Improving AlphaZero Using Monte-Carlo Graph Search

Johannes Czech [1], **Patrick Korus** [1], **Kristian Kersting** [1, 2, 3]
[1] Department of Computer Science, TU Darmstadt, [2] Centre for Cognitive Science, TU Darmstadt, [3] Hessian Center for Artificial Intelligence, Darmstadt, Germany

Summary

The *AlphaZero* algorithm has been successfully applied in a range of discrete domains, most notably board games. It utilizes a neural network that learns a value and policy function to guide the exploration in a Monte-Carlo Tree Search. We improve the search algorithm for *AlphaZero* by generalizing the search tree to a directed acyclic graph. This enables information flow across different subtrees and greatly reduces memory consumption. Along with Monte-Carlo Graph Search, we propose a number of further extensions, such as the inclusion of ϵ-greedy exploration, a revised terminal solver and the integration of domain knowledge as constraints. In our empirical evaluations, we use the *CrazyAra* engine on chess and crazyhouse as examples to show that these changes bring significant improvements to *AlphaZero*.

Motivation

Utilizing transpositions is beneficial for many domains which can be formalized as a Markov decision process with a discrete action space. As an example domain, we consider the game of chess and the chess variant crazyhouse. Here, one can obtain the King's Knight Opening position using four different move orderings, as shown in Figure 1. The corresponding search graph can either be modeled as a regular search tree or a Directed Acyclic Graph (DAG) which is both more compact and shares information between trajectories.

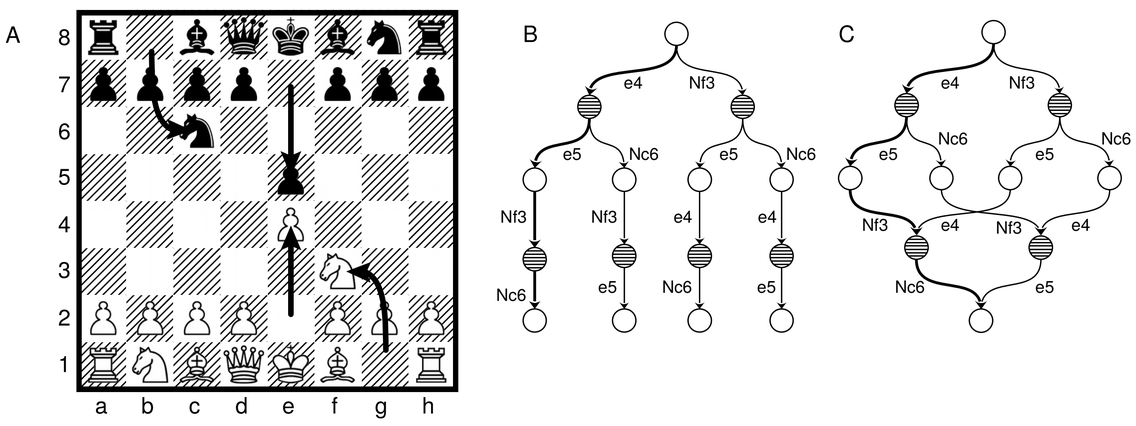

Figure 1: It is possible to obtain the King's Knight Opening (A) with different move sequences (B, C). Trajectories in bold are the most common move order to reach the final position. As one can see, graphs are a much more concise representation.

Moreover, humans excel at making valid connections between different subproblems and to reuse intermediate results for other subproblems. In other words, humans not only look at a problem step by step but are also able to jump between sequences of moves, so called trajectories; they seem to have some kind of *global memory buffer* to store relevant information. This gives a crucial advantage over a traditional tree search, which does not share information between different trajectories, although an identical state may occur. Triggered by this intuition, the search tree is generalized to a DAG, yielding Monte-Carlo Graph Search (MCGS).

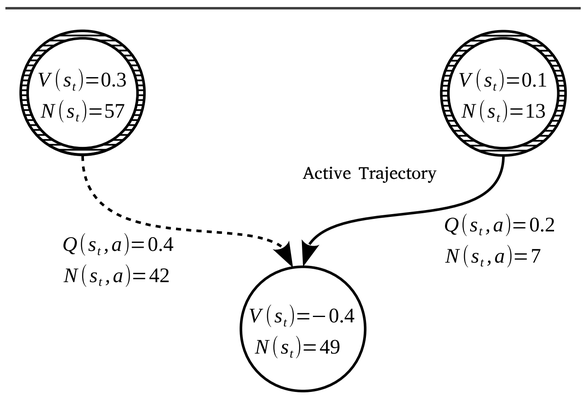

Figure 2: Scenario of accessing a child node on two possible trajectories from different parent states. The proposed data structure stores the visits and Q-values both on the edges and in the nodes.

Data Structure

Regarding the data structure, we need to consider transposition nodes, i.e nodes which have more than a single parent node, as shown in Figure 2. We store the Q-values $Q(s_t, a)$ and value estimation $V(s_t)$ to measure and correct the information leak. This procedure is both integrated in the selection and backpropagation process.
Furthermore, we can no longer backpropagate a value evaluation by storing a pointer to the leave node, but instead we store the trajectory to be later used in the back-up phase.

Correcting the Information Leak

To measure the current information leak, we calculate the residual Q_δ of our current Q-value belief $Q(s_t, a)$ compared to the more precise value estimation $V(s_{t+1})$

$$Q_\delta(s_t, a) = Q(s_t, a) - V(s_{t+1}) \; . \tag{1}$$

In the selection phase, we check if $|Q_\delta| \leq Q_\epsilon$, e.g. ≤ 0.05, and proceed to expand a new leaf node. Otherwise, we compute the correction value $Q_\phi(s_t, a)$ which brings $Q(s_t, a)$ closer to $V(s_{t+1})$

$$Q_\phi(s_t, a) = N(s_t, a) \cdot Q_\delta(s_t, a) + V(s_{t+1}) \; . \tag{2}$$

To ensure that we backpropagate a well-defined value, we clip our correction value to be within $[V_{min}, V_{max}]$, i.e.,

$$Q'_\phi(s_t, a) = \max(V_{\text{min}}, \min(Q_\phi(s_t, a), V_{\text{max}})) \; . \tag{3}$$

Empirical Evaluation

We evaluate the benefits of our MCGS and four additional search modifications empirically. In our first experiment as shown in Figure 3, we compare the scaling behaviour in crazyhouse between our presented MCGS algorithm and the *AlphaZero* search algorithm that makes use of a transposition table to reuse pre-computed evaluations. We observe that MCGS outperformed the transposition look-up table approach across all time controls, demonstrating that MCGS can be implemented efficiently and excels by providing a more expressive information flow along transposition nodes. In particular, it becomes apparent, that the Elo gap between the two algorithms slightly increases over time, suggesting an even better performance in the long run or when executed on stronger hardware. Beyond the MCGS algorithm, we propose and evaluate a set of additional independent enhancements to the *AlphaZero* planning algorithm: combining the PUCT algorithm with ϵ-greedy search adding domain specific constraints to narrow the search, using Q-value information for move selection, and incorporation a terminal solver which can choose faster wins and can handle table bases. A direct comparison is illustrated in Figure 4.

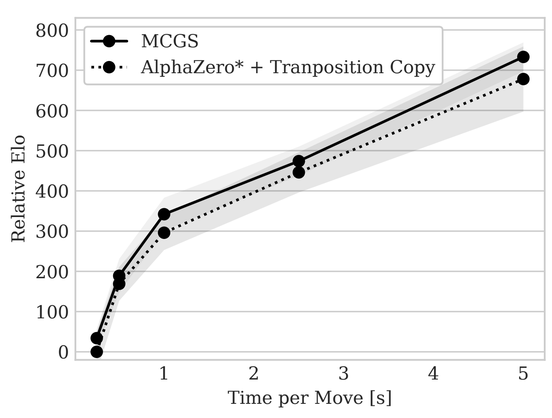

Figure 3: Elo development in crazyhouse over time of MCGS compared to MCTS which uses a hash table as a transposition buffer to copy neural network evaluations.

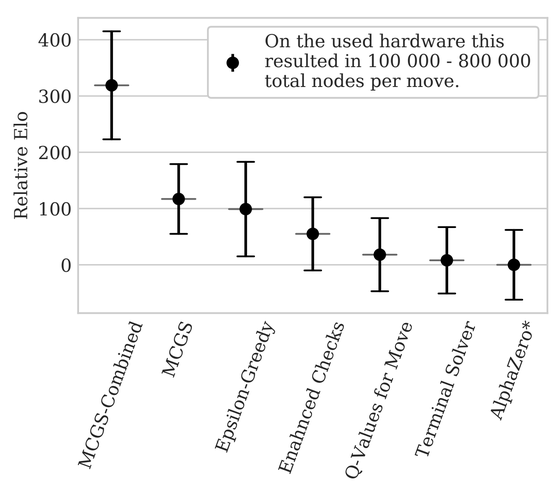

Figure 4: Elo comparison of the proposed search modification in crazyhouse using five seconds per move.

Presentation at International Conference on Automated Planning 2021
https://github.com/QueensGambit/CrazyAra