

Foundations of Artificial Intelligence

43. Monte-Carlo Tree Search: Introduction

Malte Helmert and Gabriele Röger

University of Basel

May 17, 2017

Foundations of Artificial Intelligence

May 17, 2017 — 43. Monte-Carlo Tree Search: Introduction

43.1 Introduction

43.2 Monte-Carlo Methods

43.3 Sparse Sampling

43.4 MCTS

43.5 Summary

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

43.1 Introduction

Monte-Carlo Tree Search: Brief History

- ▶ Starting in the 1930s: first researchers experiment with **Monte-Carlo methods**
- ▶ 1998: Ginsberg's **GIB** player competes with expert Bridge players ~> [this chapter](#)
- ▶ 2002: Kearns et al. propose **Sparse Sampling** ~> [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 to 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**)

43.2 Monte-Carlo Methods

Monte-Carlo Methods: Idea

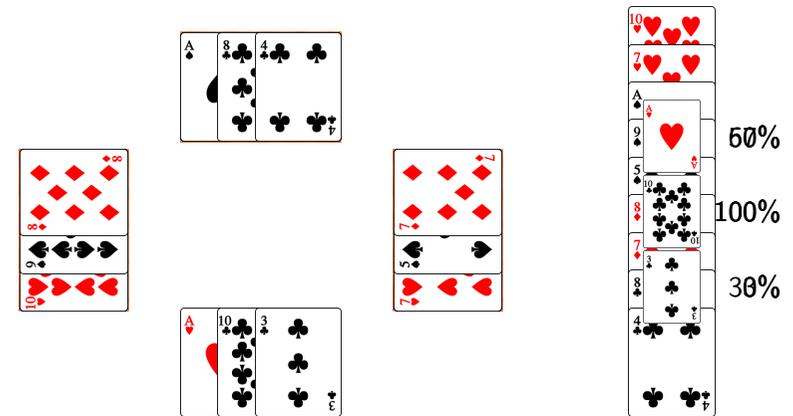
- ▶ summarize a broad **family of algorithms**
- ▶ decisions are based on **random samples**
- ▶ results of samples are **aggregated** by computing the **average**
- ▶ apart from that, algorithms can **differ** significantly

Monte-Carlo Methods: Example

Bridge Player GIB, based on **Hindsight Optimization** (HOP)

- ▶ perform **samples** as long as **resources** (deliberation time, memory) allow:
- ▶ **sample** hand for all players that is consistent with current knowledge about the game state
- ▶ for each legal action, compute if **perfect information** game that starts with executing that action is won or lost
- ▶ compute **win percentage** for each action over all samples
- ▶ play the card with the highest win percentage

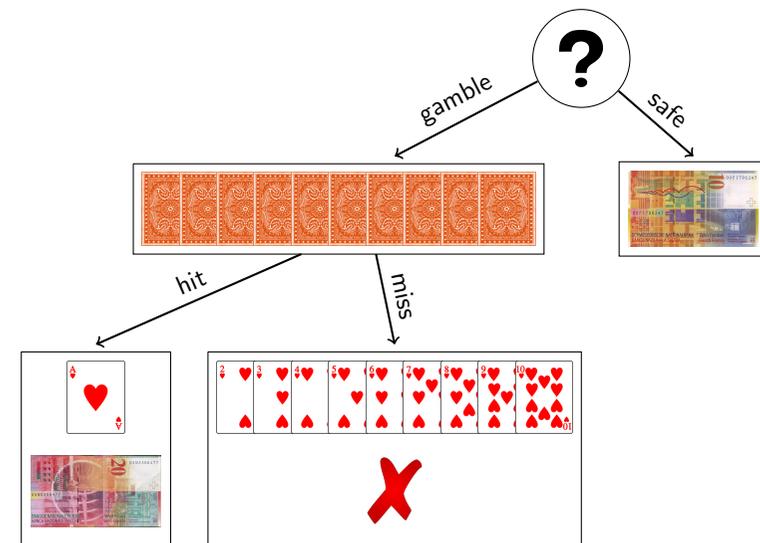
Hindsight Optimization: Example



Hindsight Optimization: Restrictions

- ▶ HOP **well-suited** for imperfect information 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

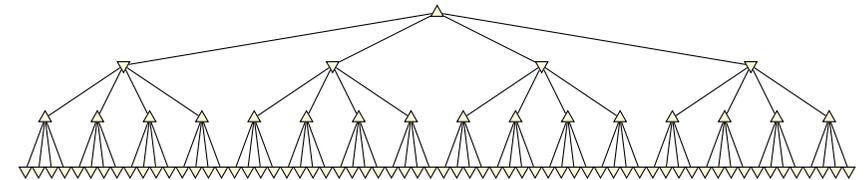
Hindsight Optimization: Suboptimality



43.3 Sparse Sampling

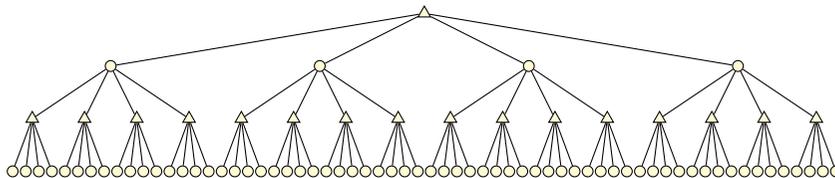
Reminder: Minimax for Games

Minimax: alternate maximization and minimization



Excursion: Expectimax for MDPs

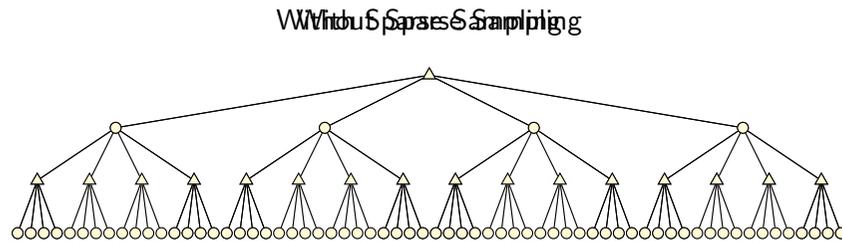
Expectimax: alternate maximization and expectation
(expectation = probability weighted sum)



Sparse Sampling: Idea

- ▶ search tree creation: **sample** a constant number of outcomes according to their probability in each state and **ignore** the rest
- ▶ update values by replacing probability weighted updates with average
- ▶ **near-optimal:** utility of resulting policy close to utility of optimal policy
- ▶ runtime **independent** from the number of states

Sparse Sampling: Search Tree



Sparse Sampling: Problems

- ▶ independent from number of states, but still **exponential in lookahead horizon**
- ▶ constant that gives the number of outcomes **large for good bounds on near-optimality**
- ▶ search time difficult to predict
- ▶ tree is **symmetric** \Rightarrow resources are **wasted** in non-promising parts of the tree

43.4 MCTS

Monte-Carlo Tree Search: Idea

- ▶ perform **iterations** as long as resources (deliberation time, memory) allow:
- ▶ **builds a search tree** of nodes n with annotated
 - ▶ **utility estimate** $\hat{Q}(n)$
 - ▶ **visit counter** $N(n)$
- ▶ initially, the tree contains only the root node
- ▶ execute the action that leads to the node with the **highest utility estimate**

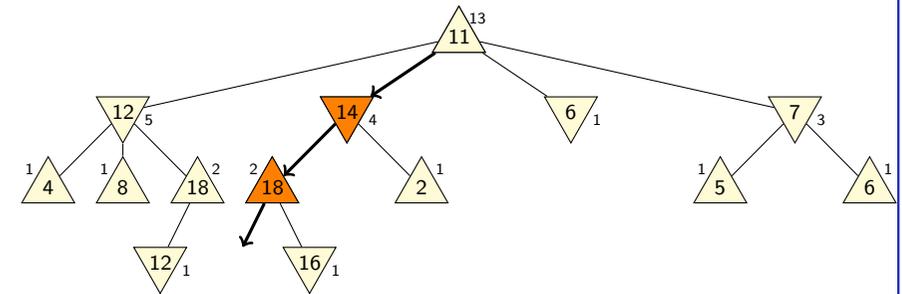
Monte-Carlo Tree Search: Iterations

Each iteration consist of four **phases**:

- ▶ **selection**: traverse the tree by applying **tree policy**
- ▶ **expansion**: add to the tree the first visited state that is not in the tree
- ▶ **simulation**: continue by applying **default policy** until terminal state is reached (which yields **utility** of current iteration)
- ▶ **backpropagation**: for all visited nodes n ,
 - ▶ increase $N(n)$
 - ▶ extend the current average $\hat{Q}(n)$ with yielded utility

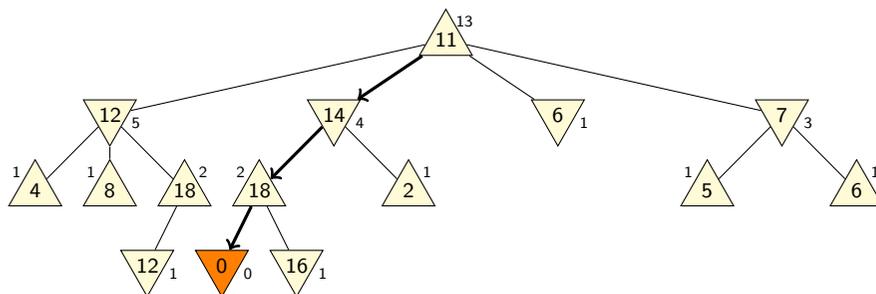
Monte-Carlo Tree Search

Selection: apply **tree policy** to traverse tree



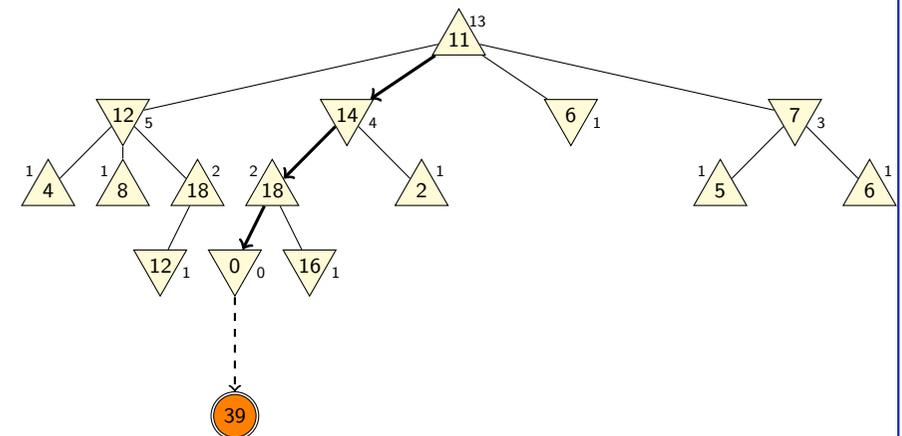
Monte-Carlo Tree Search

Expansion: create a node for **first state** beyond the tree



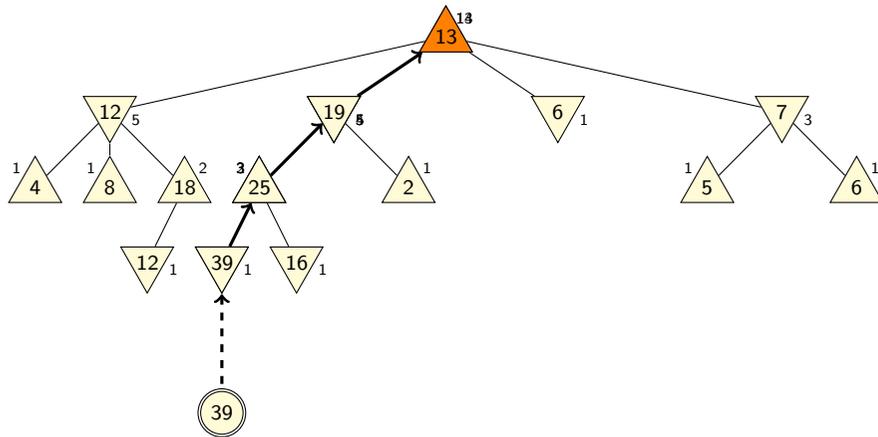
Monte-Carlo Tree Search

Simulation: apply **default policy** until terminal state is reached



Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



Monte-Carlo Tree Search: Pseudo-Code

Monte-Carlo Tree Search

```

tree := new SearchTree
n0 = tree.add_root_node()
while time_allows():
    visit_node(tree, n0)
n* = arg maxn ∈ succ(n0) Q̂(n)
return n*.get_action()

```

Monte-Carlo Tree Search: Pseudo-Code

```

function visit_node(tree, n)
if is_final(n.state):
    return u(n.state)
s = tree.get_unvisited_successor(n)
if s ≠ none:
    n' = tree.add_child_node(n, s)
    utility = apply_default_policy()
    backup(n', utility)
else:
    n' = apply_tree_policy(n)
    utility = visit_node(tree, n')
    backup(n, utility)
return utility

```

43.5 Summary

Summary

- ▶ Simple Monte-Carlo methods like **Hindsight Optimization** perform well in some games, but are suboptimal even with unbound resources
- ▶ **Sparse Sampling** allows near-optimal solutions independent of the state size, but it wastes time in non-promising parts of the tree
- ▶ **Monte-Carlo Tree Search** algorithms iteratively build a search tree. Algorithms are specified in terms of a tree policy and a default policy.