Seminar aus maschinellem Lernen

MCTS und UCT

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Übersicht

TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Historisches zu MCTS
- MCTS
- UCT
- Eigenschaften von MCTS
- Zusammenfassung

Historisches

TECHNISCHE UNIVERSITÄT DARMSTADT

2006 als Geburtsstunde von Monte Carlo Tree Search

Coulom Beschreibt in „Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search“ Monte Carlo Methoden für Baumsuche und gibt ihr den Namen „Monte Carlo Tree Search“

Koscic & Szepesvári nutzen UCB1 als Lösung für das Exploitation-Exploration Problem der Baumsuche

Gelly et al. benutzen UCT für ihr Go-Programm MoGo, das erste Programm, dass einen Profispieler in Go besiegt hat

Chaslot et al. nutzt MCTS zur Lösung von Production Management Problems (PMP)

MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorraussetzungen:

1. es existiert eine (einfache) Auswertungsfunktion für Terminalzustand
2. komplettes Regelwerk bekannt (complete information)
3. die Simulation terminiert schnell

Anwendung:

- großer branching Factor/ Suchraum
- Bewertungen von Zuständen schwer oder gar nicht berechenbar
- z.B. Go erfüllt diese Eigenschaft hervorragend

MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT

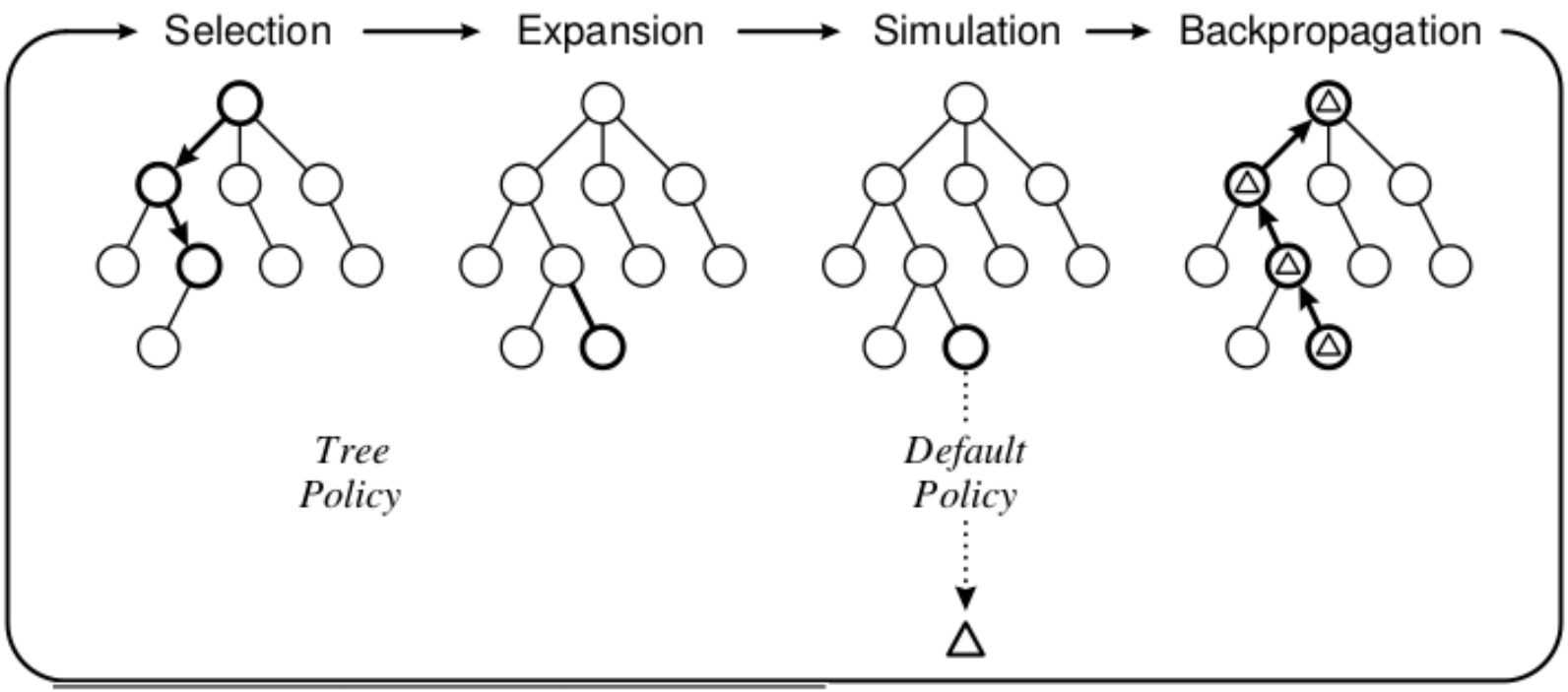
MTCS baut iterativ einen (zufälligen) Suchbaum auf und geht dabei mit den folgenden 4 Schritten vor:

1. Selection wähle den besten Kindknoten aus bis ein Knoten gefunden ist, der noch nicht besuchte Kindknoten hat oder Endknoten (Terminalzustand) ist. *Tree Policy*
2. Expansion wähle zufällig einen unbesuchten Kindknoten und erweitere den Baum damit.
3. Simulation generiere zufälliges Spiel bis Terminalzustand erreicht ist und werte diesen aus. *Default Policy*
4. Backpropagation reiche das Ergebnis an Elternknoten weiter und aktualisiere deren Zustand.

Knoten des Baumes entsprechen dabei dem Zustand/Stellung und die Kanten entsprechen Aktionen/Züge

MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT



Algorithm 1 General MCTS approach.

```
function MCTSSEARCH(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TREEPOLICY(v_0)
        Δ ← DEFAULTPOLICY(s(v_l))
        BACKUP(v_l, Δ)
    return a(BESTCHILD(v_0))
```

MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT

Mehrere Möglichkeiten für die Wahl des **BestChild**:

Max child	Knoten mit der höchsten Belohnung
Robust child	Knoten, der am häufigsten besucht wurde
Max-Robust child	Knoten, der am häufigsten besucht wurde und die höchste Belohnung verspricht. Wenn es keinen gibt, setze Suche fort bis akzeptable Lösung gefunden.
Secure child	Knoten, der eine untere Schranke maximiert

UCT

TECHNISCHE UNIVERSITÄT DARMSTADT

UCT (Upper Confidence Bounds for Trees) wurde 2006 von Koscic & Szepesvári vorgestellt mit Hinblick auf Markovian Decision Problems (MDP) und Spielbaumsuche

Ziel
- geringe Fehlerwahrscheinlichkeit, falls Algorithmus vorzeitig abgebrochen wird
- Konvergenz zur optimalen Lösung wenn genug Zeit zur Verfügung steht

Idee
- Performancegewinn durch Fokusierung auf starke Aktionen
- Nutzung/Kombination von UCB1 und Monto Carlo Methoden

UCT

TECHNISCHE UNIVERSITÄT DARMSTADT

UCT-Formel (maximiere):

$$UCT = \overline{X}_j + C_p \sqrt{\frac{2 \ln n}{n_j}}$$

mit:

$\overline{X}_j$ ist die durchschnittliche Belohnung von Knoten j
C_p ist eine Konstante > 0,
n Anzahl der Besuche durch Mutterknoten
n_j Anzahl der Besuche durch Knoten j

Exploitation-Term: $\overline{X}_j$
Exploration-Term: $C_p \sqrt{\frac{2 \ln n}{n_j}}$

UCT

TECHNISCHE UNIVERSITÄT DARMSTADT

Algorithm 2 The UCT algorithm.

function UCTSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD($v_0, 0$))

function TREEPOLICY(v)
  while v is nonterminal **do**
    if v not fully expanded **then**
      return EXPAND(v)
    else
      $v \leftarrow$ BESTCHILD(v, Cp)
  return v

function EXPAND(v)
  choose $a \in$ untried actions from $A(s(v))$
  add a new child v' to v
    with $s(v') = f(s(v), a)$
    and $a(v') = a$
  return v'

function BESTCHILD(v, c)
  return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

function DEFAULTPOLICY(s)
  while s is non-terminal **do**
    choose $a \in A(s)$ uniformly at random
    $s \leftarrow f(s, a)$
  return reward for state s

function BACKUP(v, Δ)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta(v, p)$
    $v \leftarrow$ parent of v

v bezeichnet Knoten
s bezeichnet Zustand eines Knoten (Stellung)
a bezeichnet eine Aktion (Kanten)
Q(*v*) erwartete Belohnung des Knoten (Anzahl der Siege)
N(*v*) Anzahl der Besuche durch den Knoten
Δ erwartete Belohnung im Terminalzustand

Eigenschaften von MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT

Aheuristic:

- es wird nur der Terminalzustand ausgewertet, dadurch werden keine Bewertungsfunktionen benötigt
- es wird kein Expertenwissen oder Schätzverfahren gebraucht
- die ersten erfolgreichen Go-Programme wurden von Leuten entwickelt, die selbst nur mittelmäßige Go Spieler waren
- Heuristiken können die Spielstärke aber enorm verbessern . . .
  aber kosten Zeit und verringern die Anzahl der möglichen Durchläufe
- Abwegen dieses Konflikts

Eigenschaften von MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT

Anytime:

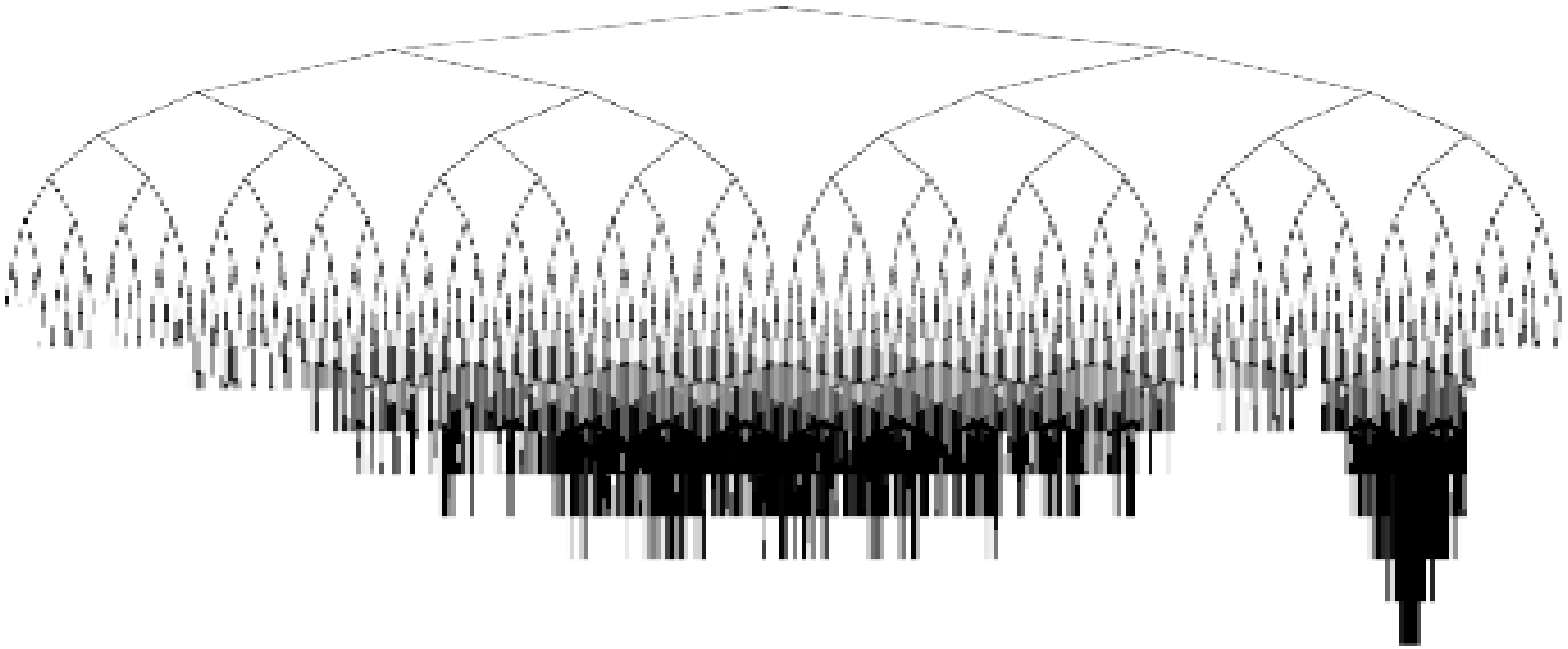
- Eins der beiden Ziele von Koscic & Szepesvári
- nach jedem Durchlauf werden die Knoten aktualisiert
- Auswertung des Baums jederzeit möglich

Eigenschaften von MCTS

TECHNISCHE UNIVERSITÄT DARMSTADT

Asymmetrie:

- die Tree Policy erlaubt das Bevorzugen von starken Zügen
- es entsteht ein asymmetischer Baum
- schlechte Züge werden jedoch nie komplett abgeschnitten



Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

MCTS:

- entstand durch die Kombination von Monte Carlo Methoden und Suchbäume
- ist eine Familie von Algorithmen (Grundgerüst)
- baut schrittweise einen Baum auf
- führte zu ersten Erfolgen von Go-Programmen

Quellen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton:
  A Survey of Monte Carlo Tree Search Methods.
- G. M. J.-B. Chaslot, S. Bakkes, I. Szita, and P. Spronck: Monte-Carlo Tree Search: A New Framework for Game AI.
- G. M. J.-B. Chaslot, S. de Jong, J.-T. Saito, and J. W. H. M. Uiterwijk: Monte-Carlo Tree Search in Production Management Problems
- R. Coulom: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search
- P. Drake, S. Uurtamo: Move Ordering vs Heavy Playouts: Where Should Heuristics be Applied in Monte Carlo Go.
- S. Gelly, Y. Wang, R. Munos, and O. Teytaud: Modification of UCT with Patterns in Monte-Carlo Go.

Quellen

TECHNISCHE UNIVERSITÄT DARMSTADT

- G. Chaslot, J.-T. Saito, J.W.H.M. Uiterwijk, B. Bouzy, H.J. von den Herik: Monte Carlo Strategies for Computer Go.
- L. Kocsis, C. Szepesvári: Bandit based Monte-Carlo Planning.
- L. Kocsis, C. Szepesvári, and J. Willemson: Improved Monte-Carlo Search.
- P.-A. Coquelin, R. Munos: Bandit Algorithms for Tree Search.