to win, trump suit ♣

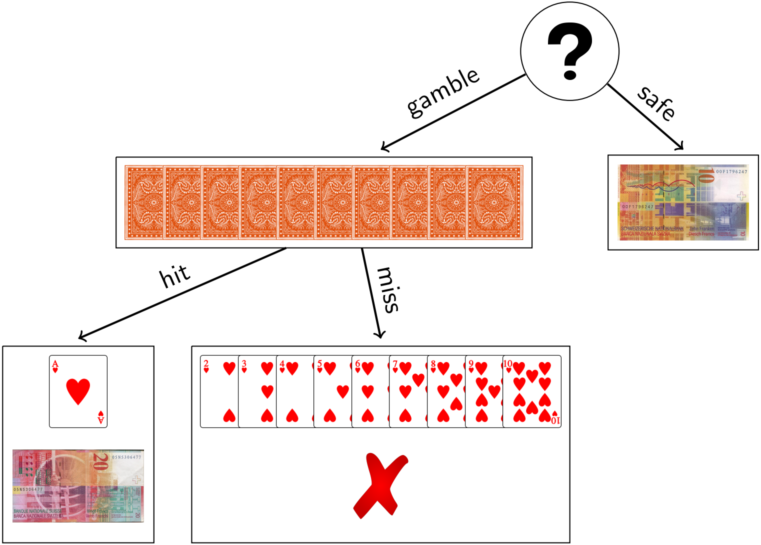
Hindsight Optimization: Restrictions

- HOP well-suited for partially observable games like most card games (Bridge, Skat, Klondike Solitaire)
- must be possible to solve or approximate sampled game efficiently
- often not optimal even if provided with infinite resources

Hindsight Optimization: Suboptimality



Hindsight Optimization: Suboptimality



Monte-Carlo Tree Search

Monte-Carlo Tree Search: Idea

Monte-Carlo Tree Search (MCTS) ideas:

- perform iterations as long as resources (deliberation time, memory) allow:
- build a partial game tree, where nodes n are annotated with
  - utility estimate $\hat{u}(n)$
  - visit counter $N(n)$
- initially, the tree contains only the root node
- each iteration adds one node to the tree

After constructing the tree, play the move that leads to the child of the root with highest utility estimate (as in minimax/alpha-beta).

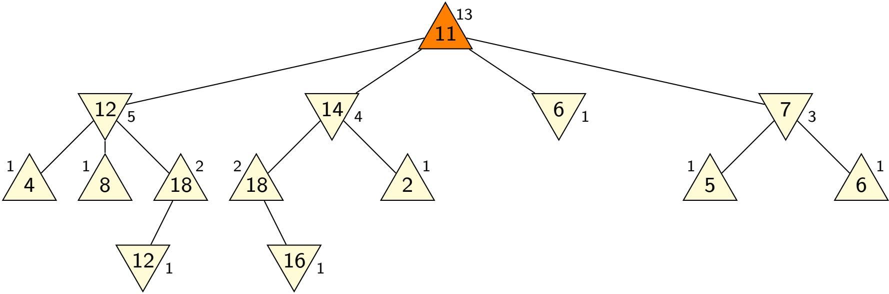
Monte-Carlo Tree Search: Iterations

Each iteration consists of four phases:

- selection: traverse the tree by applying tree policy
  - Stop when reaching terminal node (in this case, set n_{child} to that node and $p_\star$ to its position and skip next two phases)...
  - ...or when reaching a node n_{parent} for which not all successors are part of the tree.
- expansion: add a missing successor n_{child} of n_{parent} to the tree
- simulation: apply default policy from n_{child} until a terminal position $p_\star$ is reached
- backpropagation: for all nodes n on path from root to n_{child}:
  - increase $N(n)$ by 1
  - update current average $\hat{u}(n)$ based on $u(p_\star)$

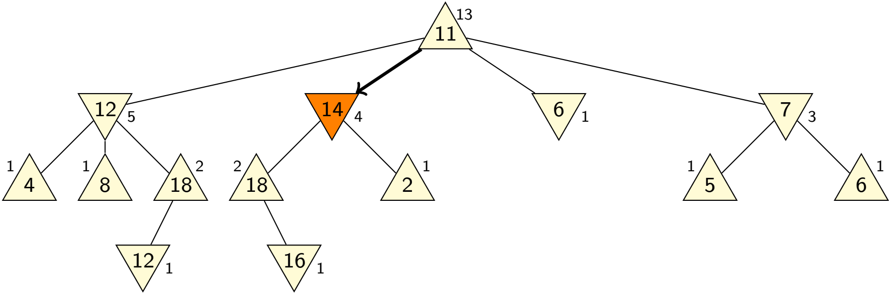
Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



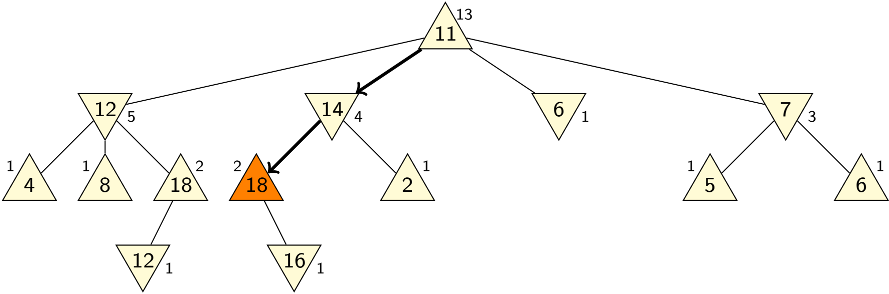
Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



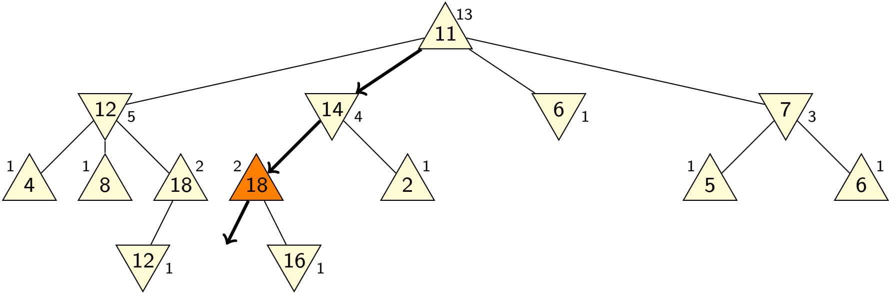
Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



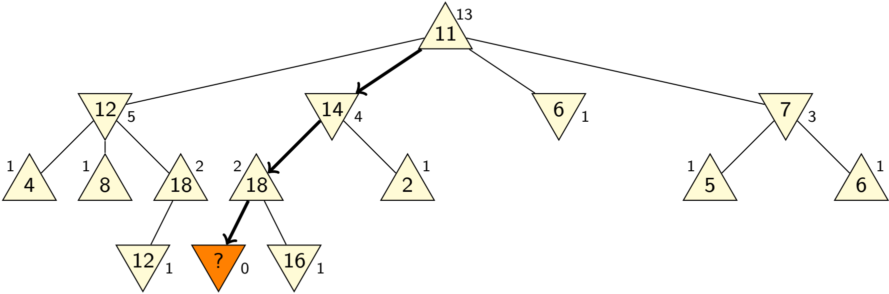
Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



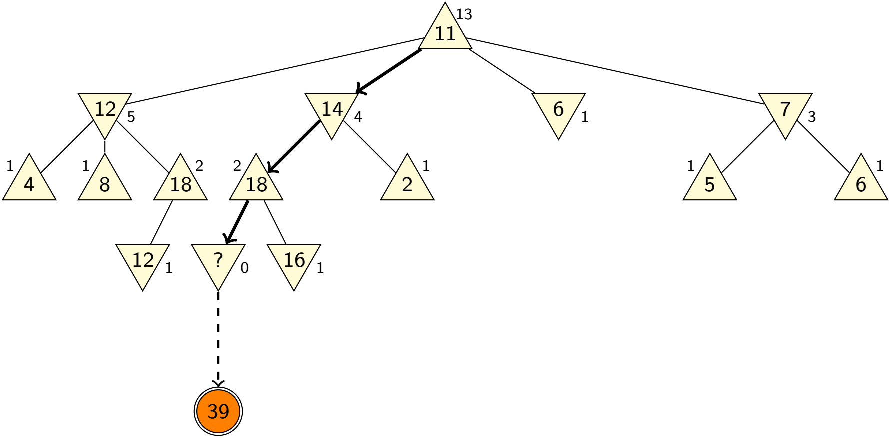
Monte-Carlo Tree Search

Expansion: create a node for first position beyond the tree



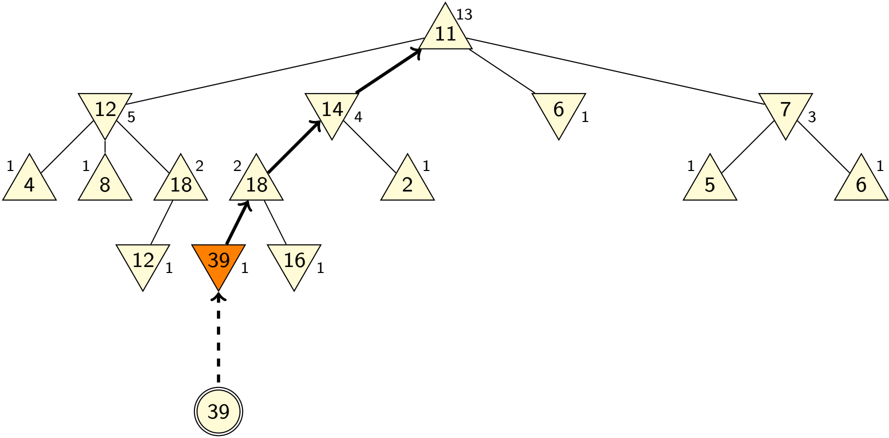
Monte-Carlo Tree Search

Simulation: apply default policy until terminal position is reached



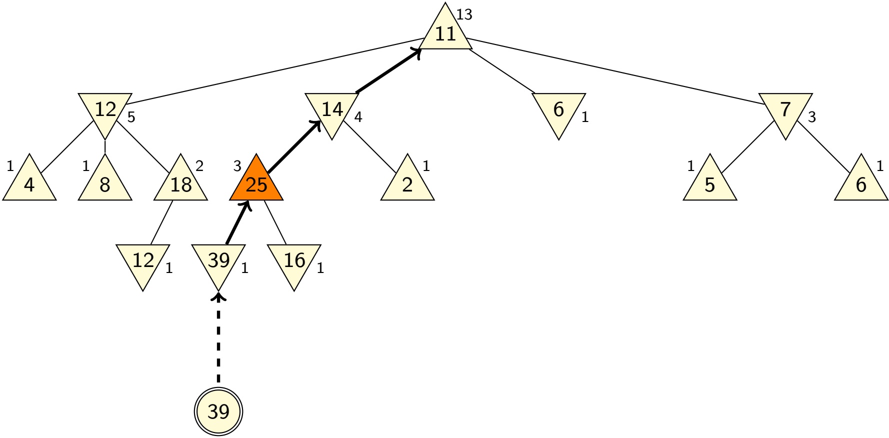
Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



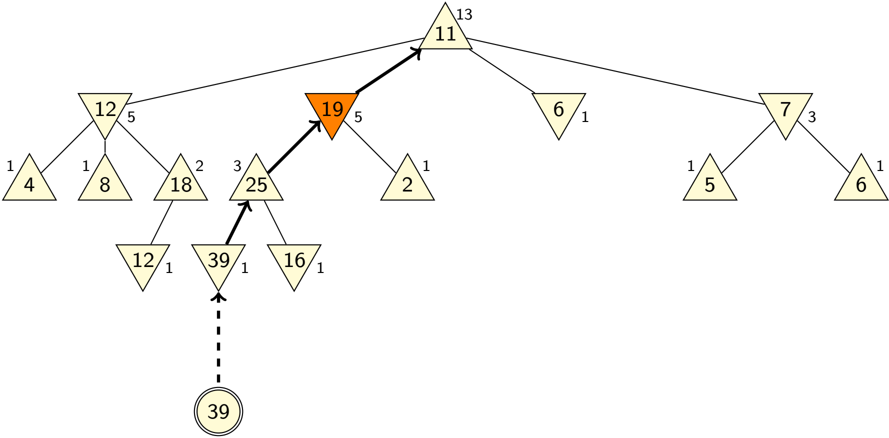
Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



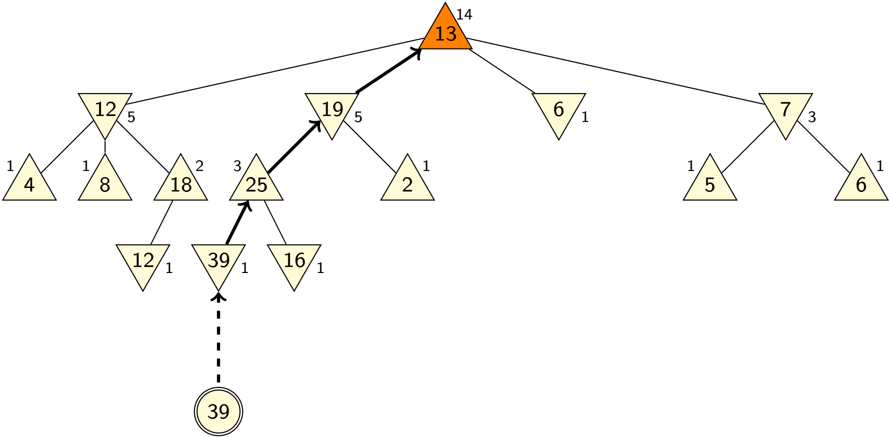
Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



Monte-Carlo Tree Search: Pseudo-Code

Monte-Carlo Tree Search

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

Monte-Carlo Tree Search: Pseudo-Code

function visit_node(n)

if is_terminal(n.position):
    $utility := u(n.\text{position})$
else:
    $p := n$.get_unvisited_successor()
    if p **is none**:
        $n' :=$ apply_tree_policy(n)
        $utility :=$ visit_node(n')
    else:
        $p_\star :=$ apply_default_policy_until_end(p)
        $utility := u(p_\star)$
        n.add_child_node(p, $utility$)
update_visit_count_and_estimate(n, $utility$)
return $utility$

Summary

Summary

- Monte-Carlo methods compute averages over a number of random samples.
- Simple Monte-Carlo methods like Hindsight Optimization perform well in some games, but are suboptimal even with unbounded resources.
- Monte-Carlo Tree Search (MCTS) algorithms iteratively build a search tree, adding one node in each iteration.
- MCTS is parameterized by a tree policy and a default policy.