Discrete Planning with End-to-end Trained Neuro-algorithmic Policies

Marin Vlastelica [1] **Michal Rolínek** [1] **Georg Martius** [1]

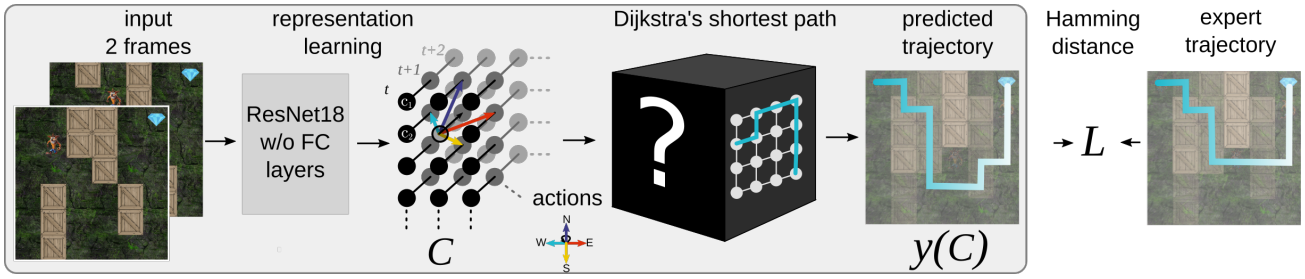

Figure 1. Architecture of the neuro-algorithmic policy. Two subsequent frames are processed by a ResNet18, whose output was modified to create a tensor (width $\times$ height $\times$ time) of vertex costs c_t^v. The time-dependent shortest path solver finds the shortest path to the goal. Hamming distance between the proposed and expert trajectory is used as a loss for training.

Abstract

Although model-based and model-free approaches to learning the control of systems have achieved impressive results on standard benchmarks, most have been shown to be lacking in their generalization capabilities. These methods usually require sampling an exhaustive amount of data from different environment configurations.

We propose a hybrid policy architecture with a deep network and a shortest path planner working in unison. The model can be trained end-to-end via blackbox-differentiation. The deep network learns to predict time-dependent way-costs such that internal plans match expert trajectories. These neuro-algorithmic policies generalize well to unseen environment configurations.

1. Introduction

One of the central topics in machine learning research is learning control policies for autonomous agents. Many different problem settings exist within this area. On one end of the spectrum are imitation learning approaches, where prior expert data is available and the problem becomes a supervised learning problem, i.e. imitating an expert. On the other end of the spectrum lie approaches that require interaction with the environment in order to obtain data for policy extraction, also known as the problem of exploration. Most Reinforcement Learning (RL) algorithms fall into this category of approaches. In this work, we concern ourselves primarily with the setting where limited data is available, and a policy needs to be extracted from it.

Independently of how a policy is extracted, a central question of interest is how well will it generalize to variations in the environment and the task. Recent studies have shown that standard deep RL algorithms require exhaustive amounts of exposure to environmental variability before starting to generalize (Cobbe et al., 2019).

There exist several approaches addressing the problem of generalization in control. One option is to employ model-based approaches that learn a transition model from data and use planning algorithms at runtime. However, learning a precise transition model is often harder than learning a policy. The reason for this is that policies are tailored to the specific problem at hand, whereas approaches involving learning global models consider data irrelevant for the optimal policy. This in turn makes them more general, but comes at a cost of increasing the problem dimensionality. This is particularly true for learning in problems with high-dimensional input space, such as image-based inputs. In order to alleviate this problem, learning specialized or partial models has shown to be a viable alternative, e.g. in MuZero (Schrittwieser et al., 2019).

We propose to use recent advances in making combinatorial algorithms differentiable in a blackbox fashion (Vlastelica et al., 2020) to train neuro-algorithmic policies with em-

[1]Max Planck Institute for Intelligent Systems. Correspondence to: Marin Vlastelica <marin.vlastelica@tue.mpg.de>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

bedded planners end-to-end. More specifically, we use a time-dependent shortest path planner acting on a temporally evolving graph generated by a deep network from the inputs. This enables us to learn the time-evolving costs of the graph,and connects us in a narrow sense to model-based approaches. Using neuro-algorithmic architectures facilitates generalization by shifting the combinatorial aspect of the problem to efficient algorithms, while using neural networks to extract a good representation for the problem at hand. They have potential to endow artificial agents with the main component of intelligence, the ability to reason.

Our contributions can be summarized as follows:

- A general differentiable policy architecture embedding shortest path algorithms.
- Demonstration of learning generalizing policies in a dynamic game environment from images.
- Showing that the policies enhanced with shortest path algorithms exhibit superior generalization performance in comparison to standard methods.

2. Related Work

Differentiable planning has been proposed in previous works, e.g. in the continuous case with CEM (Amos & Yarats, 2019; Bharadhwaj et al., 2020). Learning a representation for planning was considered, among others, in Hafner et al. (2019) but without differentiating though the planner. Silver et al. (2017) differentiate through a few steps of value prediction in a learned MDP to match the externally observed rewards. In Srinivas et al. (2018) a differentiable planner was obtained via relaxation and used to learn a representation.

Alternatively to learning the representation, a planning graph can be constructed from the agents' experience with a learned value function used for the edge costs, as done by Eysenbach et al. (2019).

An orthogonal research direction is to optimize the planners with prior experience, e.g. (Chen et al., 2020), that can be combined with our approach.

3. Markov Decision Processes and Shortest Paths

We follow the MDP framework (Puterman, 2014) in a goal-conditioned setting (Schaul et al., 2015). This is used in sequential decision making problems where a specific terminal state has to be reached.

Definition 1 *A goal-conditioned Markov Decision Process (gcMDP), $\mathcal{M}$ is defined by the tuple ($\mathcal{S}$, $\mathcal{A}$, p, g, r), where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $p(s'|a, s)$ the probability of making the transition $s \to s'$ when taking the action a, g is the goal, $r(s, a, s', g)$ the reward obtained when transitioning from state s to s' while taking action a and aiming for goal g.*

Concretely, we concern ourselves with fully observable discrete MDPs, in which the Markov assumption for the state holds and where the state and action-space are discrete. The goal of reinforcement learning is to maximize the return $G = \sum_{t=0}^{T} r_t$ of such a process. In gcMDPs the reward is such that the maximal return can be achieved by reaching the goal state g.

Given access to the transition probabilities and rewards, an optimal policy can be extracted by Dynamic Programming (Bertsekas et al., 1995). In a graph representation of an gcMDP, the set of vertices V corresponds to the set of states $\mathcal{S}$, traversing an edge corresponds to making a transition between states. Given the goal state and that the transition probabilities have Dirac densities, i.e. the process is deterministic, the optimal policy can be extracted by standard shortest path algorithms, such as Dijkstra's shortest path algorithm.

In this work, we imitate expert trajectories by training a policy with an embedded shortest path solver end-to-end. Although the actual gcMDP solved by the expert may be stochastic, we learn a deterministic latent approximate gcMDP, $\widehat{\mathcal{M}}$. Assuming that we have access to the topology of the gcMDP, by applying blackbox-differentiation theory (Vlastelica et al., 2020) we are able to learn the underlying costs (instead of rewards) of $\widehat{\mathcal{M}}$ such that the optimal policy on $\widehat{\mathcal{M}}$ is also optimal in $\mathcal{M}$. We consider cases where the reward function only depends on the current state and the goal: $r(s, a, s', g) = r(s, g)$. In this case, vertex costs c_t^v are sufficient for finding the optimal solution to the gcMDP.

4. Shortest Path Algorithm and its Differentiation

We will employ an efficient implementation of Dijkstra's algorithm for computing the shortest path. For differentiation, we rely on the framework for blackbox differentiation of combinatorial solvers (Vlastelica et al., 2020). This framework was already successfully used for ranking (Rolínek et al., 2020) and keypoint matching (Rolínek et al., 2020) problems.

4.1. Time-dependent Shortest Path

The purely combinatorial setup can be formalized as follows. Let $G = (V, E)$ be a graph. For every $v_i \in V$, let $c_i^1, \ldots, c_i^T$ be non-negative real numbers; the costs of reaching the vertex v_i at time-points $1, 2, \ldots, T$, where T is the planning horizon. The TIME-DEPENDENT-SHORTEST-PATH problem

Algorithm 1 Forward and backward Pass for the shortest-path algorithm

function FORWARDPASS(C, s, e)
  $Y := \textbf{TDSP}(C, s, e)$ *// Run Dijkstra's algorithm*
  save Y, C, s, e *// Needed for backward pass*
  return Y

function BACKWARDPASS($\nabla L(Y)$, λ)
  load Y, C, s, e
  $C_\lambda := C + \lambda \nabla L(Y)$ *// Calculate modified costs*
  $Y_\lambda := \textbf{TDSP}(C_\lambda, s, e)$ *// Run Dijkstra's algo.*
  return $\frac{1}{\lambda}(Y_\lambda - Y)$

(TDSP) has as input the graph G, a pair of vertices $s, e \in V$ (start and end) and the matrix $C \in \mathbb{R}^{|V| \times T}$ of the costs c_i^t. This version of the shortest path problem can be solved by executing the Dijkstra shortest path algorithm[1] (Dijkstra, 1959) on an augmented graph. In particular, we set

$$V^* = \{(v, t) \colon v \in V, t \in [1, T]\}$$
$$E^* = \{((v_1, t), (v_2, t+1)) \colon (v_1, v_2) \in E, t \in [1, T-1]\},$$

where the cost of vertex $(v_i, t) \in V^*$ is simply c_i^t. Note that $v_1 = v_2$ means to remain in the same vertex. In this graph, the task is to reach vertex (e, T) from $(s, 1)$ with the minimum traversal cost.

4.2. Applicability of Blackbox Differentiation

The framework presented in (Vlastelica et al., 2020) turns blackbox combinatorial solvers into neural network building blocks. The provided gradient is based on a piecewise linear interpolation of the true piecewise constant (possibly linearized) loss function. The *exact* gradient of this linear interpolation is computed efficiently via evaluating the solver on only one more instance (see Algorithm 1).

In order to apply this differentiation scheme, the solver at hand needs to have a formulation in which it minimizes an inner-product objective (under arbitrary constraints). To that end, for a given graph $G = (V, E)$ with time-dependent costs $C \in \mathbb{R}^{|V| \times T}$ we define $Y \in \{0, 1\}^{|V| \times T}$ an indicator matrix of visited vertices. In particular, $Y_i^t = 1$ if and only if vertex v_i is visited at time point t. The set of such indicator matrices that correspond to valid paths in the graph (V^*, E^*) will be denoted as $\mathrm{Adm}(G)$. The time-dependent shortest path optimization problem can be then rewritten as

$$\mathrm{TDSP}(C, s, e) = \underset{Y \in \mathrm{Adm}(G)}{\arg\min} \sum_{(i,t)} Y_i^t C_i^t. \qquad (1)$$

This is an inner-product objective and thus the theory from (Vlastelica et al., 2020) applies. In effect, the deep network producing the cost matrix C can be trained via supervision signal from ground truth shortest paths.

[1]Even though the classical formulation of Dijkstra's algorithm is edge-based, all of its properties hold true also in this vertex based formulation.

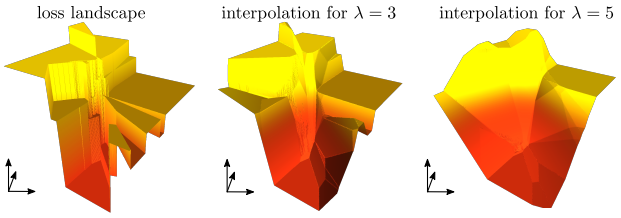

Figure 2. Differentiation of a piecewise constant loss resulting from incorporating a combinatorial solver. A two-dimensional section of the loss landscape is shown (left) along with two differentiable interpolations of increasing strengths (middle and right).

5. Cost Margin

Our work is related to the problem of metric learning in the sense that we learn the distance metric between the current position of the agent (state) and the target position in the underlying MDP, allowing us to solve it with a shortest path algorithm. It has been shown that inducing a margin on the metric can be beneficial for generalization. Similarly to (Rolínek et al., 2020) in the context of rank-based metric learning, we induce a margin α on the costs of the latent gcMDP, increasing the cost of ground truth path and decreasing the rest in the training stage of the algorithm:

$$c_i^t = \begin{cases} c_i^t + \frac{\alpha}{2} & \text{if } (v_i, t) \in Y^* \\ c_i^t - \frac{\alpha}{2} & \text{if } (v_i, t) \notin Y^* \end{cases} \forall\, (v_i, t) \in V^*. \qquad (2)$$

6. Crash Jewel Hunt Experiment

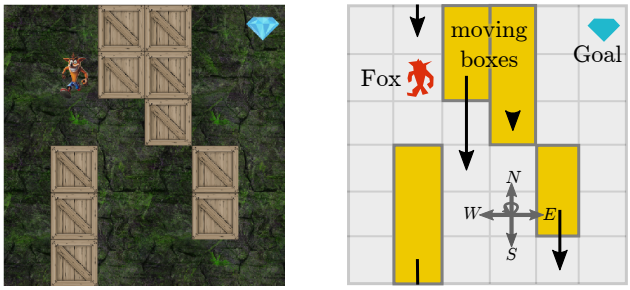

Figure 3. The Crash Jewel Hunt environment. Left: sample frame; right: schematics. The goal for the fox is to obtain the jewel in the right most column, while avoiding the moving wooden boxes (arrows). When the agent collides with a wooden box it instantly fails to solve the task.

In order to show that our method benefits generalization, we compare it to supervised imitation learning approach. Concretely, for the experimental validation, we look at the following points:

- Operating in a low data regime.
- Clear separation from train and test configurations of the environment.
- Sufficient combinatorial complexity to require non-trivial generalization.

To this end, we have constructed the Crash Jewel Hunt environment which can be seen in Fig. 3. The environment corresponds to a grid-world of dimensions $h \times w$ where the goal is to move the agent (Fox) from an arbitrary start position in the left-most column to the goal position (jewel) arbitrarily positioned in the right-most column. Between the agent and the goal are obstacles, wooden boxes that move downwards with arbitrary but fixed velocities or are static, see Fig. 3(right). At each time step, the agent can choose to move horizontally or vertically in the grid by one cell or take no action.

To make the task challenging, we sample distinct environment configurations for the training set and the test set, respectively. That is: we vary the velocities, sizes and initial positions of the boxes as well as the initial and goal positions.

6.1. Neuro-algorithmic Policy Architecture

The architecture of the policy consists of two main components: a modified ResNet18 architecture and the shortest path solver, see Fig. 1. At each time step the policy receives two images concatenated channel-wise based on which it predicts the cost matrix C for the planning horizon T. The cost matrix C is given to the solver along with the start vertex s and end vertex e to predict the time-dependent shortest path Y. Here we assume that we have access to the current position of the agent and the current goal position encoded in the latent graph. For training supervision we use the Hamming distance between the predicted plan Y and the expert plan Y^*.

As it is done in model-predictive control, at execution time we predict the plan Y for horizon T at each time step and execute the first action from the plan.

6.2. Results

To validate our claim that embedding planners into neural network architectures leads to better generalization, we compare to a standard ResNet18 architecture trained with a cross-entropy classification loss on the same dataset as our method. We train both approaches till saturation on a training set of a 1000 trajectories, resulting in close to 100% success rate when evaluating on train configurations in the environment. After training, we evaluate on a 1000 unseen environment configurations. The result is that our method with an embedded planner has almost perfect generalization performance. In contrast, the baseline generalizes poorly, in particular for larger environment sizes, see Fig. 4.

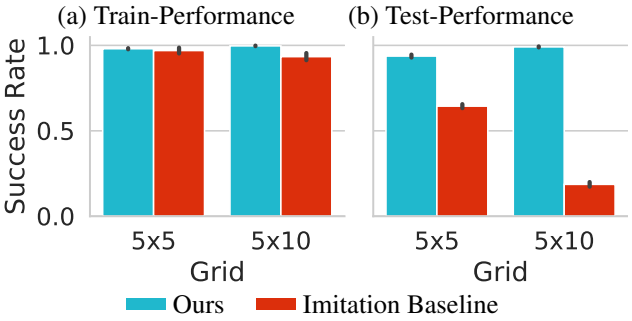

Figure 4. Performance on Crash Jewel Hunt. Our method and the baseline (for small level sizes) achieve almost perfect training performance (a). Unlike the baseline, our method is able to generalize well to unseen configurations.

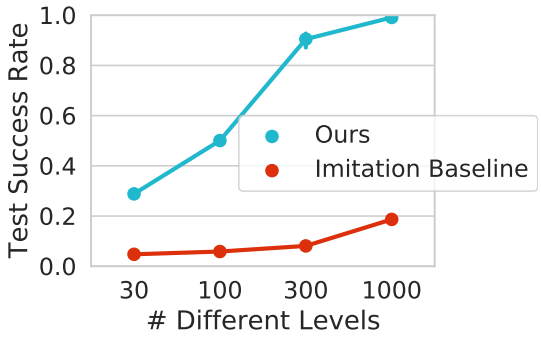

Figure 5. Generalization depending on # levels seen during training size. Shown is the success rate on the test set configurations of a 5x10 grid as dataset size increases. Our method outperforms the imitation baseline by an order of magnitude.

Furthermore, we observe that our method shows non-trivial generalization with an even smaller number of expert trajectories. Figure 5 displays the test-performance in dependence of the number of expert trajectories. Already with 30 trajectories a third of the 1000 test-levels can be solved, the baseline manages less than 50 out of the 1000.

7. Conclusion

We have shown that hybrid neuro-algorithmic policies consisting of a deep feature extraction and a shortest path solver – made differentiable via blackbox differentiation – enables learning policies that generalize to unseen environment settings.

8. Acknowledgment

We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Marin Vlastelica. We acknowledge the support from the German Federal Ministry of Education and Research (BMBF)

through the Tübingen AI Center (FKZ: 01IS18039B).

References

Amos, B. and Yarats, D. The differentiable cross-entropy method, 2019.

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

Bharadhwaj, H., Xie, K., and Shkurti, F. Model-predictive control via cross-entropy and gradient-based optimization. *arXiv preprint arXiv:2004.08763*, 2020.

Chen, B., Dai, B., Lin, Q., Ye, G., Liu, H., and Song, L. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2020.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint:1912.01588*, 2019.

Dijkstra, E. W. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959. doi: 10.1007/BF01386390.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems 32*, pp. 15246–15257. Curran Associates, Inc., 2019.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, volume 97 of *ICML'19*, pp. 2555–2565, Long Beach, California, USA, 2019.

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Rolínek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., and Martius, G. Optimizing ranking-based metrics with blackbox differentiation. In *Conference on Computer Vision and Pattern Recognition*, CVPR'20, 2020.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. Deep graph matching via blackbox differentiation of combinatorial solvers. *arXiv preprint arXiv:2003.11657*, 2020.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning*, volume 37 of *ICML*, pp. 1312–1320, 2015.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, volume 70 of *ICML'17*, pp. 3191–3199. PMLR, 2017.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, volume 80 of *ICML'18*, pp. 4732–4741, Stockholmsmässan, Stockholm Sweden, 2018.

Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, ICLR'20, 2020.