AlphaZero Techniques for Classical Planning

Serawork Wallelign [1], **Humbert Fiorino** [2], **Damien Pellier** [2]
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble France
[1]serawork-amsalu.wallelign@univ-grenoble-alpes.fr, [2]firstname.lastname@imag.fr

Abstract

Integrating planning and learning to improve performance and knowledge from planning experience has been a long time research interest. Combining these two areas of AI, AlphaZero made a breakthrough in board games achieving superhuman performance in Chess, Shogi, and Go, each with under 24 hours of self-play, using almost no specialized knowledge of the games other than the rules. In this work, techniques similar to AlphaZero are implemented to solve automated planning problems. Similar to AlphaZero, our implementation alternates between neural network evaluation and Monte Carlo Tree search policy improvement for a stable learning. We experimented on benchmark classical domains....

Automated planning is the process consisting in choosing a sequence of actions to bring the state of the world to a targeted goal within a user-supplied model of an environment (Pellier and Fiorino 2018). Most of the state-of-the art planning is guided by heuristic search derived from problem descriptions in Strips and other action languages to narrow down the search space (Bonet and Geffner 2008) and explore the most promising regions which may lead to optimal solutions. Most of existing heuristic functions compute the heuristic value every time a state is encountered and do not learn from past experience.

With the rapid development of machine learning techniques especially deep learning, there is an increased interest in integrating planning and learning (Yoon, Fern, and Givan 2006; Arfaee, Zilles, and Holte 2011; Toyer et al. 2017, 2020; Shen, Trevizan, and Thiébaux 2020). The earlier research in applying machine learning to planning mainly focuses on learning a mapping function from a state to its heuristic value for a particular planning task (Yoon, Fern, and Givan 2006; Arfaee, Zilles, and Holte 2011). Though these approaches are good to investigate and understand the parameter of neural network from simple problems (Ferber, Helmert, and Hoffmann 2020), their applicability is limited since they do not generalize beyond a single task. Recent works, however, are more focused on learning domain-independent heuristics. For instance, Shen, Trevizan, and Thiébaux proposed a new model that predicts heuristic values even for domains that that the model has not been trained on.

Learning policy is another approach in applying machine learning for planning (Gomoluch, Alrajeh, and Russo 2019; Shen et al. 2019) though it has not been studied as much as heuristic learning. Here, instead of mapping a state to its heuristic value, the network learns to map perceived states of the environment to actions to be taken. Because the network maps states to actions, policy learning is domain dependent. This work combines policy and heuristic learning in a single model. The current implementation is domain-dependent where the model generalizes only for a domain that it has been trained on; however, we have not included any domain specific knowledge to the algorithm other than the rules of the environment.

Our approach is based on Monte Carlo Tree Search (MCTS) and deep reinforcement learning that learns policy and heuristic functions. Planning new polices is performed by MCTS while the deep neural network generalizes these policies. The implementation is based on AlphaZero (Silver et al. 2018), that learned policies to beat best performing players in Go, Chess and Shogi. However, systems like AlphaZero work with a fixed state space and fixed objective, in contrast to AI planning that deals with a wide variety of different planning tasks (Ferber, Helmert, and Hoffmann 2020). This work focuses on designing an algorithm that generalizes for tasks within a given domain.

The contribution of the work are:

- A new state representation that reflects the representation in most planning algorithms.
- A training approach: In addition to a powerful algorithm, AlphaZero's success is attributed to the computational resources used to train the model. We proposed a training approach to train the model with few resources and feasible time.

The rest the paper is organized as follows: We start by providing a brief background about planning and learning; followed by an explanation about the proposed algorithm. Then, we present the experimental results obtained and conclude by recommending methods for future improvement.

Background

Planning This definition is according to (Ghallab, Nau, and Traverso 2004).

A planning domain, $\boldsymbol{D}$ is defined in terms of its states $\boldsymbol{S}$ and actions $\boldsymbol{A}$ and a transition function γ. States are described by facts, i.e a set of propositions (or predicates) that tells the properties of objects present in a task.

An action $a \, \epsilon \, \boldsymbol{A}$ is associated with three sets of state facts, Pre(a), Add(a), and Del(a) representing the precondition, add, and delete effects respectively. An action a is applicable to a state s iff $Pre(a) \subseteq s$, and the application of an action a to s, results in the new state $a(s) = (s \setminus Del(a)) \cup Add(a)$. This relationship is formally described in a state-transition function γ: $\boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{S}$.

Given a planning domain, a planning problem is a tuple (s, A, g), where $A \subseteq \boldsymbol{A}$ is a set of actions, $s \, \epsilon \, \boldsymbol{S}$ is the initial state, and g is a set of state facts representing the goal. A solution plan for a planning problem is a sequence of actions $(a_1, ..., a_l)$ from A, where the sequential application of the sequence starting in state s leads to a goal state s' where $g \subseteq s'$.

Reinforcement Learning

Reinforcement learning (RL) deals with the problem of how an agent can learn a decision making process through trial and error interaction with a dynamic environment (Sutton and Barto 2018). RL can be modeled as a Markov decision process which is a mathematical framework used for modeling decision-making problems where the outcomes are partly random and partly controllable. The goal of RL is to find a mapping from states to actions, policy π, that picks actions a in given states s maximizing the cumulative expected reward. Here we use a probabilistic policy where it draws an action, a from a distribution $a \sim \pi(s, a) = P(a|s)$. For learning to happen, the agent needs guidance to find the relationship between state, action and reward. The training samples are generated by using MCTS guided by the previously trained network as shown in figure 1.

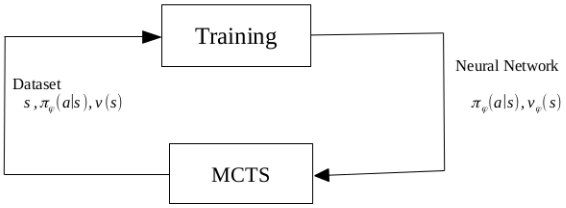

Figure 1: Iterative data generation and training

Monte Carlo Tree search

Monte Carlo Tree Search is a best-first search technique which uses stochastic simulations and was first devised for the game of Go programs (Chaslot et al. 2008). However, its application grows beyond game of Go where its variations are used in other board games and classical planning. Generally, MCTS has four steps which are repeated for x number of simulations as shown in figure 2.

- Selection: selects a node on the tree until a leaf node is reached
- Expansion: this step is performed when a node that has not been seen before is reached
- Simulation/Rollouts: From the leaf node, simulations are performed until termination condition is reached, i.e win/draw/loss.
- Backpropagation: Finally, the result from simulation is propagated back to the root.

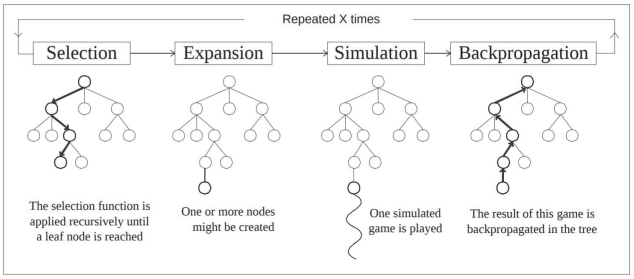

Figure 2: Steps of MCTS (Chaslot et al. 2008)

MCTS needs several simulation to converge which could be challenging to train very complex domains. To speed up optimization, domain-knowledge or domain-independent evaluation in the form of a neural network is often used to evaluate nodes. In AlphaZero, as well as our implementation, a neural network evaluation function is used instead of rollouts to evaluate a leaf node.

AlphaZero for planning

This section discuses the implementation of AlphaZero for classical planning. We will start by a brief introduction of AlphaZero (refer to Silver et al. for detail explanation) followed by its adaptation for planning.

AlphaZero uses a deep neural network, constructed from several layers of convolutional networks, $(\mathbf{p}, v) = f_\theta(s)$ with parameters θ. The neural network $f_\theta(s)$ takes the board position s as an input and outputs a vector of move probabilities $\mathbf{p}$ with components $p_a = Pr(a|s)$ for each action a, and a scalar value v estimating the expected outcome z of the game from position s, $v \approx E[z|s]$. AlphaZero learns these move probabilities and value estimates entirely from self-play; these are then used to guide its search in future games.

During self-play, a MCTS search is executed, guided by the current neural network $f(\theta)$ to narrow down the search to the most promising moves and to evaluate positions in a tree. For each search, several simulations of the game are performed to compute the move probabilities Π and the game score Z. The neural network weights are then updated by gradient descent on the loss function l to make the prediction close to the probabilities and the game outcomes of the MCTS. The updated network is used in subsequent self-play.

$$(\mathbf{p}, v) = f_\theta(s) \qquad l = (Z - v)^2 - \Pi^T log\mathbf{p} + c||\theta||^2 \quad (1)$$

where the last term is L_2 weight regularization.

Our implementation differs from AlphaZero in the following points. In games, the terminal condition is either a win, loss or a draw. For each game during self-play, AlphaZero plays for a specified number of moves and determines the reward +1, -1 or 0 depending on the winner of the game. However, due to the either all or nothing property of planning, evaluating at the terminal condition is challenging especially when the goal condition is not satisfied. We followed two approaches to evaluate at the terminal condition:

0/1 reward Setting the maximum action to take during self-play to the heuristic value of the initial state as estimated by fast-forward heuristic,

$$reward = \begin{cases} 1, & \text{if goal reached within the given length} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Using existing heuristic function to compute reward: using sparse reward results in slower learning since at the beginning of the training, the network is not capable of solving most of the problems. To speedup the optimization, we used an existing heuristic function to compute the reward using the equation below setting the maximum steps to $min(h(S\_init), T)$,

$$reward = \begin{cases} 1, & \text{if goal reached} \\ 0.9^{h}, & \text{if } h(S\_init) \leq T \\ 0.9^{h+T-h(S\_init)}, & \text{if } h(S\_init) > T \end{cases} \tag{3}$$

where $h(S\_init)$ is the heuristic value of the initial state and T is the maximum number of steps during self-play

AlphaZero for Single Player

Here instead of taking the planning problem as a zero-sum game where the value of a state is between 0 and 1, a state can take any positive value which is proportional to the length to the goal state. The difference between the above approach is in the backpropagation phase of the MCTS. In the previous case, after simulations the value of the leaf node is either 0/1 or a value inversely proportional to the heuristic value of the leaf node depending on the type of reward used. However, here the negative of the heuristic value is propagated back and each state between the initial state and the leaf node have different values.

Representation

Most of the earliest heuristic learning methods used a fixed length, hand engineered features to represent a planning problem. Since selecting features requires domain knowledge, the performance of the final model depends on these features (Arfaee, Zilles, and Holte 2011). In addition, the need of domain knowledge makes this approach to be effective only for a single task in a domain. Learning features automatically from data has been a research interest in machine learning (Bengio, Courville, and Vincent 2013) as well as in the context of planning (Yoon, Fern, and Givan 2008). Due to the success of convolutional networks for object recognition, (Loreggia et al. 2016) and (Sievers et al. 2019) encoded the problem description or the grounded problem as an image.

Due to the textual nature of problem and domain descriptions, we use a Bag-of-Words (BoW) model (Zhang, Jin, and Zhou 2010) to represent states. The BoW model is a way of representing text data and has seen a great success in natural language processing and document classification. It uses a vocabulary of known words and a measure of occurrence of known words to represent a text. To represent a state in a planning task, the vocabulary becomes all the facts in the domain and if the fact is true in the state it is set otherwise it is zero. The total number of facts in a domain vary depending on the number of objects in the problem however neural networks take fixed length inputs. Therefore, in this work, to get a fixed length representation for all instances in the domain, the maximum number of objects is fixed to some specified number. This limits the models applicability only to problems with objects less than the maximum.

Network architecture

The type of neural network architecture to use is dependent on the task to be modeled and the data. We use a type of recurrent neural network, Long Short Term Memory networks (LSTMs), which are suitable for learning dependencies in sequential data. The neural network we used has an input layer followed by four layers of LSTM and a two head output layer (for policy and value). To achieve problem independence, the goal is encoded together with the state. Therefore, the number of nodes in the input layer is twice the maximum number of facts. Different number of nodes are used for LSTM layers depending on the complexity of the domain.

Training algorithm

The training alternates between data generation using self-play and neural network training. At the beginning, the neural network is initialized with random weights. Then using the current neural network predicting the prior probability and value estimation for states, starting from the root node, S_{root} MCTS is conducted traversing a tree selecting an action with low visit count (not previously frequently explored), high move probability and high value. At the end of the simulation a probability distributions over all actions, $\pi_a = Pr(a|S_{root})$ is returned. To encourage exploration at the beginning, an action is selected with probability π, however, later an action with maximum probability is selected. The search is repeated updating the root with the state reached when the selected action is applied to S_{root} until a goal is reached or the maximum steps are taken. At the end of the self-play, a reward of 1 or 0 is given for each state encountered depending on whether a goal is reached or not. The states encountered, probability of actions and the reward is added to the training data. The process is repeated for several problems until the collected training data reaches a specified amount. This data is then used to update the neural network weights.

Unlike games which are highly studied, the starting and terminal conditions are known, initial and goal states in planning problems varies that in turn affects their complexity.

Therefore, for each problem, different maximum number of steps is used during self-play and its value is determined as follows. For each problem P in training problems, solve a problem P_i using informed A* search. For problems where a solution is found with in some time limit, the length of the solution will be used as maximum steps during self-play. Otherwise, the problem is removed from the training data. The training procedure is summarized in Algorithm 1.

Algorithm 1: AlphaZero for planning

```
Input: Set of problems P = {P_1, P_2, P_3, ...};
Result: Trained model
Initialize neural network;
data = {} ;
for n iterations do
    for each problem P_i in P do
        Get initial_state and goal;
        max episode, T_i= heuristic value of initial
          state;
        for m iterations do
            t=1;
            z=0;
            temp_data = {};
            S = initial_state;
            while not goal state and t < T_i do
                perform MCTS with S as root using
                  current NN;
                choose an action with probability
                  from MCTS;
                temp_data = temp_data ∪ (S, π, z,
                  goal);
                S ← (S, action);
                t ← t+1;
            end
            reward ← Evaluate the terminal state;
            for all samples in temp_data do
                set z=reward;
            end
            data = data ∪ temp_data;
        end
        train the neural network;
    end
end
```

Experimental evaluation

The algorithm is tested on two benchmark planning domains - Blocksworld and gripper. The total iterations, n, is set to 100; during each self-play 5000 new data is generated before retraining the neural network. A FIFO queue with 100000 maximum capacity is used. The choice of the maximum capacity affects the speed and stability of the training. Due to the fixed number of neural network inputs and outputs, problems containing maximum of 20 blocks and 42 balls for blocksworld and grippers respectively can be in the training and test problems.

The results are evaluated based on the number of nodes explored and CPU time comparing with existing heuristic functions. Figure below shows the experimental evaluations.

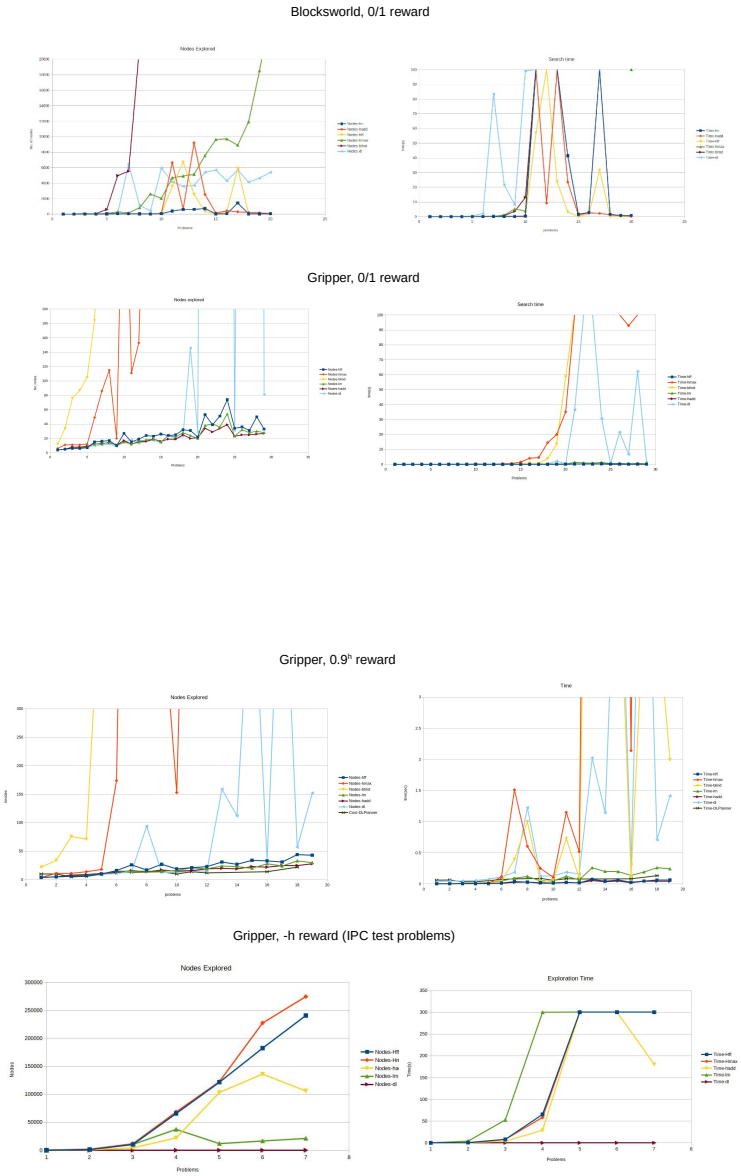

Figure 3: Plots of experimental results with different types of rewards - 0/1, inverse of heuristic value and negative of heuristic value

Conclusion and future work

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17): 2075–2098.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8): 1798–1828.

Bonet, B.; and Geffner, H. 2008. Heuristics for planning with penalties and rewards formulated in logic and computed through circuits. *Artificial Intelligence* 172(12-13): 1579–1604.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-Carlo Tree Search: A New Framework for Game AI. In *AIIDE*.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural network heuristics for classical planning: A study of hyperparameter space. ECAI.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Gomoluch, P.; Alrajeh, D.; and Russo, A. 2019. Learning classical planning strategies with policy gradient. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 637–645.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. A. 2016. Deep Learning for Algorithm Portfolios. In *AAAI*, 1280–1286.

Pellier, D.; and Fiorino, H. 2018. PDDL4J: a planning domain description library for java. *Journal of Experimental & Theoretical Artificial Intelligence* 30(1): 143–176.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 574–584.

Shen, W.; Trevizan, F. W.; Toyer, S.; Thiébaux, S.; and Xie, L. 2019. Guiding Search with Generalized Policies for Probabilistic Planning. In *SOCS*, 97–105.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep learning for cost-optimal planning: Task-dependent planner selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7715–7723.

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

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research* 68: 1–68.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2017. Action schema networks: Generalised policies with deep learning. *arXiv preprint arXiv:1709.04271* .

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9(Apr): 683–718.

Yoon, S. W.; Fern, A.; and Givan, R. 2006. Learning Heuristic Functions from Relaxed Plans. In *ICAPS*, volume 2, 3.

Zhang, Y.; Jin, R.; and Zhou, Z.-H. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1(1-4): 43–52.