UCB, Monte Carlo Tree Search, AlphaZero

Milan Straka

December 07, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Upper Confidence Bound

Revisiting multi-armed bandits with ε-greedy exploration, we note that using same epsilon for all actions in ε-greedy method seems inefficient.

One possible improvement is to select action according to upper confidence bound (instead of choosing a random action with probability ε):

$$A_{t+1} \stackrel{\text{def}}{=} \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

where:

- t is the number of times any action has been taken;
- $N_t(a)$ is the number of times the action a has been taken;
- if $N_t(a) = 0$, the right expression is frequently assumed to have a value of ∞.

The updates are then performed as before (e.g., using averaging, or fixed learning rate α).

Motivation Behind Upper Confidence Bound

Actions with little average reward are probably selected too often.

Instead of simple ε-greedy approach, we might try selecting an action as little as possible, but still enough to converge.

Assuming that random variables X_i bounded by $[0,1]$ and $\bar X = \frac{1}{N}\sum_{i=1}^N X_i$, (Chernoff-)Hoeffding's inequality states that

$$P\big(\mathbb{E}[\bar X] - \bar X \geq \delta\big) \leq e^{-2N\delta^2}.$$

Our goal is to choose δ such that for every action,

$$P\big(Q_t(a) \leq q_*(a) - \delta\big) \leq \left(\frac{1}{t}\right)^\alpha.$$

We can fulfil the required inequality if $e^{-2N_t(a)\delta^2} \leq \left(\frac{1}{t}\right)^\alpha$, which yields

$$\delta \geq \alpha/2 \cdot \sqrt{(\ln t)/N_t(a)}.$$

Asymptotical Optimality of UCB

We define *regret* as the difference of maximum of what we could get (i.e., repeatedly using the action with maximum expectation) and what a strategy yields, i.e.,

$$\mathrm{regret}_N \stackrel{\text{def}}{=} N \max_a q_*(a) - \sum_{i=1}^{N} \mathbb{E}[R_i].$$

It can be shown that regret of UCB is asymptotically optimal, see Lai and Robbins (1985), *Asymptotically Efficient Adaptive Allocation Rules*; or the Chapter 8 of the 2018 *Bandit Algorithms Book* available online at https://banditalgs.com/.

Upper Confidence Bound Multi-armed Bandits Results

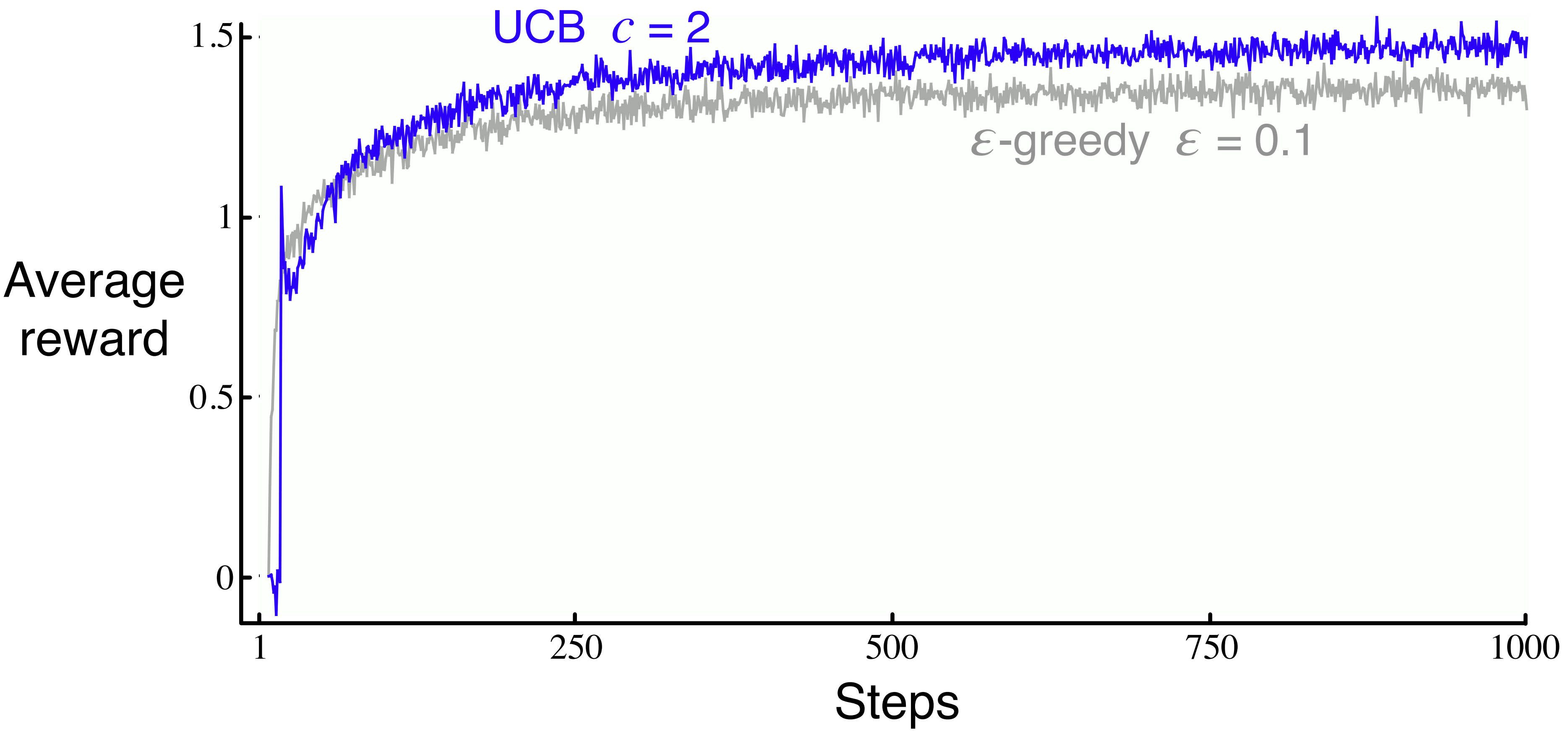


Figure 2.4 of "Reinforcement Learning: An Introduction, Second Edition".

Multi-armed Bandits Comparison

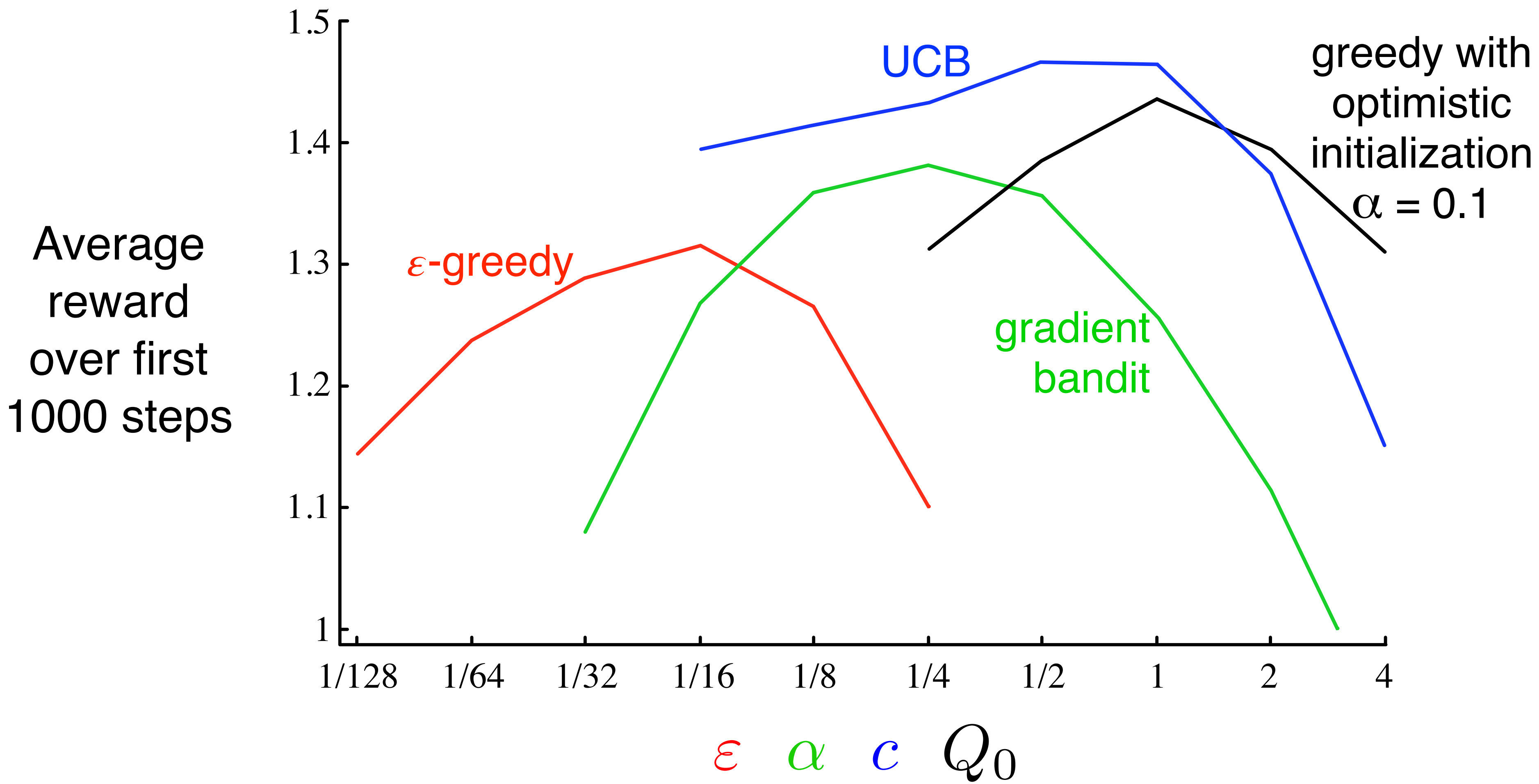


Figure 2.6 of "Reinforcement Learning: An Introduction, Second Edition".

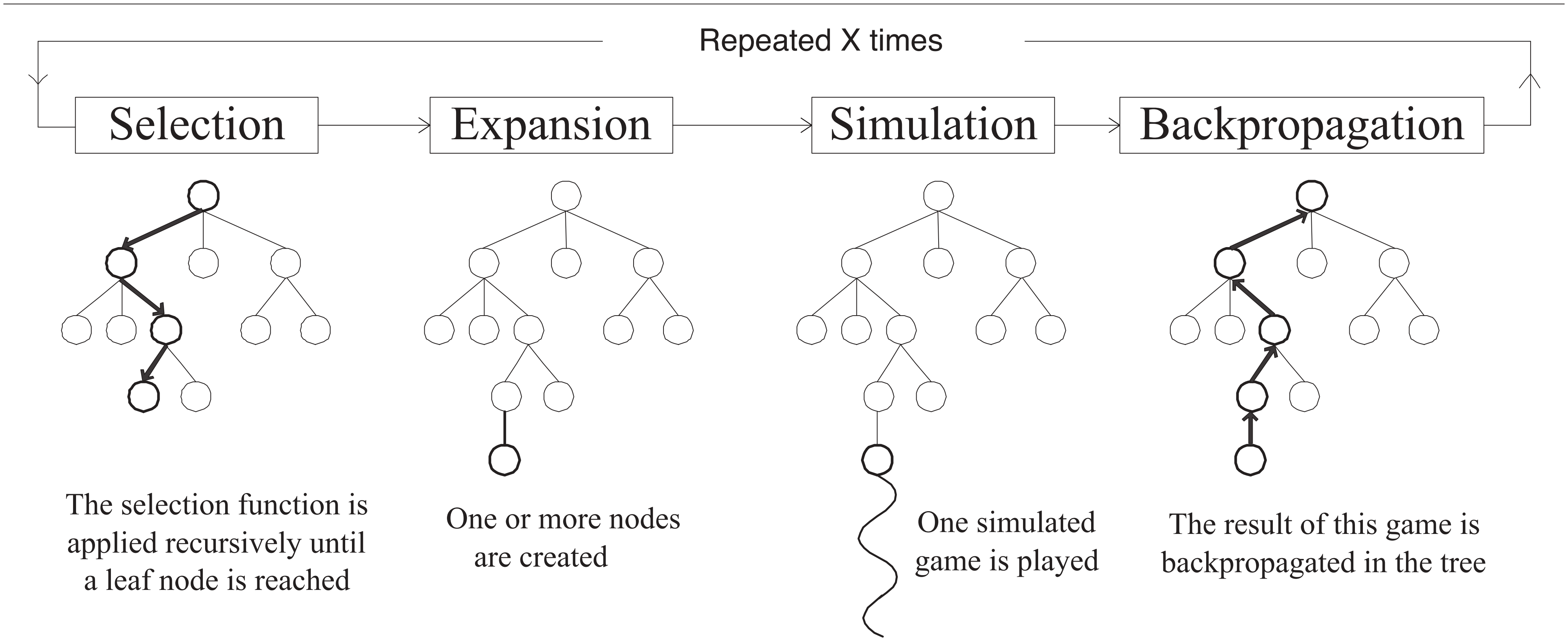


Figure 1 of the paper "Monte-Carlo Tree Search: A New Framework for Game AI" by Guillaume Chaslot et al.

Monte Carlo Tree Search

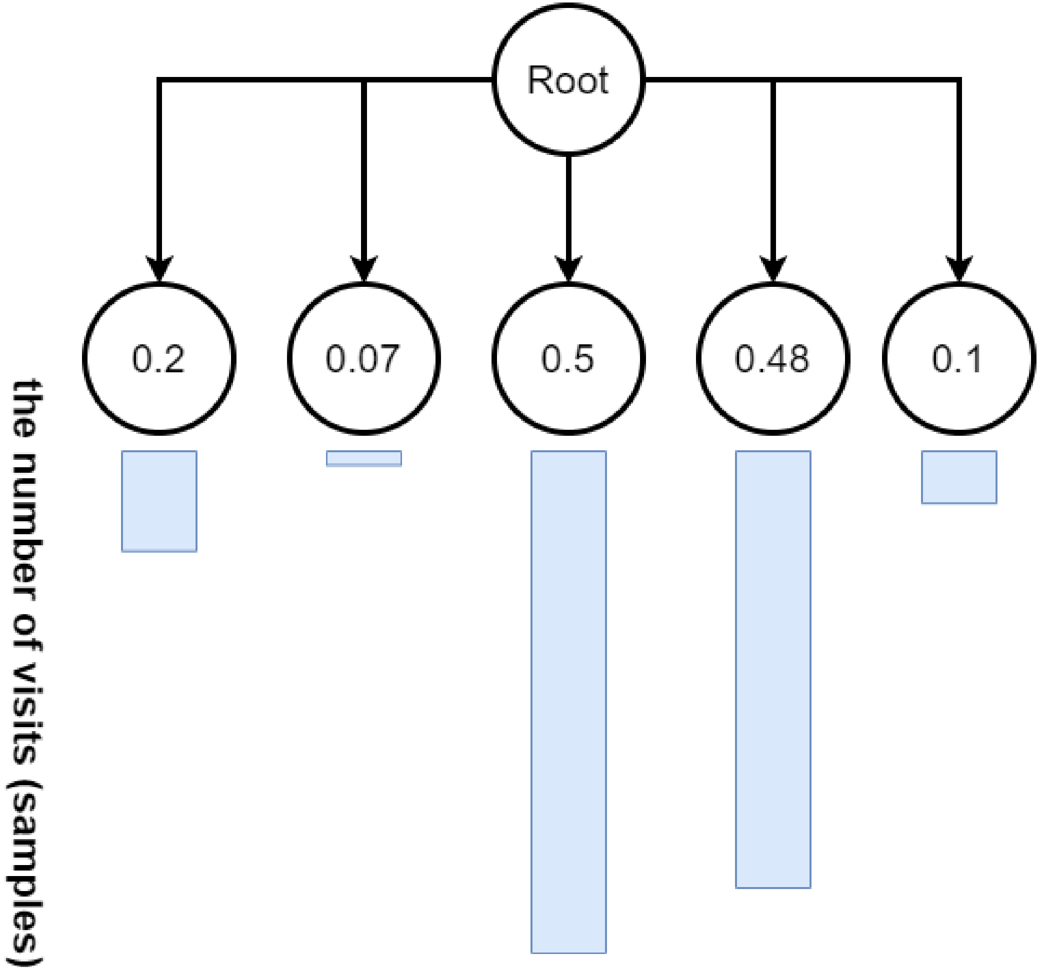


Figure 2 of the paper "Monte Carlo Tree Search: A Review of Recent Modifications and Applications" by M. Świechowski et al.

On 7 December 2018, the AlphaZero paper came out in Science journal. It demonstrates learning chess, shogi and go, *tabula rasa* – without any domain-specific human knowledge or data, only using self-play. The evaluation is performed against strongest programs available.

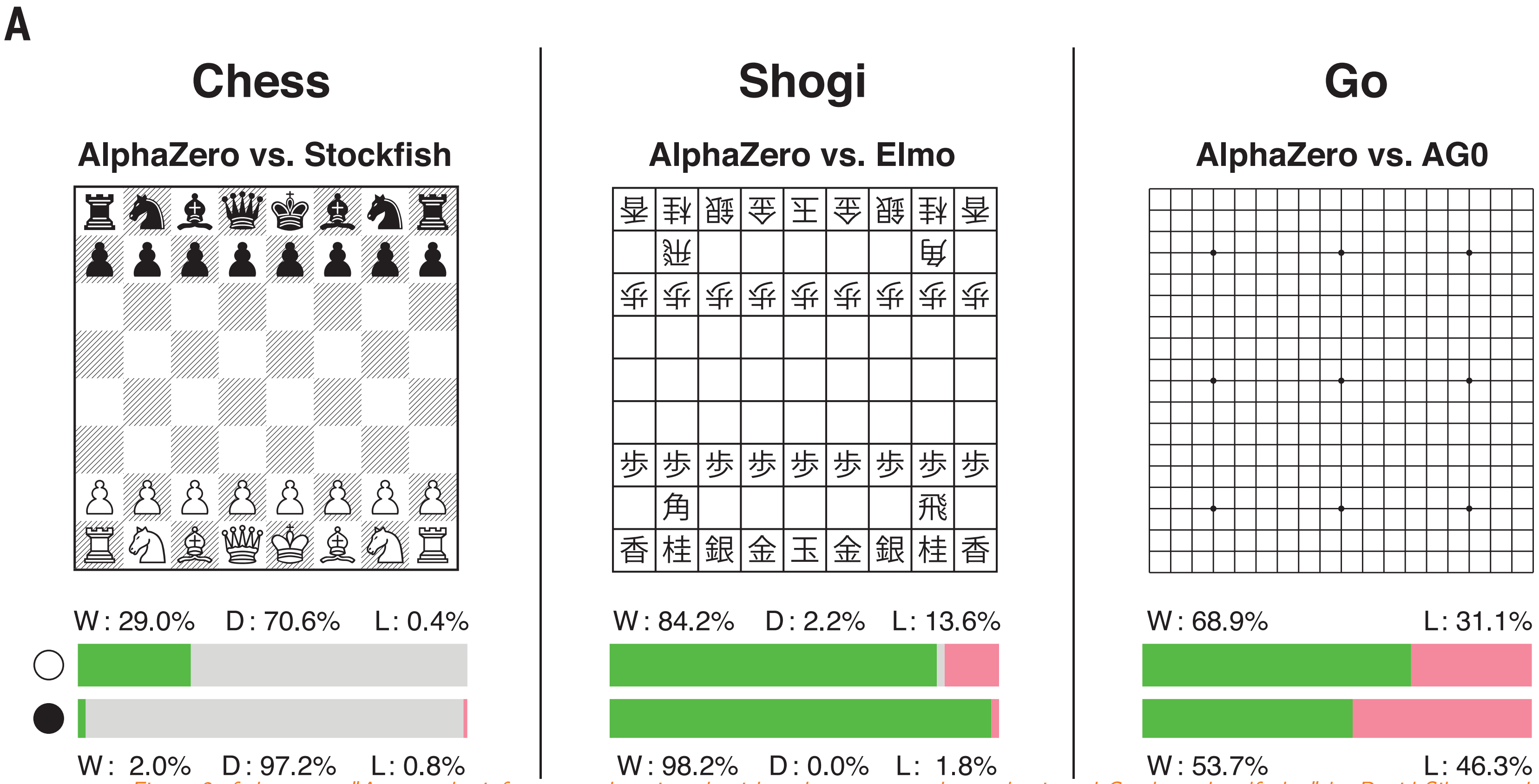


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Overview

AlphaZero uses a neural network predicting $(\boldsymbol{p}(s), v(s)) = f(s; \boldsymbol{\theta})$ for a given state s, where:

- $\boldsymbol{p}(s)$ is a vector of move probabilities, and
- $v(s)$ is expected outcome of the game in range $[-1, 1]$.

Instead of the usual alpha-beta search used by classical game playing programs, AlphaZero uses Monte Carlo Tree Search (MCTS).

By a sequence of simulated self-play games, the search can improve the estimate of $\boldsymbol{p}$ and v, and can be considered a powerful policy evaluation operator – given a network f predicting policy $\boldsymbol{p}$ and value estimate v, MCTS produces a more accurate policy $\boldsymbol{\pi}$ and better value estimate w for a given state s:

$$(\boldsymbol{\pi}(s), w(s)) \leftarrow \mathrm{MCTS}(\boldsymbol{p}(s), v(s), f) \text{ for } (\boldsymbol{p}(s), v(s)) = f(s; \boldsymbol{\theta}).$$

AlphaZero – Overview

The network is trained from self-play games.

A game is played by repeatedly running MCTS from a state s_t and choosing a move $a_t \sim \boldsymbol\pi_t$, until a terminal position s_T is encountered, which is then scored according to game rules as $z \in \{-1, 0, 1\}$.

Finally, the network parameters are trained to minimize the error between the predicted outcome v and the simulated outcome z, and maximize the similarity of the policy vector $\boldsymbol p$ and the search probabilities $\boldsymbol\pi$ (in other words, we want to find a fixed point of the MCTS):

$$\mathcal{L} \stackrel{\text{def}}{=} (z - v)^2 - \boldsymbol\pi^T \log \boldsymbol p + c\|\boldsymbol\theta\|^2.$$

The loss is a combination of:

- a mean squared error for the value functions;
- a crossentropy/KL divergence for the action distribution;
- L2 regularization.

AlphaZero – Monte Carlo Tree Search

MCTS keeps a tree of currently explored states from a fixed root state. Each node corresponds to a game state and to every non-root node we got by performing an action a from the parent state. Each state-action pair (s, a) stores the following set of statistics:

- visit count $N(s, a)$,
- total action-value $W(s, a)$,
- mean action value $Q(s, a) \stackrel{\text{def}}{=} W(s, a)/N(s, a)$, which is usually not stored explicitly,
- prior probability $P(s, a)$ of selecting action a in state s.

AlphaZero – Monte Carlo Tree Search

Each simulation starts in the root node and finishes in a leaf node s_L. In a state s_t, an action is selected using a variant of PUCT algorithm as

$$a_t = \operatorname{arg\,max}_a \big(Q(s_t, a) + U(s_t, a)\big),$$

where

$$U(s, a) \stackrel{\text{def}}{=} C(s) P(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)},$$

with $C(s) = \log\left(\frac{1 + N(s) + c_\text{base}}{c_\text{base}}\right) + c_\text{init}$ being slightly time-increasing exploration rate.

The paper uses $c_\text{init} = 1.25$, $c_\text{base} = 19652$ without any supporting experiments.

Also, the reason for the modification of the UCB formula was never discussed in any AlphaZero paper and is not obvious.

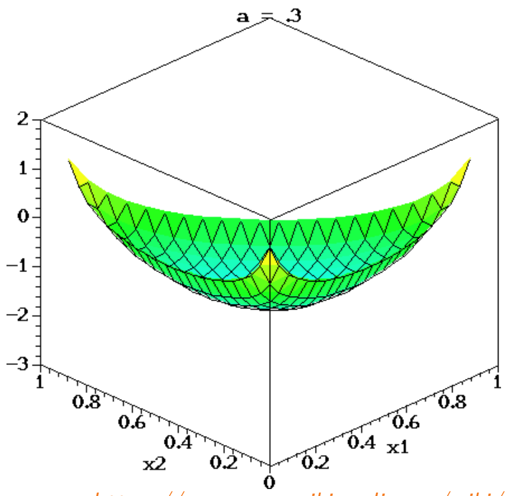
Additionally, exploration in the root state s_root is supported by including a random sample from Dirichlet distribution,

$$P(s_\text{root}, a) = (1-\varepsilon)p_a + \varepsilon\,\mathrm{Dir}(\alpha),$$

with $\varepsilon = 0.25$ and $\alpha = 0.3, 0.15, 0.03$ for chess, shogi and go, respectively.

Note that using $\alpha < 1$ makes the Dirichlet noise non-uniform, with a smaller number of actions with high probability.

The Dirichlet distribution can be seen as a limit of the Pólya's urn scheme, where in each step we sample from a bowl of balls (with initial counts α) and return an additional ball of the same color to the bowl.



https://commons.wikimedia.org/wiki/File:LogDirichletDensity-alpha_0.3_to_alpha_2.0.gif

To sample from a symmetric Dirichlet distribution, we can:

- sample x_i from a Gamma distribution $x_i \sim \mathrm{Gamma}(\alpha)$,
- normalize the sampled values to sum to one, $p_i = \frac{x_i}{\sum_j x_j}$.

When reaching a leaf node s_L, we:

- evaluate it by the network, generating $(\boldsymbol{p}, v)$,
- add all its children with $N = W = 0$ and the prior probability $\boldsymbol{p}$,
- in the backward pass for all $t \leq L$, we update the statistics in nodes by performing
  - $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$, and
  - $W(s_t, a_t) \leftarrow W(s_t, a_t) \pm v$, depending on the player on turn.

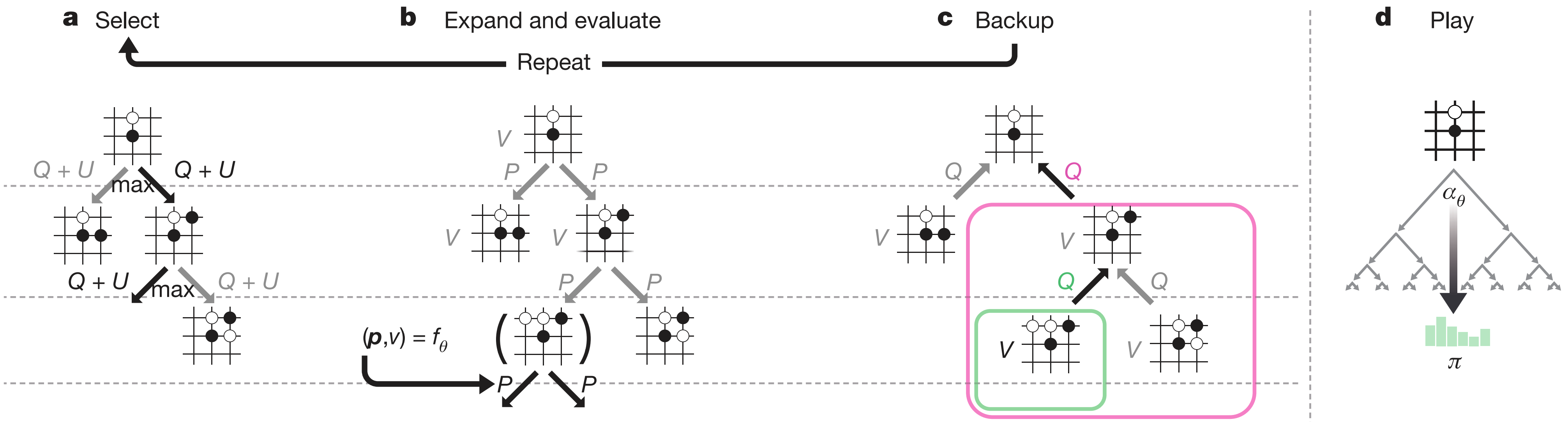


Figure 2 of the paper "Mastering the game of Go without human knowledge" by David Silver et al.

The Monte Carlo Tree Search runs usually several hundreds simulations in a single tree. The result is a distribution proportional to exponentiated visit counts $N(s_{\mathrm{root}}, a)^{\frac{1}{\tau}}$ using a temperature τ ($\tau = 1$ is mostly used), together with the predicted value function.

The next move is chosen as either:

- proportional to visit counts $N(s_{\mathrm{root}}, \cdot)^{\frac{1}{\tau}}$:

$$\boldsymbol{\pi}_{\mathrm{root}}(a) \propto N(s_{\mathrm{root}}, a)^{\frac{1}{\tau}},$$

- deterministically as the most visited action

$$\boldsymbol{\pi}_{\mathrm{root}} = \arg\max_a N(s_{\mathrm{root}}, a).$$

During self-play, the stochastic policy is used for the first 30 moves of the game, while the deterministic is used for the rest of the moves. (This does not affect the internal MCTS search, there we always sample according to PUCT rule.)

Visualization of the 10 most visited states in a MCTS with a given number of simulations. The displayed numbers are predicted value functions from the white's perspective, scaled to $[0, 100]$ range. The border thickness is proportional to a node visit count.

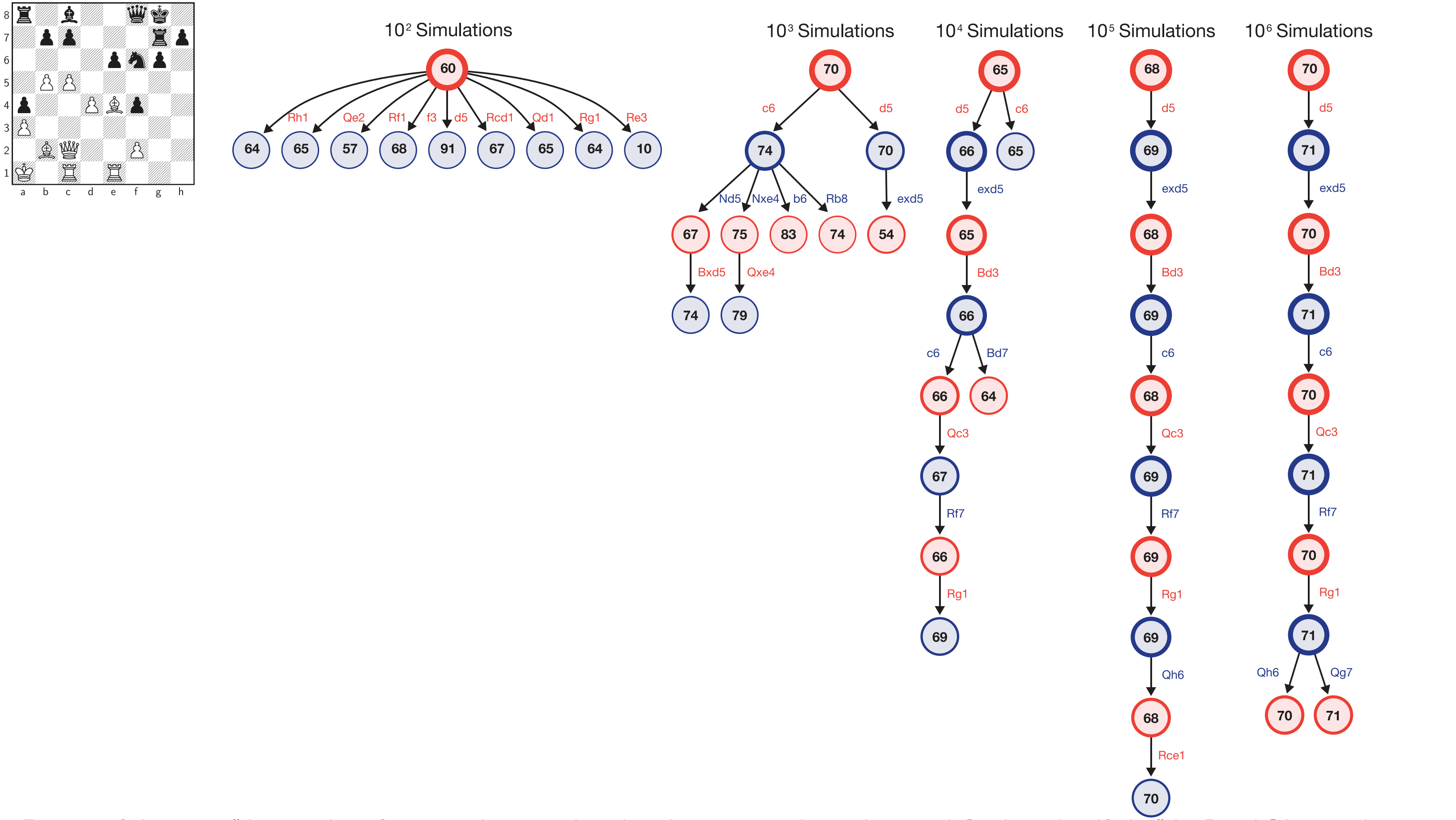


Figure 4 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Network Architecture

The network processes game-specific input, which consists of a history of 8 board positions encoded by several $N \times N$ planes, and some number of constant-valued inputs.

Output is considered to be a categorical distribution of possible moves. For chess and shogi, for each piece we consider all possible moves (56 queen moves, 8 knight moves and 9 underpromotions for chess).

The input is processed by:

- initial convolution block with CNN with 256 3×3 kernels with stride 1, batch normalization and ReLU activation,
- 19 residual blocks, each consisting of two CNN with 256 3×3 kernels with stride 1, batch normalization and ReLU activation, and a residual connection around them,
- *policy head*, which applies another CNN with batch normalization, followed by a convolution with 73/139 filters for chess/shogi, or a linear layer of size 362 for go,
- *value head*, which applies another CNN with one 1×1 kernel with stride 1, followed by a ReLU layer of size 256 and a final $\tanh$ layer of size 1.

AlphaZero – Network Inputs

Go		Chess		Shogi	
Feature	Planes	Feature	Planes	Feature	Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

Table S1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Network Outputs

Chess		Shogi	
Feature	Planes	Feature	Planes
Queen moves	56	Queen moves	64
Knight moves	8	Knight moves	2
Underpromotions	9	Promoting queen moves	64
		Promoting knight moves	2
		Drop	7
Total	73	Total	139

Table S2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Training

Training is performed by running self-play games of the network with itself. Each MCTS uses 800 simulations. A replay buffer of one million most recent games is kept.

During training, 5000 first-generation TPUs are used to generate self-play games. Simultaneously, network is trained using SGD with momentum of 0.9 on batches of size 4096, utilizing 16 second-generation TPUs. Training takes approximately 9 hours for chess, 12 hours for shogi and 13 days for go.

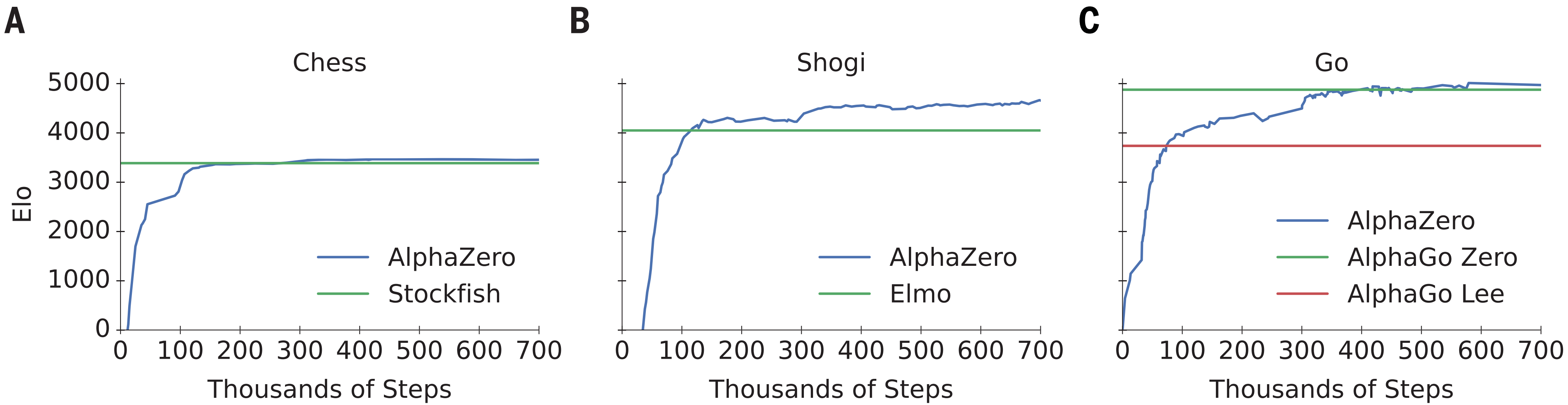


Figure 1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

	Chess	Shogi	Go
Mini-batches	700k	700k	700k
Training Time	9h	12h	13d
Training Games	44 million	24 million	140 million
Thinking Time	800 sims	800 sims	800 sims
	$\sim$ 40 ms	$\sim$ 80 ms	$\sim$ 200 ms

Table S3 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

According to the authors, training is highly repeatable.

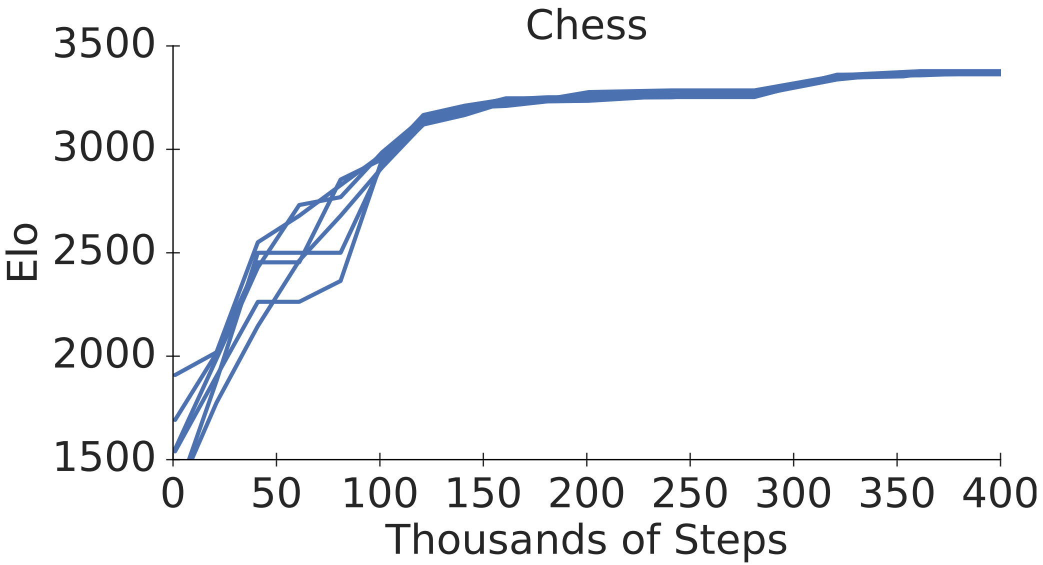


Figure S3 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Symmetries

In the original AlphaGo Zero, symmetries (8 in total, using rotations and reflections) were explicitly utilized, by

- randomly sampling a symmetry during training,
- randomly sampling a symmetry during MCTS evaluation.

However, AlphaZero does not utilize symmetries in any way (because chess and shogi do not have them).

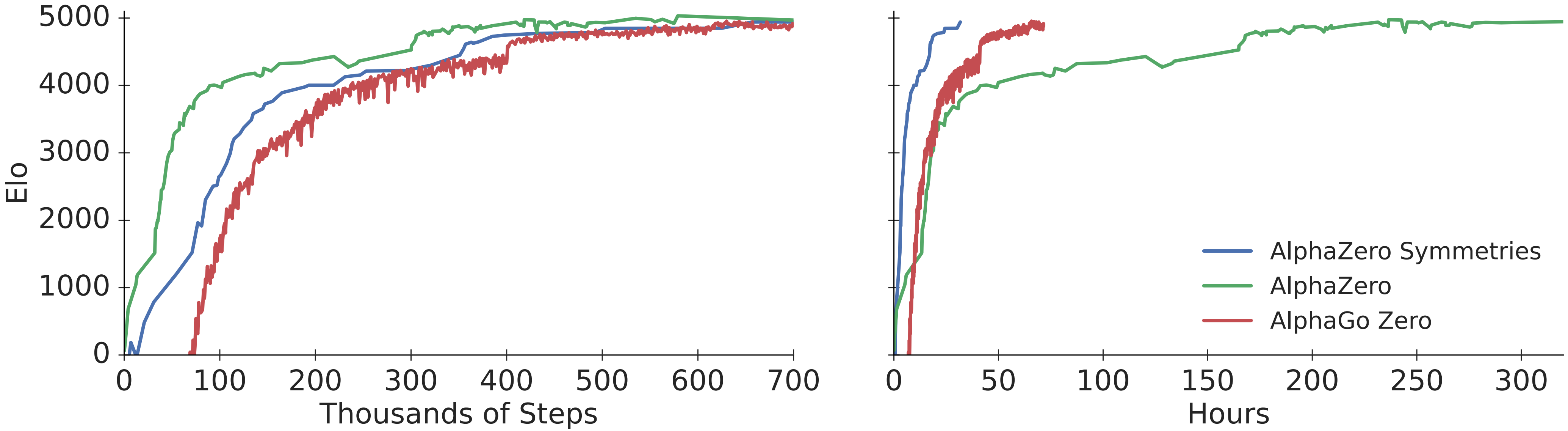


Figure S1 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Inference

During inference, AlphaZero utilizes much less evaluations than classical game playing programs.

Program	Chess	Shogi	Go
AlphaZero	63k (13k)	58k (12k)	16k (0.6k)
Stockfish	58,100k (24,000k)		
Elmo		25,100k (4,600k)	
AlphaZero	1.5 GFlop	1.9 GFlop	8.5 GFlop

Table S4 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Ablations

Fig.	Match	Start Position	AlphaZero Book	AlphaZero Main	AlphaZero Inc	Opponent Book	Opponent Main	Opponent Inc	Opponent Program
2A	Main	Initial Board	No	3h	15s	No	3h	15s	Stockfish 8
2B	1/100 time	Initial Board	No	108s	0.15s	No	3h	15s	Stockfish 8
2B	1/30 time	Initial Board	No	6min	0.5s	No	3h	15s	Stockfish 8
2B	1/10 time	Initial Board	No	18min	1.5s	No	3h	15s	Stockfish 8
2B	1/3 time	Initial Board	No	1h	5s	No	3h	15s	Stockfish 8
2C	latest Stockfish	Initial Board	No	3h	15s	No	3h	15s	Stockfish 2018.01.13
2C	Opening Book	Initial Board	No	3h	15s	Yes	3h	15s	Stockfish 8
2D	Human Openings	Figure 3A	No	3h	15s	No	3h	15s	Stockfish 8
2D	TCEC Openings	Figure S4	No	3h	15s	No	3h	15s	Stockfish 8

Table S8 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

Fig.	Match	Start Position	AlphaZero Book	AlphaZero Main	AlphaZero Inc	Opponent Book	Opponent Main	Opponent Inc	Opponent Program
2A	Main	Initial Board	No	3h	15s	Yes	3h	15s	Elmo
2B	1/100 time	Initial Board	No	108s	0.15s	Yes	3h	15s	Elmo
2B	1/30 time	Initial Board	No	6min	0.5s	Yes	3h	15s	Elmo
2B	1/10 time	Initial Board	No	18min	1.5s	Yes	3h	15s	Elmo
2B	1/3 time	Initial Board	No	1h	5s	Yes	3h	15s	Elmo
2C	Aperyqhapaq	Initial Board	No	3h	15s	No	3h	15s	Aperyqhapaq
2C	CSA time control	Initial Board	No	10min	10s	Yes	10min	10s	Elmo
2D	Human Openings	Figure 3B	No	3h	15s	Yes	3h	15s	Elmo

Table S9 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

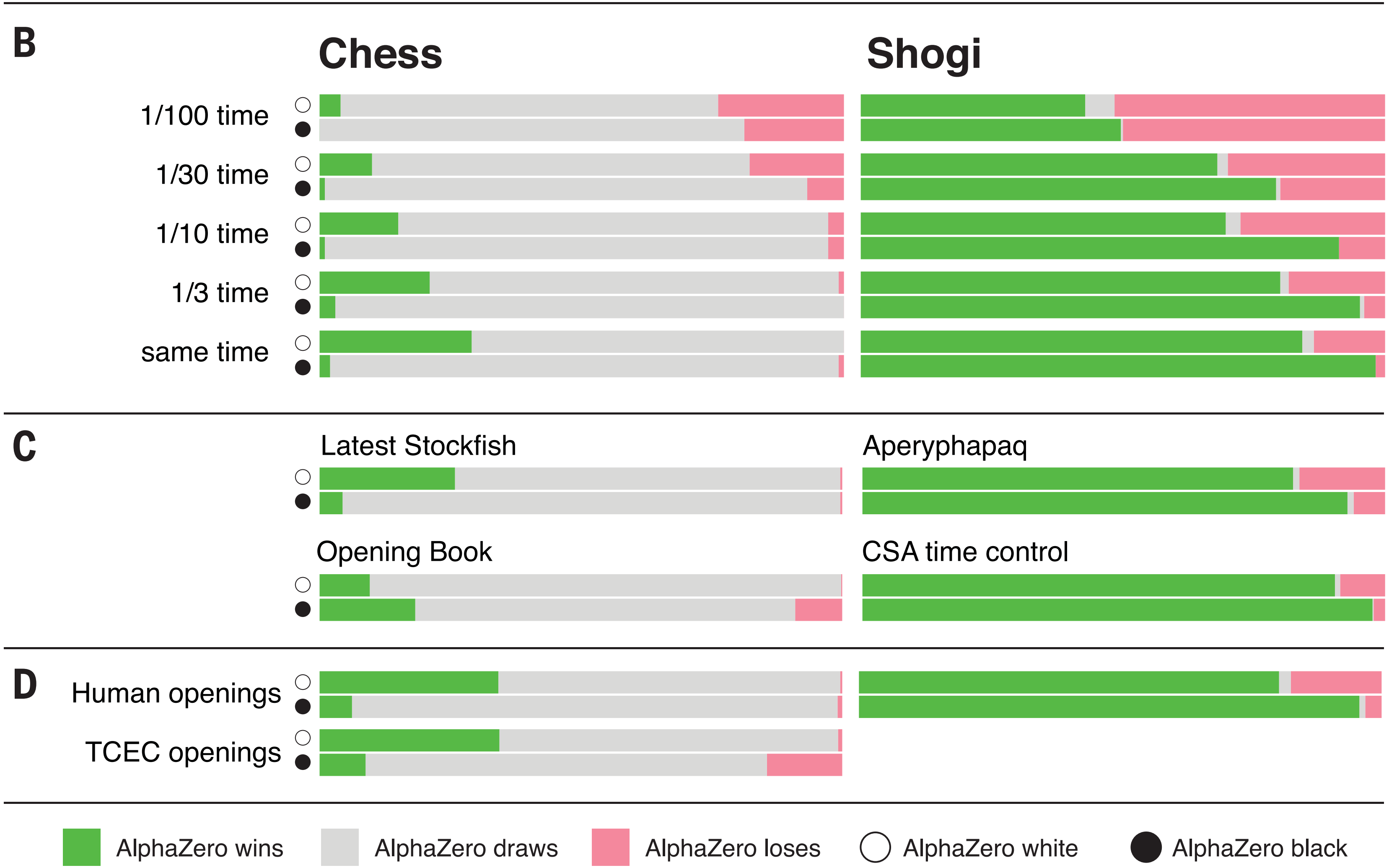


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero – Ablations

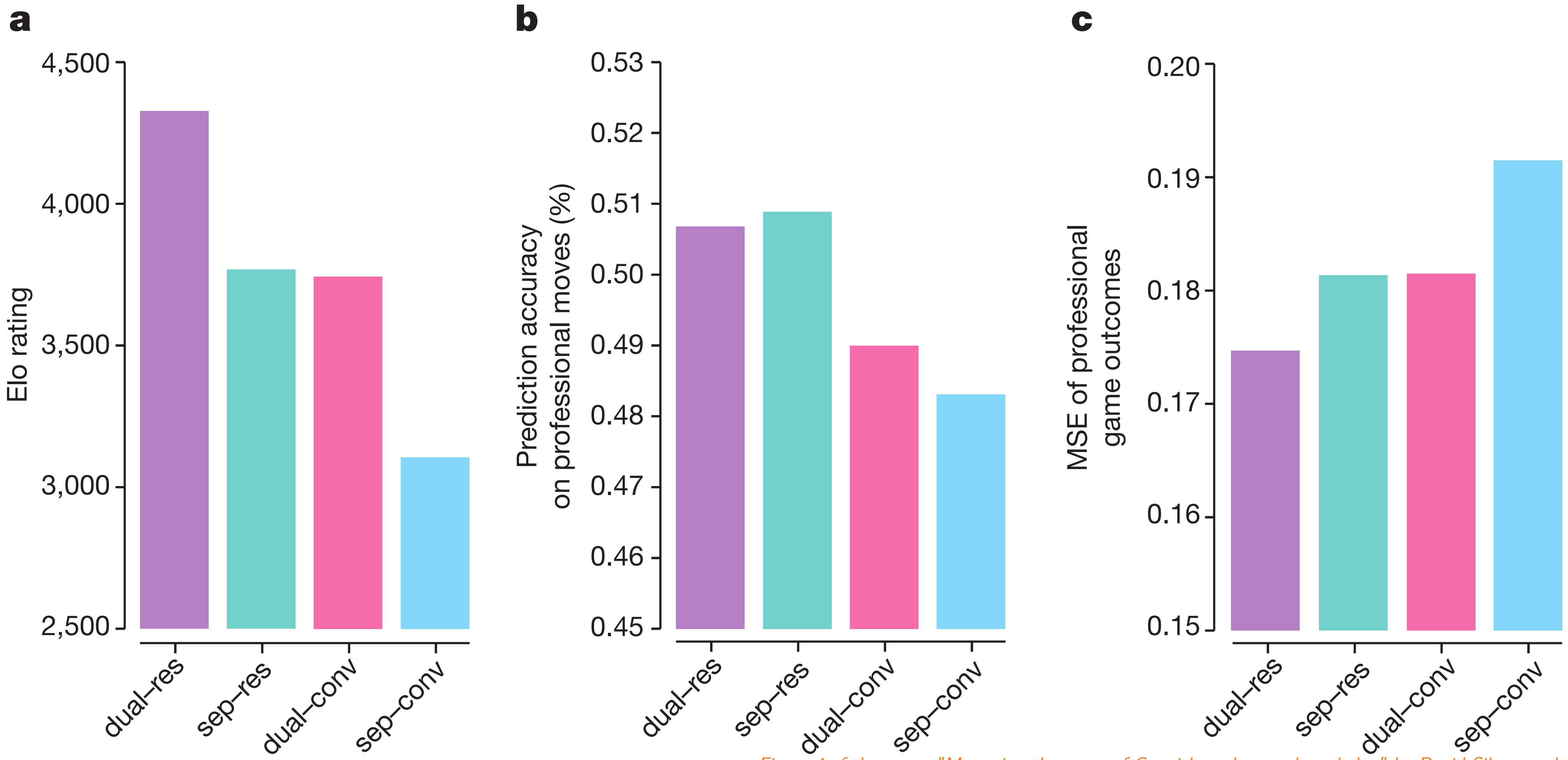


Figure 4 of the paper "Mastering the game of Go without human knowledge" by David Silver et al.

AlphaZero – Preferred Chess Openings

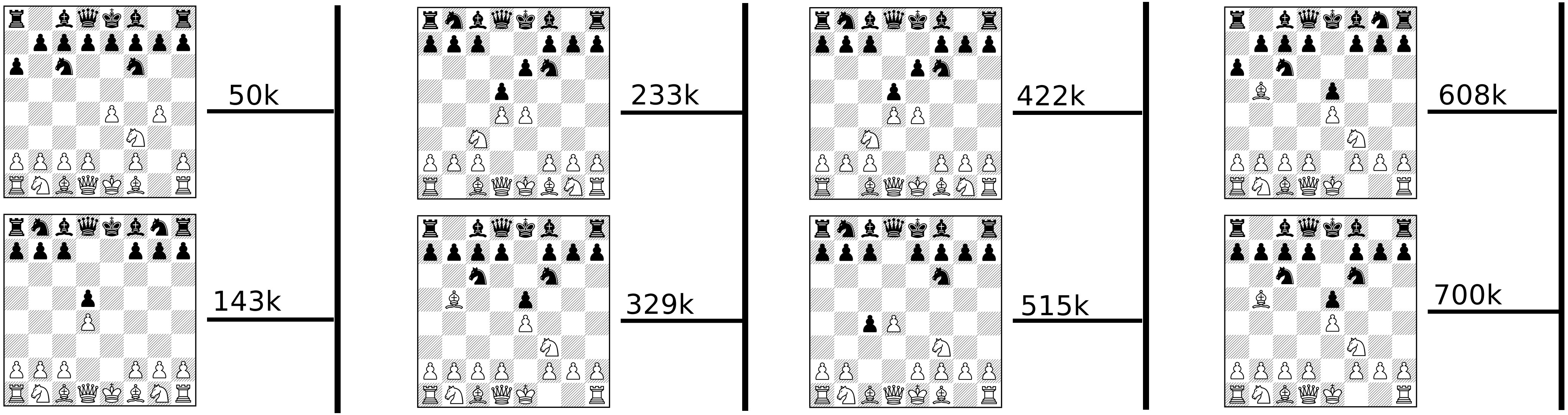


Figure S2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.