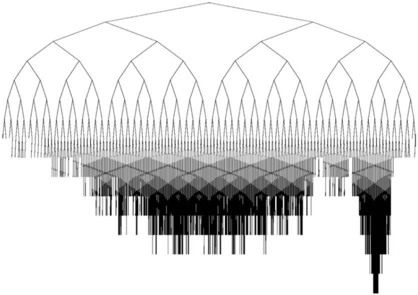
Monte Carlo Tree Search

Tackleing high branching factors

Robert Klassert



Universität Heidelberg
Artificial Intelligence for Games, summer term 2019
under supervision of Prof. Ullrich Köthe

What are we going to learn?

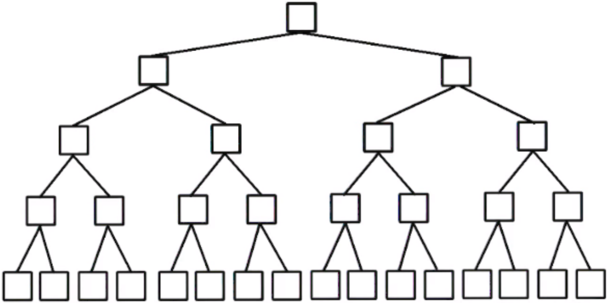
1. Introduction
   - Recap: Game tree search
   - Motivation: Why use MCTS?
2. Step back: Decision Theory
   - Markov Decision Processes
   - Application to combinatorial games
3. Vanilla MCTS
   - Dynamic tree building
   - Rollout and backpropagation
   - Policies
4. Exploration & Exploitation
   - Multi-armed bandits
   - Upper confidence bound action selection
   - The UCT algorithm
   - Enhancements
5. How well does it do? - Examples
6. Conclusion

Recap: Game tree search

- Game tree: depth d, branching factor b
- minimax: b^d (leave nodes grow exponentially)
- $\alpha - \beta$ speed up: $b^{d/2}$
- Domain dependent heuristics: e.g. quiescence search, null-move pruning

$\rightarrow$ success depends on evaluation function

$\rightarrow$ very limited search depth for large b (Chess $\bar{b} \approx 35$, Go $\bar{b} \approx 250$)



Motivation: Why use MCTS?

- No evaluation function needed! (domain independent)
- Grows a tree instead of pruning it $\rightarrow$ high branching factors not so bad
- Built-in "iterative deepening" and asymmetric tree growth
- Used by *Alpha Zero*, *Leela Chess Zero* and even *Total War: Rome II*

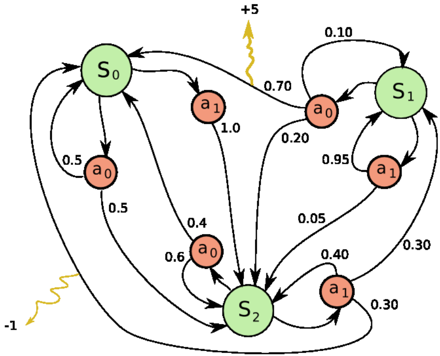
Takeaways

- Generality of MCTS
- Be able to implement MCTS by your own

Model for a sequential decision problem:

- Set of states S
- Set of actions A
- transition model $T(s, a, s')$
  $\rightarrow$ Markov property
- reward $R(s, a, s')$

Objective:
Find function $\pi : S \rightarrow A$ that maximizes expected reward $\bar{R}$



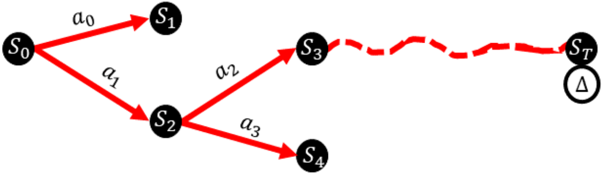
towardsdatascience.com

Application to combinatorial games

Combinatorial games

Two player, zero-sum, perfect information, deterministic, sequential

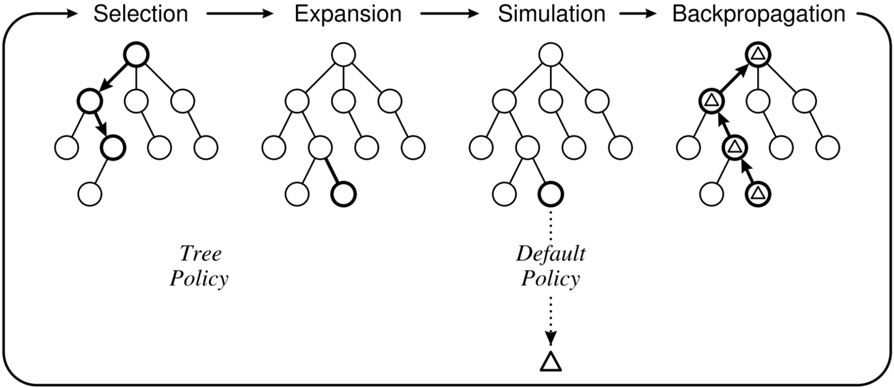
- $S \triangleq$ all possible board positions
- $A \triangleq$ all possible moves
- transition model $T(s,a,s') = \begin{cases} 0 \text{ if } a \text{ is illegal} \\ 1 \text{ if } a \text{ is legal} \end{cases}$
- reward $R(s,a,s') = \begin{cases} 0 \text{ if } s' \text{ is non} - \text{terminal} \\ \Delta \text{ if } s' \text{ is terminal} \end{cases}$



Step 1 Start MCTS in root (current) node

Step 2 Run until a computational/time constraint is met

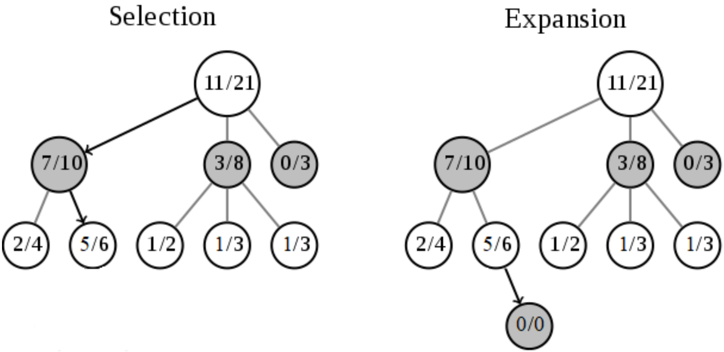
Step 3 Take action to the "best" child node → Step 1



Browne et al. [2]

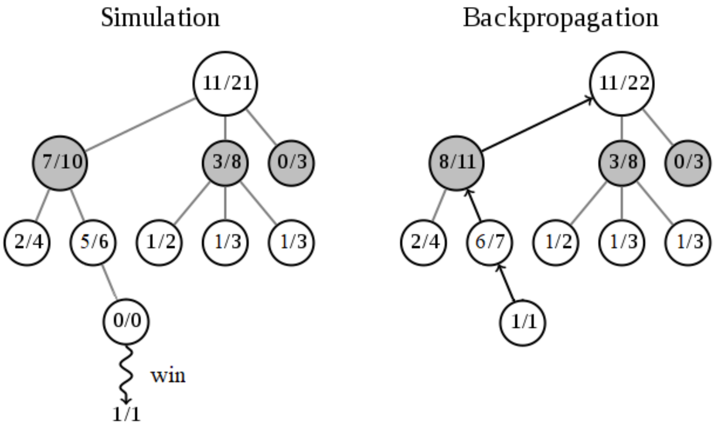
Dynamic tree building

- Selection of child nodes based on node statistics (here: wins/visits $\approx Q$-value)
- Expansion of the tree when reaching a leave node



Rollout and backpropagation

- Simulation until reaching sufficient depth (here: terminal state)
- Update of node statistics based on the simulation outcome



Policies

Rollout/default policy

- responsible for simulation $\rightarrow$ value estimation
- usually: select random action at every node (*flat* Monte-Carlo)
- *AlphaGo*: rollout replaced by NN value estimation

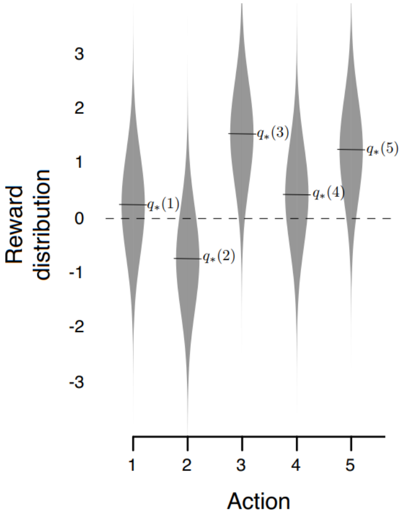
Tree policy

- responsible for selection and expansion $\rightarrow$ building a useful tree
- for example: $\hat{a}(s) = \mathrm{argmax}_a(Q(s'))$ $\rightarrow$ this is too greedy!
- Ideally: *always* select action close to, and ultimately converge to the optimal action $\rightarrow$ UCT algorithm (Kocsis and Szepesvari [5])

- Slot-machine with k arms and random payoffs X_j
- Task: maximize payoff over a period of time
- But: the reward distributions are unknown

  → Trade-off needed between exploration of the reward distribution and its exploitation

  → Policy must consider mean μ_j and variance σ_j of the sample distribution



Sutton et al. [6]

Upper confidence bound (UCB) action selection

- Basic idea: Select arm that maximizes an UCB, e.g. $\hat{a} = \mathrm{argmax}_a(\mu_j + \sigma_j)$
- Problem: some trade-off, but no game-theoretic guarantees
- Solution: use an UCB that minimizes the *regret* $R_N = \mu^* N - \sum_j \mu_j \mathbf{E}[N_j]$

$$UCB1 = \mu_j + \sqrt{2 \ln N / N_j} \text{ (Auer et al. [1])}$$

$\rightarrow$ Growth of R_N within a factor of an optimal policy

$\rightarrow$ Also applicable to non-stationary distributions! [5]

The UCT algorithm (Upper Confidence bounds for Trees)

Back to MCTS: Model node selection as multi-armed bandit problem

- Arms correspond to actions
- Payoffs correspond to expected reward Q_j/N_j

UCT policy

$\hat{a} = \mathrm{argmax}_a(Q_j/N_j + C \cdot \sqrt{2 \ln N/N_j})$

- large first term: exploitation
- large second term: exploration, C domain-dependent parameter
- For $N \to \infty$, game-theoretic minimax tree is built!

Enhancements

Tree-policy:

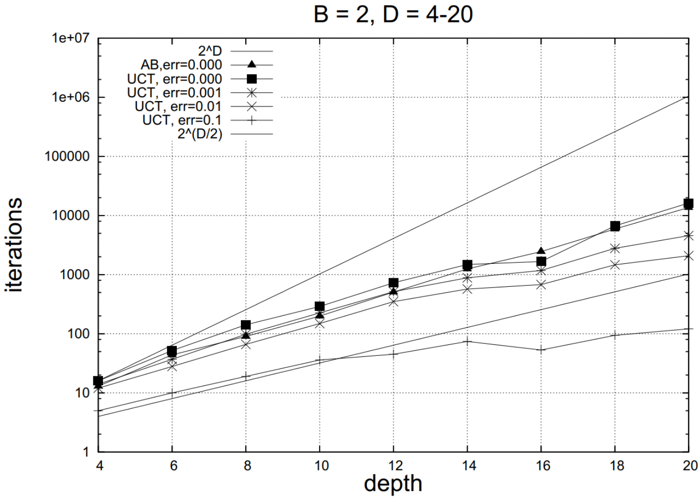
- First Play Urgency → encourage early exploitation
- Progressive bias → blend in heuristic value for low visit count
- Search seeding → keep value estimates of previous MCTS runs

Rollout policy: e.g. use (learned) evaluation function
Update step: e.g. All Moves As First (AMAF)
General enhancements:

- Parallelization → run MCTS multiple times
- Pruning

Overview: Browne et al., *A Survey of Monte Carlo Tree Search Methods*, 2012 [2]

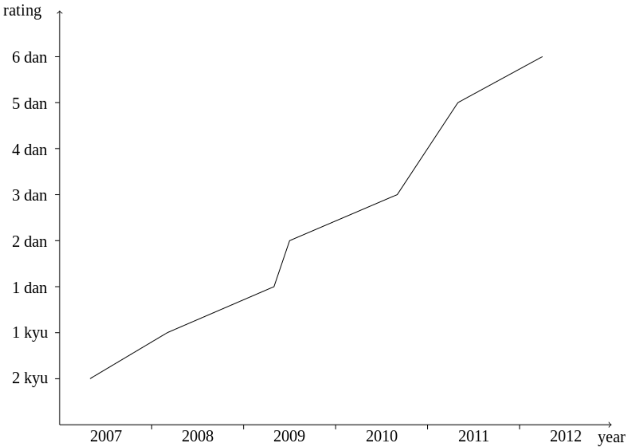
Example: random game trees (P-games)



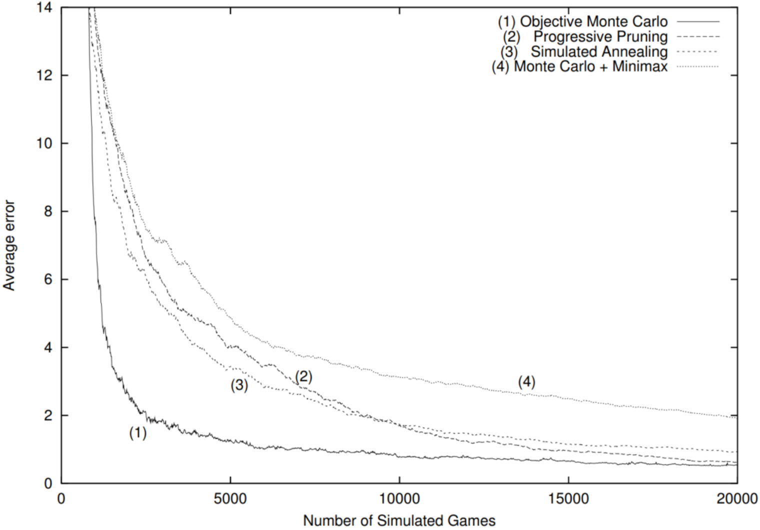
Koscsis and Sepesvari [5]

Example: Go

Improvements in computer Go since onset of MCTS



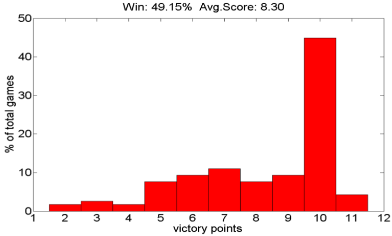
Example: Go



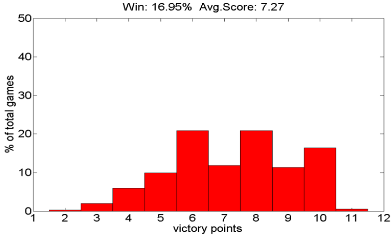
Chaslot et al. 2006 [4]

Example: Settlers of Catan

- JSettlers is a handcrafted agent
- 1 MCTS player (with some domain knowledge) vs 3 JSettlers



MCTS-10000 player



JSettlers player

Szita et al. [7]

Conclusion

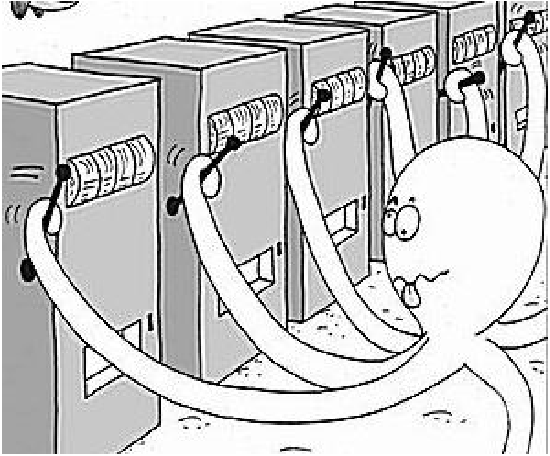
MCTS is ...

- ... a way to iteratively approximate decision trees
- ... employing random simulations to deal with delayed rewards
- ... aheuristic while converging to an optimal strategy
- ... more similar to how humans play games
- ... an important part of modern RL systems

However, ...

- ... still needs domain knowledge to produce human-like performance
- ... runs/games are independent $\rightarrow$ there is no learning, intelligence?

Thank you for your attention!
Questions? Ideas? Comments?



https://s4745.pcdn.co/wp-content/uploads/2015/10/casino.jpg

References I

P. Auer, N. Cesa-Bianchi, and P. Fischer.
Finite-time analysis of the multiarmed bandit problem.
Machine Learning, 47(2):235–256, May 2002.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton.
A Survey of Monte Carlo Tree Search Methods.
IEEE Transactions on Computational Intelligence and AI in Games, 4(1):1–43, 2012.

G. Chaslot, S. Bakkes, I. Szita, and P. Spronck.
Monte-Carlo Tree Search: A New Framework for Game AI.
In *AIIDE*, 2008.

G. Chaslot, J. takeshi Saito, J. W. H. M. Uiterwijk, B. Bouzy, and H. J. Herik.
Monte-Carlo Strategies for Computer Go.

References II

L. Kocsis and C. Szepesvári.
Bandit based Monte-Carlo Planning.
In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

R. S. Sutton, A. Barto, and A. G. Barto.
Reinforcement Learning.
Adaptive computation and machine learning. MIT Press, Cambridge, Mass. [u.a.], 3. printing edition, 2000.

I. Szita, G. Chaslot, and P. Spronck.
Monte-Carlo Tree Search in Settlers of Catan.
volume 6048, pages 21–32, 05 2009.