Lecture 8: MCTS and AlphaGo

Viliam Lisý & Branislav Bošanský

Artificial Intelligence Center
Department of Computer Science, Faculty of Electrical Eng.
Czech Technical University in Prague

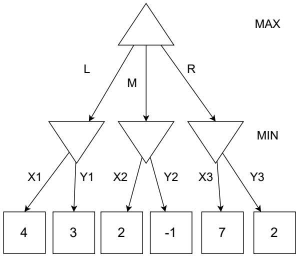
viliam.lisy@fel.cvut.cz

April, 2021

Plan of today's lecture

1. Monte Carlo Tree Search
2. Overview of basic improvements
3. The challenge of computer Go
4. AlphaGo

Recap.: Solving the Two-Player Games



Similarly to deterministic uninformed search, we can use a depth-first search algorithm. For a history h:

1. if h is a terminal history ($h \in Z$), then return $u(z)$,
2. if h is a decision node, evaluate all children $v_a = \text{search}(A(h))$ and
    1. if $h \in H_1$, return $\max_{a \in A(h)} v_a$
    2. if $h \in H_2$, return $\min_{a \in A(h)} v_a$

This baseline algorithm is known as **minimax** algorithm or simply a **backward induction** in two-player perfect information games.

The utility of player 1 when both players play optimally is called **the value of the game**.

Games are BIG

The number of reachable states:

- Chess: $\approx 10^{45}$ ~~10^{45}~~ 10^{23}
- Go: $\approx 10^{170}$ ~~10^{170}~~ 10^{85}

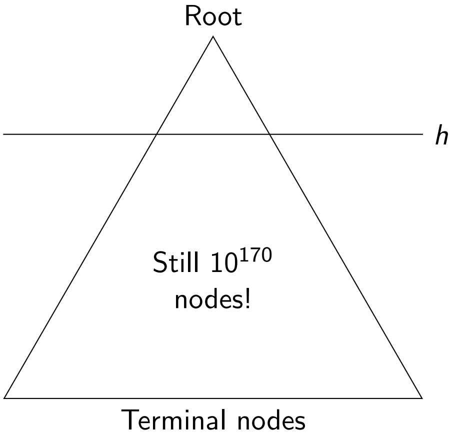
Can't we just prune most of the states out?

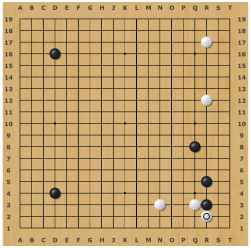
Theorem

For a game with branching factor b and depth d, $\alpha\beta$-search will evaluate at least $b^{d/2} = \sqrt{b^d}$ nodes.

RCI cluster (most powerful Czech AI cluster for 2M EUR in 2018)
1.5TB $= 10^{12}$ of RAM, 226TB of drives $= 2.2 \times 10^{14}$
10^9 RCI clusters necessary to store the strategy with 1B per state

Depth-limited game solving





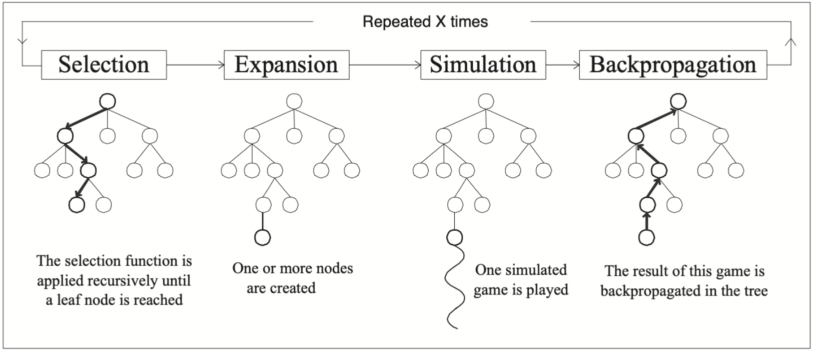
Who will win?

Sometimes very hard to make a good heuristic evaluation.

Monte Carlo Tree Search

Idea:

1. Instead of evaluation function, use random roll-outs (simulations) of the rest of the game
2. Store detailed statistics only in relevant parts of the game tree


Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

(Image from Chaslot et al. 2007)

Monte Carlo Tree Search - Demonstration

$$\begin{tikzcd}
\text{Selection} \arrow[d] \\
\text{Expansion} \arrow[d] \\
\text{Simulation} \arrow[d] \\
\text{Backpropagation} \arrow[uuu, bend left=60]
\end{tikzcd}$$

MCTS Selection

We want to explore the more promising actions more often
We want to learn which actions are the most promising

Does it sound familiar?

Exploration vs. exploitation dilemma
- Any algorithm for the multi-arm bandit problem can be used
- MCTS + UCB = UCT – the most popular MCTS variant

$$A_t(s) \doteq \arg\max_a \left[Q_t(s,a) + c\sqrt{\frac{\log N_t(s)}{N_t(s,a)}} \right]$$

Where s is the node in the tree where we perform the selection.

MCTS Expansion

The part of the search space storing the statistics is expanded

- all actions may be added
- a single state-action may be added
- a node may be expanded only after visited multiple times

Progressive widening

- games may have many actions – Go (19^2), Arrimaa ($\approx 20k$)
- a single state-action may be added at a time
- PW:
  - start with few (heuristically chosen?) actions initially
  - add more once the previously added are explored sufficiently
  - works even in with infinite number of actions
  - keep $k = \lceil C \cdot N(s)^\alpha \rceil$ actions with $\alpha < 0.5$
  - studied in bandit literature on infinitely many armed bandits

MCTS Simulation and Backpropagation

Simulation: choose actions based on fast policies until game ends

- purely random surprisingly effective
- hand-coded knowledge
- learned knowledge

Backpropagation: update statistics used by the selection

- $N(s), N(s,a)$
- $Q(s,a)$
- whatever – rewards range, variance, $Q(a)$, etc.
- each player stores his perspective vs. min / max

MCTS Is useful even in non-game setting

First developed and popularised in games

Everything works as well with single player

PROST, POMCP, etc.

More on it in B(E)4M36PUI – Planning for Artificial Intelligence

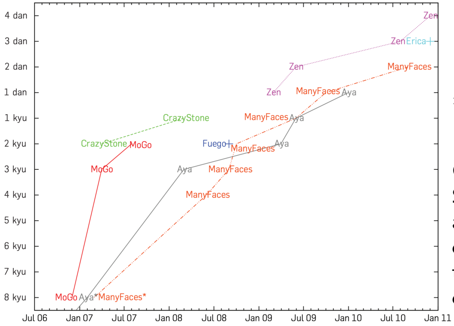
Further reading on MCTS

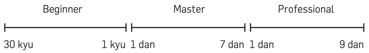
- RL Introduction (Book) – Section 8.11
- Browne, C., Powley, E., Whitehouse, D., et al. 2012. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1), pp.1-43.

The challenge of Go

Following DeepBlue's victory in 1997, Go was the next challenge

- branching $\approx 35 \rightarrow\approx 350$
- game length ≈ 57 moves $\rightarrow\approx 300$ moves
- popular: 4000+ years old and $\approx 27M$ players worldwide

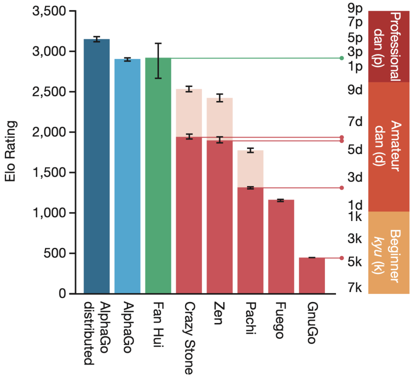




* pre-MCTS approaches

Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C. and Teytaud, O., **2012**. The grand challenge of computer Go: Monte Carlo tree search and extensions. Communications of the ACM, 55(3), pp.106-113.

AlphaGo



Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), pp.484-489.

AlphaGo

Idea:

- Use MCTS as the base algorithm
- Capture the existing human knowledge in a policy $p_\sigma(a|s)$
- Learn a fast simulation policy $p_\pi(a|s)$ for rollouts
- Use RL techniques to optimize policy $p_\rho(a|s)$ in self-play
- Use RL to learn a value function $v(s)$
- Guide MCTS by policies and combine simulations with v

AlphaGo – learning policies

Supervised learning of human policy

- Data: (s_i, a_i) for 30 million positions from KGS Go Server
- Stochastic Gradient Ascent maximizing $\mathbb{E}_i \log p_\sigma(a_i|s_i)$
- Final prediction accuracy was 57%
- 1000x faster roll-out policy p_π trained the same achieved 24% accuracy

Improving policy in self-play

- Initialise p_ρ by p_σ
- Play one match $s_1, \dots, s_T$ and receive outcome $z \in \{-1, 1\}$
- Use SGA to maximize $\mathbb{E}_{t<T} \log p_\sigma(a_t|s_t)z$
- Eventually p_ρ wins over p_σ in 80% of games

AlphaGo – learning value function

The goal is to estimate state value under policy p_ρ:

$$v^{p_\rho}(s) = \mathbb{E}[z|s_t = s, a_{t\dots T} \sim p_\rho]$$

- Data: (s_i, z_i) for 30 million self-play games (only per game)
- Use Stochastic Gradient Descent to minimize $\mathbb{E}_i(v(s_i) - z)^2$
- Resulting v consistently more accurate than p_π rollouts

AlphaGo – search

Selection:

$$a_t = \arg\max_a \left(Q(s_t, a) + c \frac{\boxed{p_\sigma}(a|s_t)\sqrt{N(s_t)}}{1 + N(s_t, a)} \right)$$

Expansion:

“leaf node may be expanded” hence, likely not always

Simulation:

The result of the value function and simulation $z \sim p_\pi$ is combined

$$V(s_L) = (1 - \lambda)v(s_L) + \lambda z$$

Backpropagation:

For all visited (s_t, a_t)

$$N(s_t, a_t) \mathrel{+}= 1$$

$$Q(s_t, a_t) \mathrel{+}= \frac{1}{N(s_t, a_t)} V(s_L)$$

AlphaGo – results

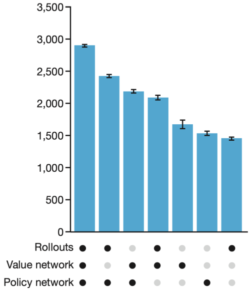
AlphaGo won 494/496 matches against the existing programs

AlphaGo won 5 – 0 against professional European champion

Larger distributed version on 1202 CPUs and 176 GPUs

Observations:

- All components are important →
- AlphaGo evaluated thousands times less positions than DeepBlue
- 10 years earlier than expected
- Human policies still helped in 2015



Further advancements

AlphaGo Zero (2017)

- No human knowledge

AlphaZero (2018)

- No simulation
- Chess: 9 hours, shogi: 12 hours, Go: 13 days

MuZero (2020)

- Not even game rules are necessary

Imperfect information games?

Summary

Common games are large

If you can create a good evaluation function, use $\alpha\beta$ variants

If it is hard to provide evaluation function, use MCTS

If you do not mind a lot of training, combine MCTS with learned policy and value functions

Playing perfect information games is mostly a solved problem