# Artificial Intelligence

## Adversarial Search: Monte-Carlo Tree Search

Jendrik Seipp

Linköping University

based on slides by Thomas Keller and Malte Helmert (University of Basel)

# Introduction

# 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

# Monte-Carlo Tree Search: Applications

Examples for successful applications of MCTS in games:

- board games (e.g., Go)
- card games (e.g., Poker)
- AI for computer games (e.g., Starcraft)
- Story Generation
  (e.g., for dynamic dialogue generation in computer games)
- General Game Playing

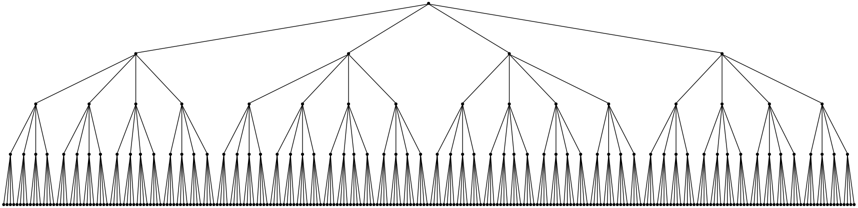
Also many applications in other areas, e.g.,

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

# Monte-Carlo Tree Search

# Minimax Tree

full tree up to depth 4



# 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 action that leads to the child of the root with highest utility estimate (as in minimax/alpha-beta).

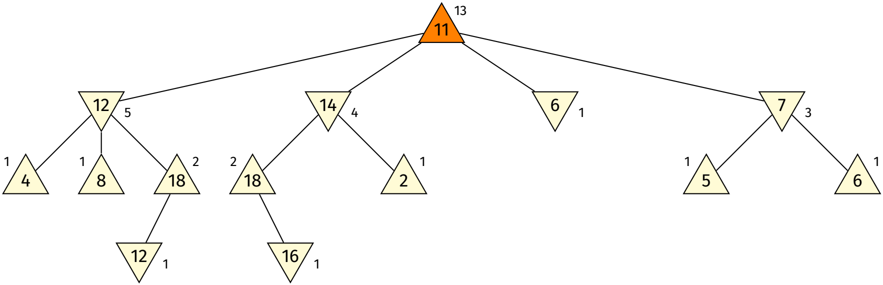
# Monte-Carlo Tree Search: Iterations

each iteration consists of four phases:

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

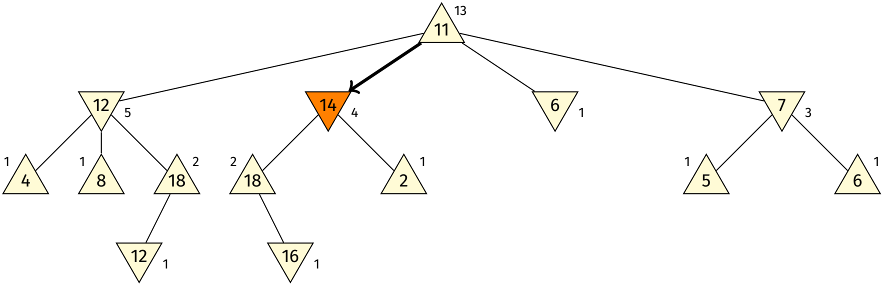
# Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



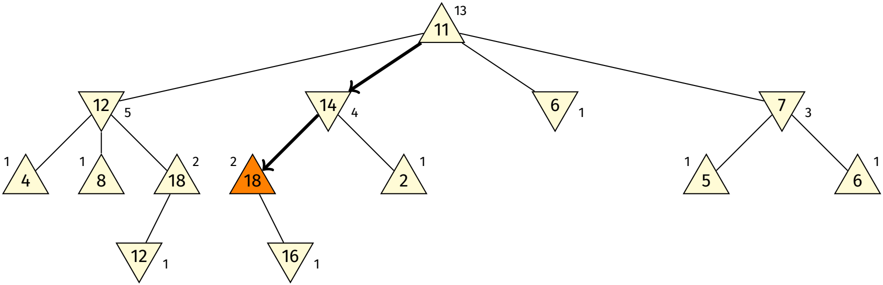
# Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



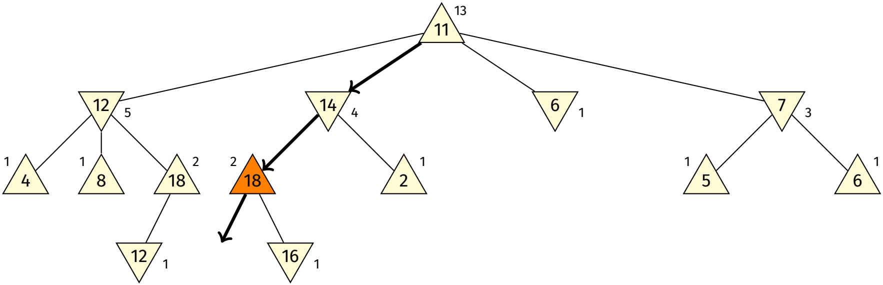
# Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



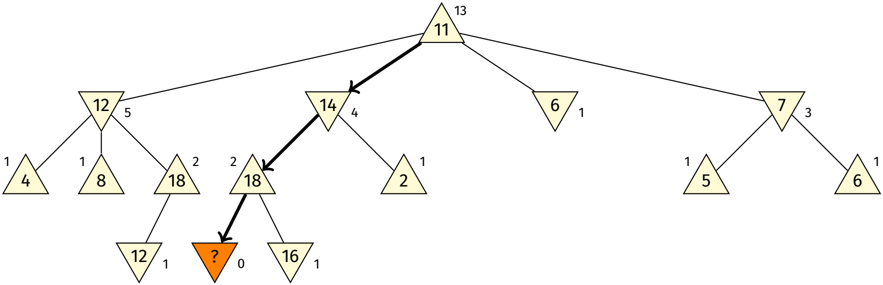
# Monte-Carlo Tree Search

Selection: apply tree policy to traverse tree



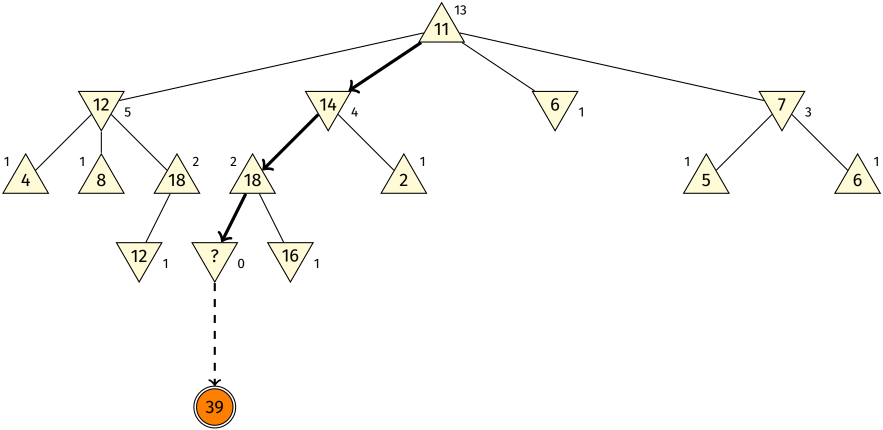
# Monte-Carlo Tree Search

Expansion: create a node for first state beyond the tree



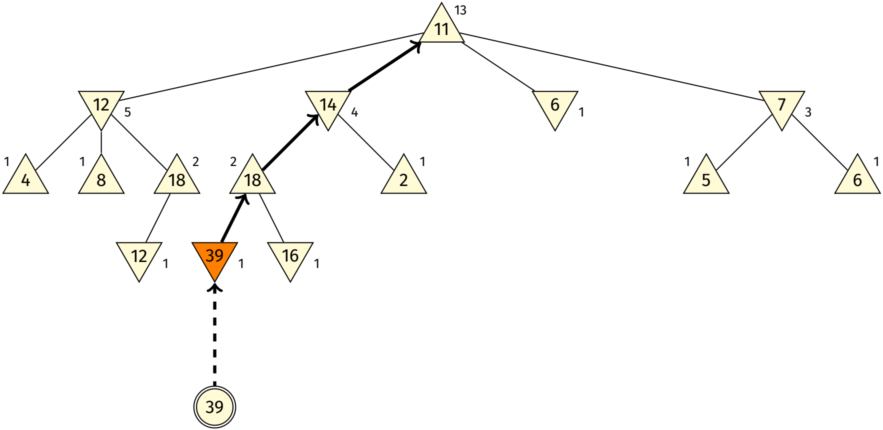
# Monte-Carlo Tree Search

Simulation: apply default policy until terminal state is reached



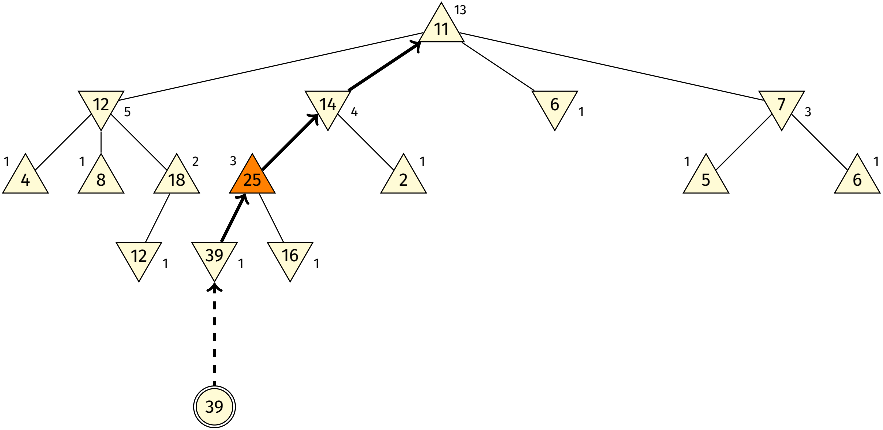
# Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



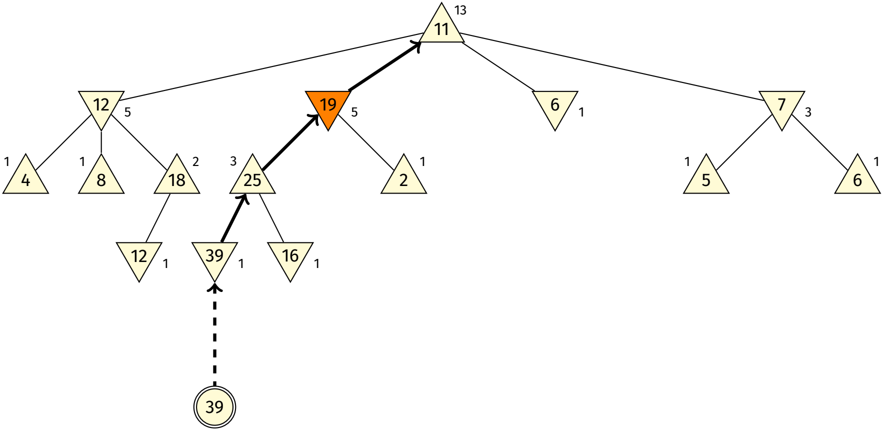
# Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



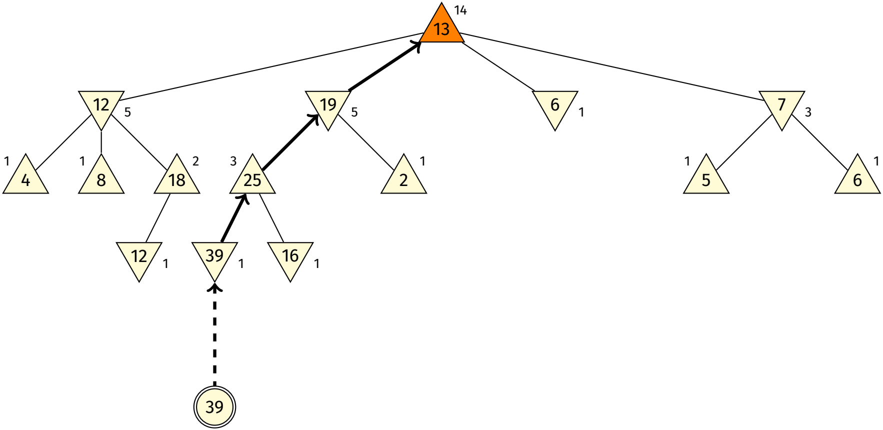
# Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



# Monte-Carlo Tree Search

Backpropagation: update utility estimates of visited nodes



# MCTS Tree