AlphaSnake: Policy Iteration on a Nondeterministic NP-Hard Markov Decision Process (Student Abstract)

Kevin Du[1], **Ian Gemp**[2], **Yi Wu**[3], **Yingying Wu**[4,5,*]
[1] Harvard University, Cambridge, U.S.
[2] DeepMind, London, U.K.
[3] Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
[4] Center of Mathematical Sciences and Applications, Harvard University, Cambridge, U.S.
[5] University of Houston, Department of Mathematics, Houston, U.S.
ywu68@uh.edu

Abstract

Reinforcement learning has been used to approach well-known NP-hard combinatorial problems in graph theory. Among these, Hamiltonian cycle problems are exceptionally difficult to analyze, even when restricted to individual instances of structurally complex graphs. In this paper, we use Monte Carlo Tree Search (MCTS), the search algorithm behind many state-of-the-art reinforcement learning algorithms such as AlphaZero, to create autonomous agents that learn to play the game of Snake, a game centered on properties of Hamiltonian cycles on grid graphs. The game of Snake can be formulated as a single-player discounted Markov Decision Process (MDP), where the agent must behave optimally in a stochastic environment. Determining the optimal policy for Snake, defined as the policy that maximizes the probability of winning — or win rate — with higher priority and minimizes the expected number of time steps to win with lower priority, is conjectured to be NP-hard. Compared to prior work in the Snake game, our algorithm is the first to achieve a win rate over 0.5, compared with a uniform random policy, which achieves a win rate $< 4.16 \times 10^{-17}$, demonstrating the versatility of AlphaZero in tackling NP-hard problems.

Introduction

General-purpose reinforcement learning algorithms have made significant progress in achieving superhuman performance in games (Silver et al. 2018). Recent work has investigated the potential of applying these algorithms to complexity theory — in particular, to solve NP-hard graph problems for which there are no known polynomial time solutions. A common framework for formulating these problems is the deterministic MDP (Abe et al. 2019), where an agent optimizes its performance in a deterministic environment.

In this paper, however, we formulate the game of Snake as a nondeterministic environment in which the agent must maximize its expected rewards, in resemblance to the inherent uncertainty in real-world problems. Prior work has used Deep Q-Learning and tree-based Monte Carlo models to train autonomous agents in Snake, but our work is the first to use AlphaZero's policy iteration and search algorithms. To our best knowledge, our algorithm is the first to achieve a win rate over 0.5, compared with a uniform random policy, which achieves a win rate $< 4.16 \times 10^{-17}$, computed with a dynamic programming paradigm.

*corresponding author

The Snake Environment

The game of Snake is played on an $n \times n$ board, where the player controls the movement of the snake head, moving one unit in three of the four orthogonal directions every time step. The snake body follows the head's movements and grows by one unit when the snake eats an apple, increasing the score of the player by one. Once the apple is eaten, another apple is placed in an empty cell uniformly at random. The snake head can not intersect the snake body, and if the head is surrounded by the body, the game is lost. Starting from a snake size of 2 units, the goal of the game is to maximize the length of the snake; the game is won when the snake body fills the grid, defining a Hamiltonian path on the $n \times n$ grid graph. Minimizing the win time, or number of time steps required to achieve the maximum snake size, is a secondary objective. Figure 1 shows various example game states.

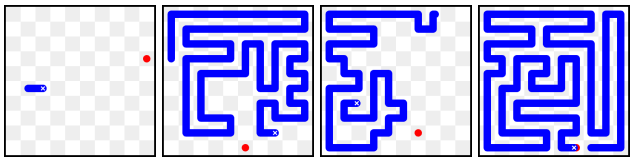

Figure 1: Game states from left to right: example (1) starting state (2) intermediate state (3) losing state (4) winning state.

Due to stochasticity in the environment, Snake can be formulated as a nondeterministic MDP, in which finding the optimal policy requires the agent to maximize its expected rewards. The rewards used in this project are $+1$ for eating an apple, -10 for a losing end state, and $+10$ for a winning end state. These rewards were chosen arbitrarily, but we found experimentally that this scheme was effective.

Deterministic Algorithms and Complexity

There exists a straightforward strategy that is guaranteed to win Snake with a worst-case win time of $\Theta(n^4)$ (See

Strategy	Average score	Win rate
AlphaZero algorithm	98.227	944/1000
Uniform random policy	6.277	0/1000
Hamiltonian cycle strategy	30.026	0/1000
Naive tree search	59.09	0/100

Table 1: Average score and win rate, as fraction of wins within 1,200 steps.

Supp. Eqn. 2). This is to simply follow a Hamiltonian cycle that traverses the $n \times n$ grid graph, which guarantees that the snake will not intersect itself and will eat any apple placed in the grid. However, this strategy does not minimize the expected win time, and is thus not optimal.

The optimal policy for Snake is defined as the policy that, for an arbitrary game state, maximizes the win rate with higher priority and minimizes the expected win time with lower priority. It is known that the optimal policy for Snake has an expected win time of $\Omega(n^3)$ (See Supp. Eqn. 3). It is also conjectured that determining the optimal policy for Snake is NP-hard, since the problem of extending an arbitrary partial path on a grid graph to a Hamiltonian cycle can be reduced to finding the optimal policy for Snake (See Supp. Problem 1). To better approximate this optimal policy, we adopt the algorithm behind AlphaZero.

Adapting AlphaZero to Discounted MDPs

To modify the AlphaZero algorithm to the game of Snake, a few differences between Snake and AlphaZero's target games — chess, Go, and Shogi — are identified:

Low action count: Snake has far fewer actions than Chess or Go, with at most three legal moves in any position. In fact, many states in Snake have only one possible action. Thus, in our implementation of the Snake environment, we remove these single-action states by simulating forced actions.

Long horizon: While the games of Chess and Go typically last under 300 moves, the game of Snake requires $\Omega(n^3)$ time steps in the worst-case scenario. Furthermore, intermediate rewards in Snake are easier to identify, since a higher score in intermediate game states is a strong signal of progress. Thus, we use a discounted MDP to model Snake.

To adapt the AlphaZero algorithm to MDPs, we will need a value network that gives, instead of the expected final outcome of the game as in AlphaZero, the discounted value:

$$V(s) = \mathbb{E}[R(s) + \gamma^{t(s')-t(s)} R(s') + \gamma^{t(s'')-t(s)} R(s'') + \cdots]$$

where $s, s', s'', \ldots$ are the future states of the game, $R(s)$ is the reward given in the state s, and $t(s)$ is the time index in state s. Just like in AlphaZero, MCTS with leaf evaluation given by the value network is used to compute the improved policy π, which is used to simulate rollouts. However, instead of two players who alternate between minimizing and maximizing the expected value, a single player seeks to maximize the value while a "chance" agent branches into new nodes uniformly at random to simulate stochasticity.

The value network is trained with mean-squared loss on the estimates of discounted return obtained from rollouts.

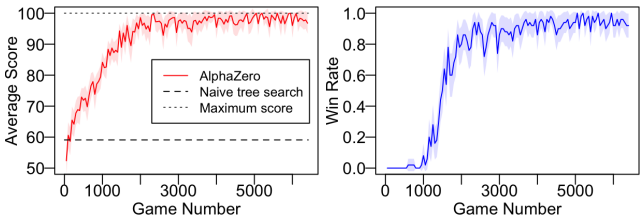

Figure 2: (Left) Graph of scores in comparison to Naive Tree Search, (Right) Graph of win rate averaged every 50 games. Shaded areas are 2σ error bars.

The policy network is trained with cross-entropy towards the state-action counts of tree search (See Supp.).

Experiments

In our experiments, we ran the above algorithm on a 10×10 game, using a tree search size of 200 states for 6,000 games. The resulting performance of the algorithm is seen in Figure 2. Table 1 compares this performance with several other algorithms for Snake: a random policy, the Hamiltonian cycle strategy, and a naive tree search (without policy or value predictions) with a search size of 10,000 states.

Conclusion

Our work shows that AlphaZero is resilient in performing in a nondeterministic game, developing strategies that are successful irrespective of chance. The game of Snake may be a relevant model for real-world security systems and anti-poaching strategies that often require protecting targets from adversary response (De Nittis and Trovo 2016). Generalizing AlphaZero to stochastic multiagent environments appears to be promising for tackling real-world situations.

Acknowledgments

We would like to thank Clifford Taubes, Hui Jiang, Richard Peng, and Roger Fu for helpful discussions.

References

Abe, K.; Xu, Z.; Sato, I.; and Sugiyama, M. 2019. Solving np-hard problems on graphs with extended alphago zero. *arXiv preprint arXiv:1905.11623*.

De Nittis, G.; and Trovo, F. 2016. Machine learning techniques for stackelberg security games: a survey. *arXiv preprint arXiv:1609.09341*.

Haythorpe, M. 2018. FHCP challenge set: The first set of structurally difficult instances of the hamiltonian cycle problem. *Bulletin of the ICA*, 83: 98–107.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.