Discrete Planning with Neuro-algorithmic Policies

Marin Vlastelica[1], **Michal Rolínek**[1], **Georg Martius**[1]
{marin.vlastelica, michal.rolinek, georg.martius}@tue.mpg.de

1

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 introduce a neuro-algorithmic policy architecture with the ability to plan consisting of a model working in unison with a shortest path solver to predict trajectories with low way-costs. These policies can be trained end-to-end by blackbox differentiation. We show that this type of architecture generalizes well to unseen environment configurations.

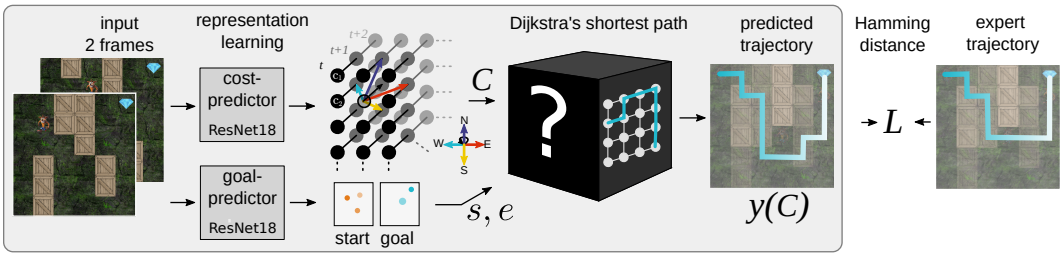


Figure 1: Architecture of the neuro-algorithmic policy. Two subsequent frames are processed by two simplified ResNet18s: the cost-predictor outputs a tensor (width $\times$ height $\times$ time) of vertex costs c_t^v and the goal-predictor outputs heatmaps for start and goal. The time-dependent shortest path solver finds the shortest path to the goal. Hamming distance between the proposed and expert trajectory is used as loss for training.

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.

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).

[1]Max Planck Institute for Intelligent Systems, Tübingen, Germany

1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020, Vancouver, Canada.

We propose to use recent advances in making combinatorial algorithms differentiable in a blackbox fashion as proposed by Vlastelica et al. (2020) to train neuro-algorithmic policies with embedded 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. We demonstrate the effectiveness of this approach in an imitation learning setting, where a few expert trajectories are provided. Due to the combinatorial generalization capabilities of planners, our learned policy is able to generalize to new variations in the environment out of the box and orders of magnitude faster than naive learners. 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 dynamic game environments from images.
- Showing that the policies enhanced with shortest path algorithms exhibit superior generalization performance in comparison to standard methods.

2 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, $p(s' \mid a, s)$ 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.

In this work, we imitate expert trajectories by training a policy with an embedded time-dependent 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 $\mathcal{M}$.

3 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 has already been successfully used for ranking Rolínek et al. (2020) and keypoint matching Rolínek et al. (2020) problems.

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 (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[2] Dijkstra (1959) on an augmented graph. In particular, we set

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

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

where the cost of vertex $(v_i, t) \in V^*$ is simply c_i^t and $E^{\hookrightarrow}$ is the original edge set E appended with all self-loops. This allows to "wait" at a fixed vertex v from timestep t to timestep $t+1$. In this graph, the task is to reach vertex (e, T) from $(s, 1)$ with the minimum traversal cost.

We can take advantage of this formulation of the time-dependent shortest path problem in a model-predictive control fashion in that we predict the time dependent costs. We learn this by applying blackbox differentiation from Vlastelica et al. (2020).

4 Neuro-algorithmic Policy Architecture

We propose the Neuro-algorithmic Policy (NAP) framework, that is an end-to-end trainable deep policy architecture embedding an algorithmic component using the afore-mentioned techniques. In this paper we consider the concrete architecture consisting of two main components: a backbone ResNet18 (without the final fully connected layers, similarly as in Vlastelica et al. (2020)) architecture and the shortest path solver, see Fig. 1. At each time step the policy receives two images concatenated channel-wise from which it predicts the cost matrix C for the planning horizon T with the *cost-predictor* and the start vertex s and end vertex e with the *goal predictor*, explained in the appendix.

The cost matrix C is given to the solver along with the start vertex s and end vertex e to compute the time-dependent shortest path Y. The *cost-predictor* is trained using the Hamming distance between the predicted plan Y and the expert plan Y^* that we use for supervision. Such a cost-prediction ca be seen in Fig. 2. It is necessary to know the start and goal vertex when applying the shortest-path solver for which we learn respective predictors, which we describe in appendix Sec. C.

The policy is used in a model-predictive control fashion, i.e. at execution time we predict the plan Y for horizon T at each time step and execute the first action from the plan.

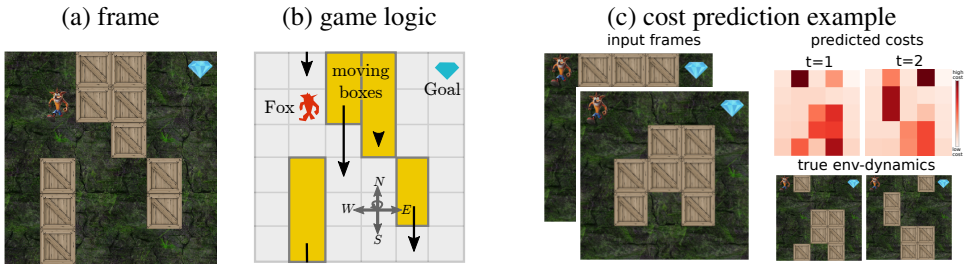


Figure 2: The CRASH JEWEL HUNT environment. The goal for the fox, see (a), is to obtain the jewel in the right most column, while avoiding the moving wooden boxes (arrows in (b)). When the agent collides with a wooden box it instantly fails to solve the task. A prediction of the costs is shown in (c).

5 Experiments

To validate our hypothesis that embedding planners into neural network architectures leads to better generalization, we consider several procedurally generated environments (from the ProcGen suite Cobbe et al. (2019) and CRASH JEWEL HUNT) with considerable variation between the realizations, called *levels*. A more detailed description of the environments can be found in section F of the appendix.

We compare to two baselines: a standard *imitation learning* baseline using a ResNet18 architecture trained with a cross-entropy classification loss on the same dataset as our method; and a reinforcement learning baseline using the PPO algorithm. We note that a comparison to PPO is to some extent unfair, since the algorithm also needs to handle the problem of exploration. Nevertheless, the PPO agent observes much more diverse data than our method, yet fails to generalize in comparison.

For the experimental validation, we aim to answer the following questions:

- Can NAP be trained to perform well as policies in procedurally generated environments?
- Can NAP generalize in a low data regime, i.e. after seeing only few different levels?

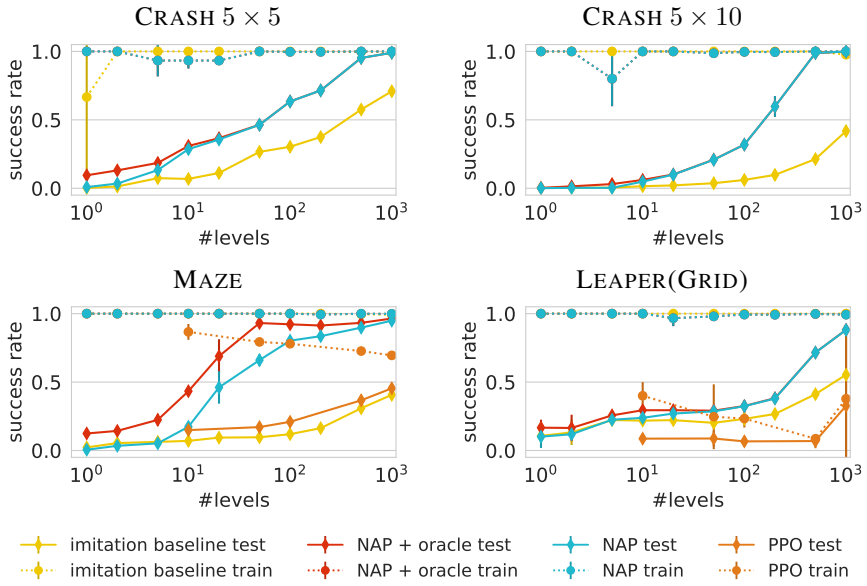


Figure 3: Generalization performance depending on the number of seen levels during training. Dashed lines indicate performance on training-levels and solid lines show the generalization to unseen levels. NAP shows good generalization already after 100 levels in most cases.

5.1 Results

We train our method (NAP) and the imitation learning baseline until saturation on the training set, resulting in virtually 100% success rate when evaluating on train configurations in the environment. For the PPO baseline we use the code from Cobbe et al. (2019) and provide also two subsequent frames and 200M time steps for training. For our method we also provide a version with access to the true start and end-point prediction (NAP+ oracle)

In Fig. 3 we show the performance of the methods when exposed to different number of levels at training time. As reported in Cobbe et al. (2019), the baselines have a large generalization gap, and also poor performance when $< 10\,000$ levels are seen. We find that NAP shows strong generalization performance, already for < 500 levels. In some environments, such as the MAZE we obtain near 80% success rate already with just 100 levels which is reached by PPO after seeing $200\,000$ levels. For the CRASH JEWEL HUNT 5×5 already with 30 trajectories a third of the 1000 test-levels can be solved, the baseline manages less than 50 out of the 1000. In the CHASER env, see Sec. G, subgoals need to be predicted, which is illustrated in Fig. 6. We show strong generalization with very few training levels – reward of ~ 6 on unseen test-levels when using 100 training levels.

6 Discussion

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 in the low-data regime*. These policies can be trained end-to-end by employing blackbox differentiation theory Vlastelica et al. (2020).

Although there is a clear benefit in using NAP, the methods comes with certain caveats. We assume that the topological structure of the latent planning graph is known a priori, relaxing this assumption we leave for future work. Furthermore, we assume that the structure of the latent graph is fixed and not dynamically changing over time, i.e. that each available action at a vertex corresponds to the same edge.

7 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

Brandon Amos and Denis Yarats. The differentiable cross-entropy method. In *International Conference on Machine Learning, ICML*, 2020.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. *arXiv preprint arXiv:2002.08676*, 2020.

Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. *arXiv preprint arXiv:2004.08763*, 2020.

Sebastian Blaes, Marin Vlastelica Pogančić, Jiajie Zhu, and Georg Martius. Control what you can: Intrinsically motivated task-planning agent. In *Advances in Neural Information Processing Systems 32*, pp. 12541–12552. Curran Associates, Inc., 2019.

Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2020.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint:1912.01588*, 2019.

Emir Demirovic, Peter J. Stuckey, James Bailey, Jeffrey Chan, Christopher Leckie, Kotagiri Ramamohanarao, and Tias Guns. Predict+optimise with ranking objectives: Exhaustively learning linear functions. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 1078–1085. ijcai.org, 2019. doi: 10.24963/ijcai.2019/151. URL https://doi.org/10.24963/ijcai.2019/151.

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

Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pp. 1013–1023, 2017.

Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *ArXiv*, abs/1710.08005, 2017.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. 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.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *AAAI*, pp. 1504–1511, 2020.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 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.

Jaynta Mandi, Emir Demirovic, Peter J. Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. *CoRR*, abs/1911.10092, 2019. URL http://arxiv.org/abs/1911.10092.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. Sparsemap: Differentiable sparse structured inference. *arXiv preprint arXiv:1802.04223*, 2018.

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

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

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. *arXiv preprint arXiv:2003.11657*, 2020.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, volume 37 of *ICML*, pp. 1312–1320, 2015.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, volume 70 of *ICML'17*, pp. 3191–3199. PMLR, 2017.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 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.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, December 2005. ISSN 1532-4435.

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

Po-Wei Wang, Priya L Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv preprint arXiv:1905.12149*, 2019.

A Related Work

Planning in reinforcement learning Differentiable planning has been proposed in previous works, e.g. in the continuous case with CEM Amos & Yarats (2020); 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 for learning a representation. Alternatively to learning the representation, a planning graph can be constructed from the agents' experience using a learned value function 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.

Discrete algorithms in end-to-end trainable networks Incorporating expert discrete solvers into end-to-end trainable architectures is a topic with both rich history and exciting recent developments. The simpler setup, in which the output of the non-differential discrete algorithm is directly compared to ground truth values has been first addressed by max-margin techniques (Tsochantaridis et al., 2005). Since then, various adaptations have been proposed such as the "predict-and-optimize" framework and its variants (Elmachtoub & Grigas, 2017; Demirovic et al., 2019; Mandi et al., 2019). Also, specializations for cases of concrete types of discrete optimization problems such as sparse structured inference (Niculae et al., 2018), logical satisfiability (Wang et al., 2019), submodular optimization (Djolonga & Krause, 2017) or mixed integer programming (Ferber et al., 2020).

The harder task of purely providing a gradient and therefore allowing *entirely hybrid* architectures containing a discrete algorithm as an intermediate layer has only been addressed recently. First, by (Vlastelica et al., 2020) who introduce an efficient implicit piece-wise linear interpolation scheme of the true underlying piece-wise constant function. On the other hand, Berthet et al. (2020) offer a Monte Carlo technique for estimating the Jacobian of a Gaussian smoothing of the piecewise constant function at hand.

B 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, see Fig. 4. The *exact* gradient of this linear interpolation is computed efficiently via evaluating the solver on only one more instance (see Algorithm 1) in appendix.

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

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

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

C Goal Prediction, Static and Dynamic

In order to apply the solver to the learned latent graph representation, we need to map the current state of the environment to appropriate start and end vertices (s, e). To this end, we employ a second ResNet18 – the *goal-predictor* – similar to the *cost-predictor* that learns to extract the agent start position and a suitable target position. The training of this predictor is using a Cross-Entropy loss and is independent of learning the costs of the latent graph representation.

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 $\frac{1}{\lambda}\left(Y_\lambda - Y\right)$

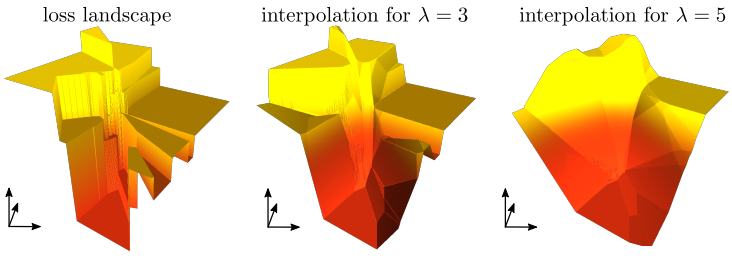


Figure 4: Differentiation of a piecewise constant loss resulting from incorporating a combinatorial solver. A two-dimensional section of the loss landscape is shown (left) along with two differentiable interpolations of increasing strengths (middle and right).

At training time, given the expert trajectories Y^* we have access to the current position of the agent and its position in the future. Thus, for predicting s we have a simple supervision signal, namely the current position. For the goal prediction e we extract a set of suitable goal locations from the expert. Here, we distinguish between *global* and *local* planning.

In the *global* setting, the last position of the expert is the goal e, corresponding to, for instance, the jewel in CRASH JEWEL HUNT, see Fig. 2.

In the *local* setting, we expect the end vertex to be an intermediate goal ("collect an orb"), which effectively allows for high-level planning strategies while the low-level planning is delegated to the discrete solver. In this case, the positively labeled supervision at time t are all locations of the (expert) agent between step $t + T$ and $t + 2T$.

The local setting allows to limit the complexity of our method, which grows with the planning horizon. It also is a trade-off between the combinatorial complexity solved by the TDSP and the goal predictor. Ideally, the planning horizon T used for the cost-prediction is long enough to capture the combinatorial intricacies of the problem at hand, such as creating detours towards the goal in the case of future dangerous states, or avoiding dead-ends in a maze.

This formulation makes our architecture a hierarchical method similar to Blaes et al. (2019); Nachum et al. (2018), and allows for solving tasks that are not typical goal-reaching problems, such as the CHASER environment, see below.

D Sensitivity to the Planning Horizon

We provide a sensitivity analysis of the performance with different planning horizons. Our results indicate that longer horizons benefit environments with increased dynamical interactions. As apparent from Fig. 5, our method outperforms the imitation baseline greatly in the crash environment, the gap between the methods being correlated with the complexity of the environment (5×5 vs 5×10). It can be seen also that making the planning horizon smaller in these environments hurts performance.

On the other hand, for environments with no dynamics, such as the maze environment, there is no benefit in using time-dependent costs, as expected. Nevertheless, there is still strong performance gain in generalization when using NAP oppose to vanilla imitation learning from expert trajectories.

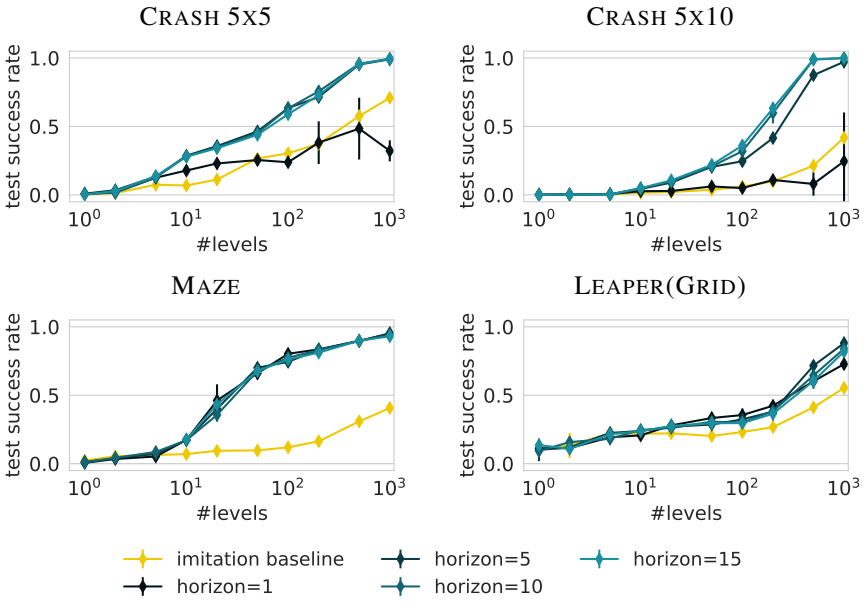


Figure 5: Test success rate of our method with different horizon lengths.

E 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 gcMDP, 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:

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

During evaluation, the cost margin is removed from the shortest path calculation.

F Environments

F.1 CRASH JEWEL HUNT

We first consider an environment we constructed to test NAP, called CRASH JEWEL HUNT which can be seen in Fig. 2. 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 cyclic boundary conditions) with velocities that vary across levels but not within a level, see Fig. 2(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. More concretely, we vary the velocities, sizes and initial positions of the boxes as well as the initial and goal positions.

F.2 ProcGen Benchmark

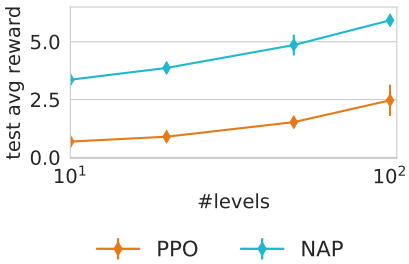
In addition to the jewel hunt environment, we evaluate our method on the MAZE, LEAPER and CHASER environments from the ProcGen suite Cobbe et al. (2019). We have chosen these environments because their structure adheres to our assumptions. For the LEAPER we modified the environment such that a grid-world dynamics applies (LEAPER(GRID)). Based on performance of the baselines, the resulting (LEAPER(GRID)) is not an easier environment.

The MAZE and the LEAPER(GRID) tasks have a static goal whose position only varies across levels, whereas the CHASER requires collection of all orbs without contact with the spiders, so the local goals need to be inferred on the fly. The CHASER environment is also **particularly challenging** as even the expert episodes require on average 150 steps, each of which carries a risk of dying. For this reason, we used three expert trajectories per level.

G Results on CHASER Environment

Here we provide additional results of NAP in comparison to PPO on the CHASER environment. This environment is specific in that there are not explicit goals needed to be reached, but the agent needs to pick up all of the green orbs while avoiding the enemies. This poses a challenge, since we are reduced to a very short-horizon in comparison to the episode length of the expert. Furthermore, the environment doesn't have independent dynamics but the dynamics depend on the movement of the agent. Experiments show that NAP still outperforms PPO considerably in the low data regime (Fig 6)

(a) Performance on CHASER



(b) Short-horizon plans

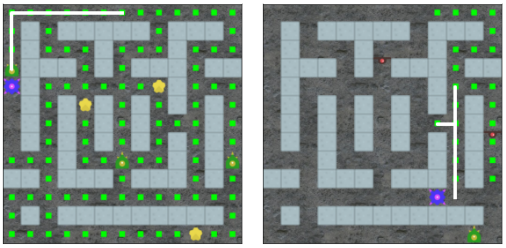


Figure 6: Performance of NAP against PPO on the CHASER environment (a) trained on 10, 20, 50, and 100 levels. In (b) we show the short-horizon plans (white) of the agent (blue) at step 5 and 110 in the environment.