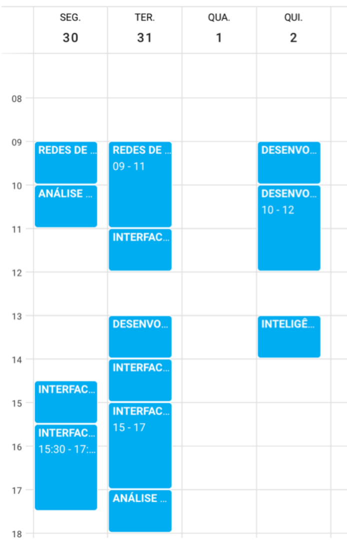
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

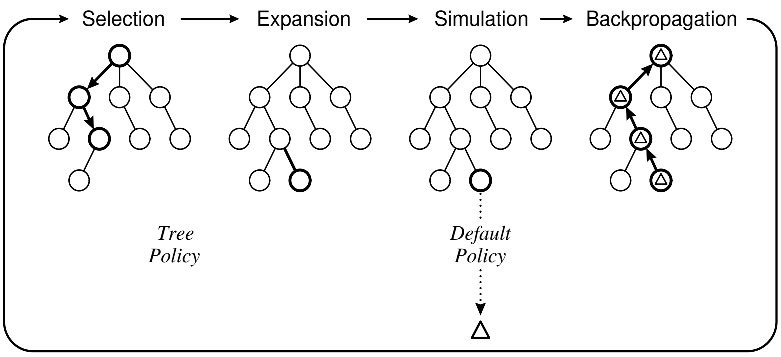
Monte Carlo Tree Search (cont.)

Classes for next week

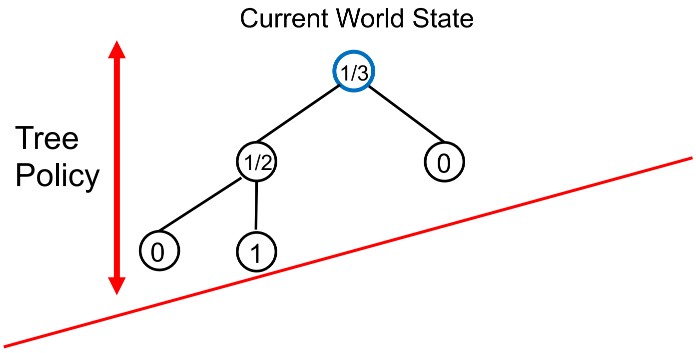


30-Oct-23 http://tiny.cc/ia2023-24 10 – 2

Monte Carlo Tree Search (Kocsis & Szepesvari, 2006)



UCT Algorithm (Kocsis & Szepesvari, 2006)



What is an appropriate tree policy?

When all node actions tried once, select action according to tree policy

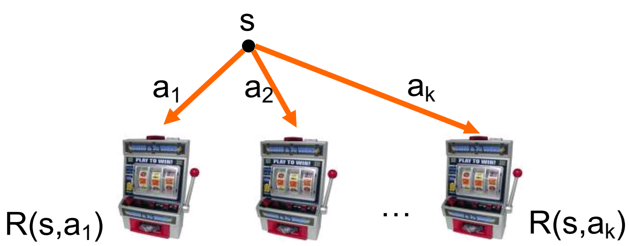
Played them again…

-1€, 20€ 2€, 3€ 10€, 2€

- Problem: which machine should you play next?

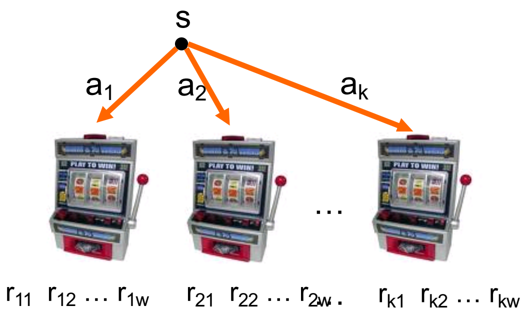
Multi-armed Bandit Problem, informally

- Find an arm-pulling strategy such that the expected total reward at time *n* is close to the best possible (*i.e.*, always pulling the best arm)
  - Must balance **exploring** machines to find good payoffs and **exploiting** current knowledge



Uniform (Naïve) bandit algorithm (Even-Dar et. al., 2002)

1. Pull each arm *w* times (uniform pulling).
2. Return arm with best average reward.



Uniform is a poor choice – waste of time on bad arms

Exploration vs. exploitation

- When to explore (try a new machine)?
- When to stop exploring and start exploiting (play the apparently best machine)?

UCT Algorithm (Kocsis & Szepesvari, 2006)

- Tree policy is guided by:

Exploration Parameter

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

  - Q(s,a): average reward received in current trajectories after taking action a in state s, Q(s,a)=wins(s,a)/n(s,a)
  - n(s): number of times s encountered in the sims
  - n(s,a): number of times action a taken in s

30-Oct-23 http://tiny.cc/ia2022-23 21-9

UCT Algorithm (Kocsis & Szepesvari, 2006)

- Q(s,a)=wins(s,a)/n(s,a)
- n(s): number of times s encountered in the sims
- n(s,a): number of times action a taken in s

Exploration Parameter

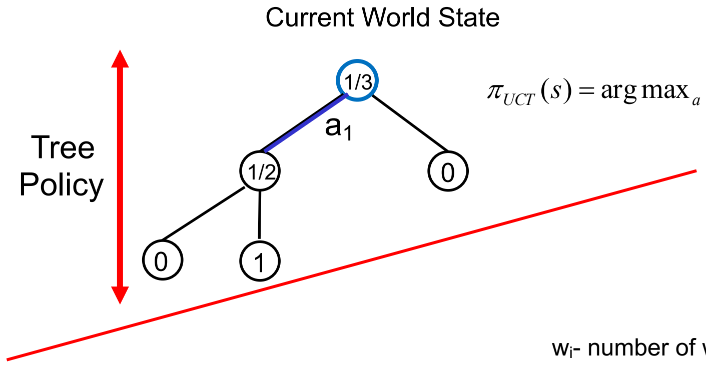
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Value Term:
favors actions that looked good historically

Exploration Term:
actions get an exploration bonus that grows with ln *n(s)*

Doesn't waste much time on sub-optimal arms unlike uniform!

UCT Algorithm (Kocsis & Szepesvari, 2006)



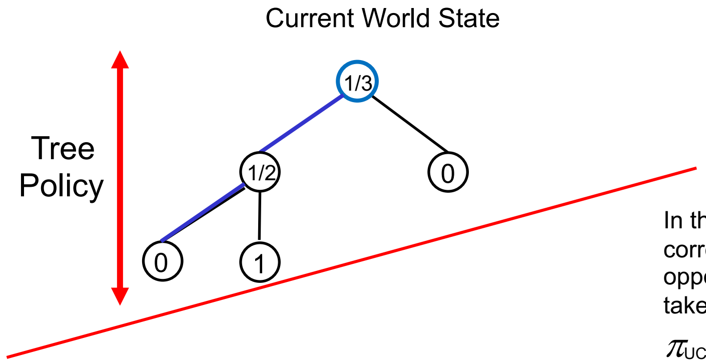
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

w_i - number of wins in node *i*
n_i – number of sims in i
t – number of sims in the father of i

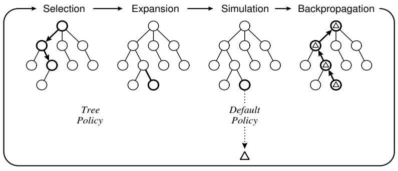
When all node actions tried once, select action according to tree policy

UCT Algorithm (Kocsis & Szepesvari, 2006)



In this level, corresponding to opponent turn, take the min of $\pi_{UCT}(s)$

UCT Recap



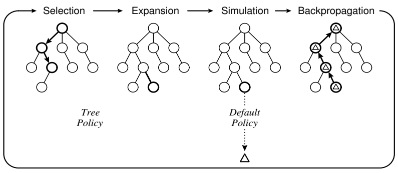
- To select an action at a state *s*
  - Build a tree using *N* iterations of monte-carlo tree search
    - Default policy is uniform random
    - Tree policy is based on UCT rule, i.e., select action *a* that maximizes

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

- More simulations more accuracy

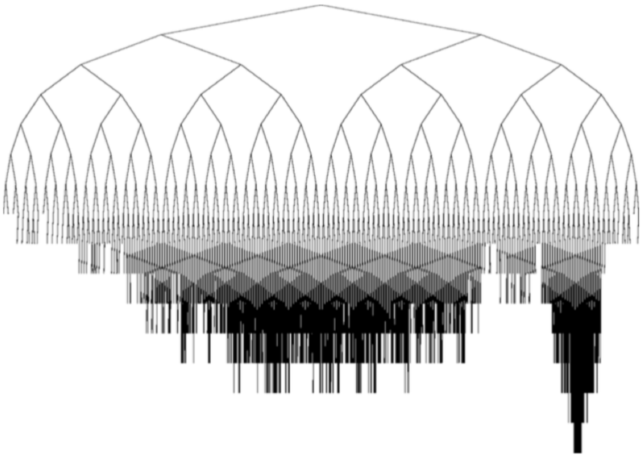
UCT Recap



Algorithm 1 General MCTS approach.

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

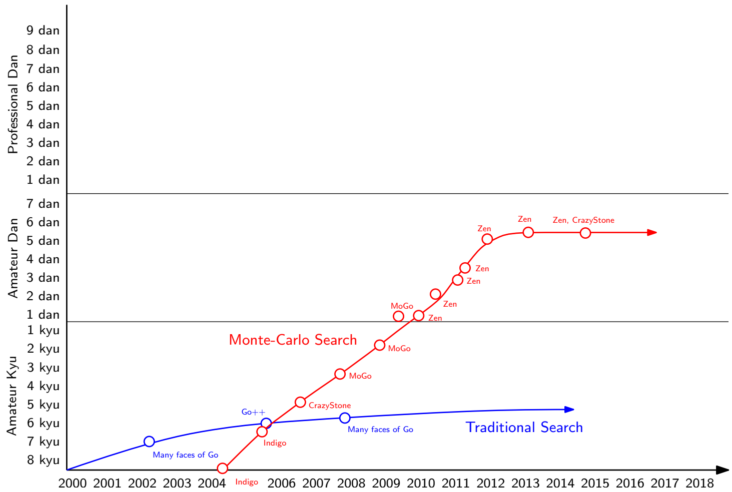
Result: Partial exploration of the game tree



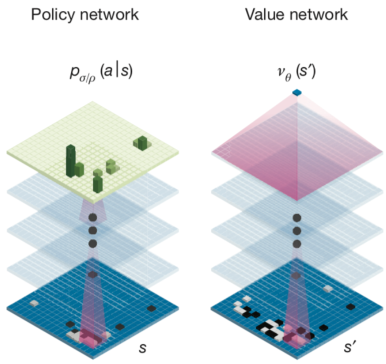
Some Improvements

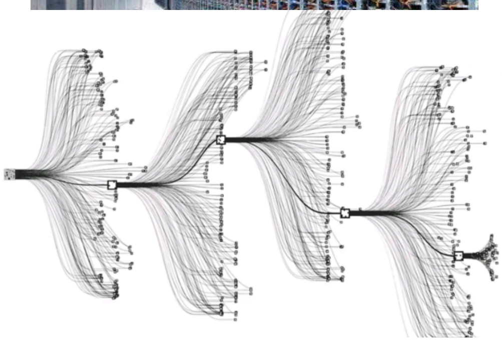
- Use domain knowledge to figure out a better than random (rollout) policy
  - e.g., don't choose obviously bad actions
- Use domain knowledge to predict the opponent actions
- Learn a heuristic function to evaluate positions and use it to evaluate and select leaf nodes for further simulations

Evo of computer GO

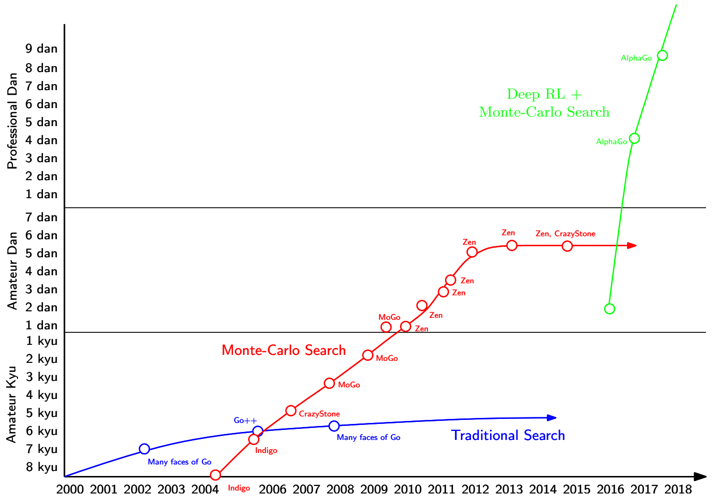


Meet AlphaGo





Evo of computer GO



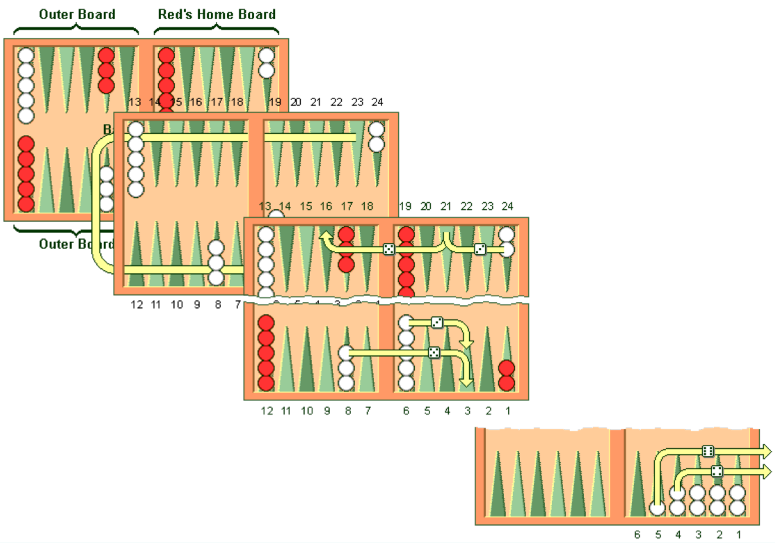
MCTS/UCT take home points

- State of the Art any-time look-ahead probabilistic search for sequential decisions in the presence of uncertainty.

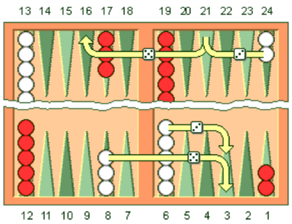
Previously on AI: A classification of games

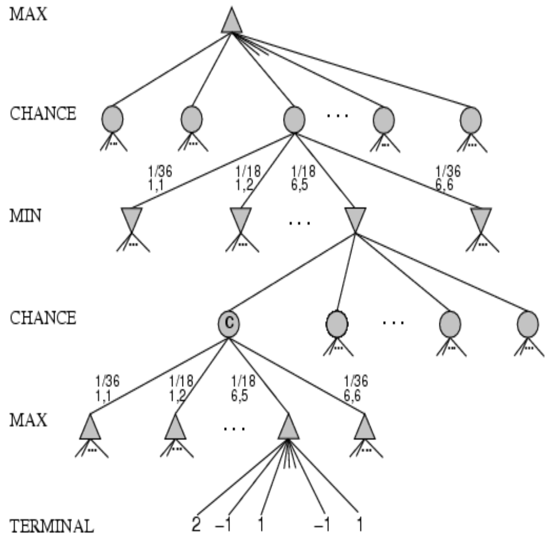
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Backgammon: A game that includes chance



Chance nodes



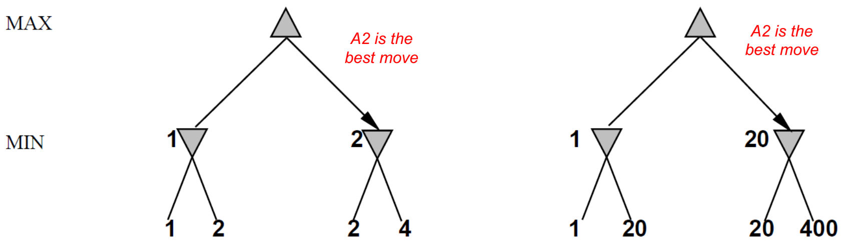


Expected minimax value

EXPECTED-MINIMAX-VALUE(n) =

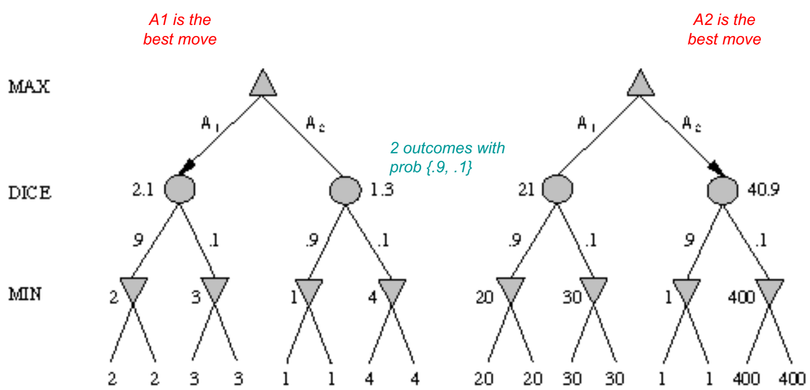
$$\begin{cases} \text{UTILITY}(n) & \text{If } n \text{ is a terminal} \\ \max_{s \in successors(n)} \text{MINIMAX-VALUE}(s) & \text{If } n \text{ is a max node} \\ \min_{s \in successors(n)} \text{MINIMAX-VALUE}(s) & \text{If } n \text{ is a min node} \\ \sum_{s \in successors(n)} P(s) * \text{EXPECTEDMINIMAX}(s) & \text{If } n \text{ is a chance node} \end{cases}$$

Minimax: exact values of Eval are irrelevant



- Behavior is preserved under any monotonic transformation of Eval function.

Expected minimax value, example of



Backgammon in practice

- Dice rolls increase the branching factor: 21 possible outcomes with 2 dices
- α-β pruning becomes much less effective.
- TDGammon: uses depth-2 search + very good Eval (neural-network with reinforcement learning);

  ranked among the top three players in the world

Classes for next week