Neuro-algorithmic Policies Enable Fast Combinatorial Generalization

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

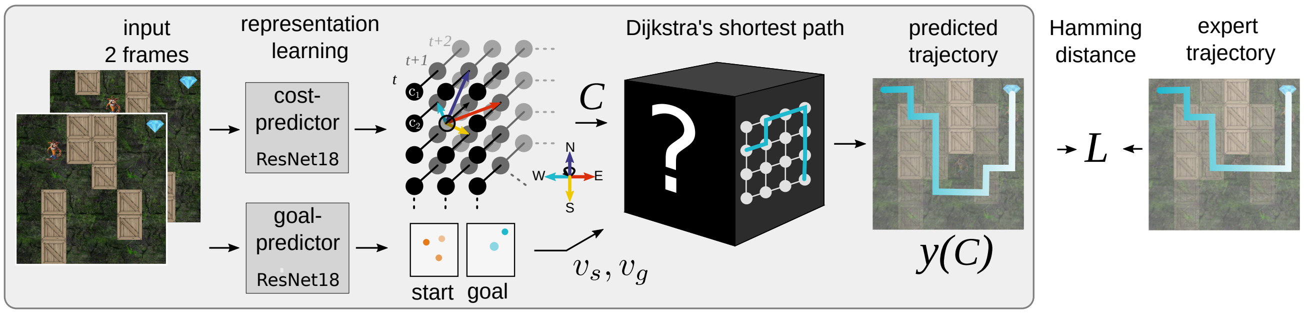


Figure 1: Architecture of the neuro-algorithmic policy. Two subsequent frames are processed by two simplified ResNet18s: the cost-predictor outputs a tensor (width × height × time) of vertex costs $C^t(v)$ and the goal-predictor outputs heatmaps for start and goal. The time-dependent shortest path solver finds the shortest path to the goal. Hamming distance between the proposed and expert trajectory is used as loss for training. Videos and Code are available at martius-lab.github.io/NAP.

Abstract

Although model-based and model-free approaches to learning the control of systems have achieved impressive results on standard benchmarks, generalization to task variations is still lacking. Recent results suggest that generalization for standard architectures improves only after obtaining exhaustive amounts of data. We give evidence that generalization capabilities are in many cases bottlenecked by the inability to generalize on the combinatorial aspects of the problem. We show that, for a certain subclass of the MDP framework, this can be alleviated by a neuro-algorithmic policy architecture that embeds a time-dependent shortest path solver in a deep neural network. Trained end-to-end via blackbox-differentiation, this method leads to considerable improvement in generalization capabilities in the low-data regime.

[1]Max Planck Institute for Intelligent Systems, Tübingen, Germany. Correspondence to: Marin Vlastelica <mvlastelica@tue.mpg.de>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

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. On the other end lie approaches that require interaction with the environment to obtain data for policy extraction, posing the problem of exploration. Most Reinforcement Learning (RL) algorithms fall into the latter category. In this work, we concern ourselves primarily with the setting where limited expert data is available, and a policy needs to be extracted by imitation learning.

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 methods require exhaustive amounts of exposure to environmental variability before starting to generalize (Cobbe et al., 2020).

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 or to improve value-learning. This has been argued to be the best strategy in the presence of an accurate model and sufficient computation time (Daw et al., 2005). Furthermore, one can use the transition model alongside a reward model to generate offline data to improve value function learning (Sutton, 1991; Janner et al., 2019). However, learning a precise transition model is often harder than learning a policy. The transition model often has a much larger dimensionality than the policy since it needs to model aspects of the environmental dynamics that are perhaps irrelevant for the task. This is particularly true for learning in problems with high-dimensional inputs, such as raw images. 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., 2020).

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 embedded planners end-to-end. More specifically, we use a time-dependent shortest path (TDSP) planner acting on a temporally evolving graph generated by a deep network from the inputs. By learning the time-evolving costs of the graph, our method builds a specific model of the system that is sufficient for planning. This choice is akin to goal-conditioned MDPs since the shortest path algorithm expects a goal to be reached. To tackle more general settings, we predict subgoals as part of the algorithm. We demonstrate the effectiveness of this approach in an offline 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 needs orders of magnitude fewer samples 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 the potential to endow artificial agents with the main component of intelligence, the ability to reason.

Our contributions can be summarized as follows: **(i)** We show that combinatorial inductive biases implemented through neuro-algorithmic policies can be used to tackle the generalization problem in reinforcement learning. **(ii)** We show that architectures embedding TDSP solvers are applicable beyond goal-reaching environments. **(iii)** We demonstrate learning neuro-algorithmic policies in dynamic game environments from images.

2. Related Work

Planning There exist multiple lines of work aiming to improve classical planning algorithms such as improving sampling strategies of Rapidly-exploring Random Trees (Gammell et al., 2014; Burget et al., 2016; Kuo et al., 2018). Similarly, along this direction, Kumar et al. (2019) propose a conditional VAE architecture for sampling candidate waypoints. Orthogonal to this are approaches that learn representations such that planning is applicable in the latent space. Hafner et al. (2019) employ a latent multistep transition model. Savinov et al. (2018) propose a semi-parametric method for mapping observations to graph nodes and then applying a shortest path algorithm. Asai & Fukunaga (2017); Asai & Kajino (2019) use an autoencoder architecture to learn a discrete transition model suitable for classical planning algorithms. Li et al. (2020) learn compositional Koopman operators with graph neural networks mapping to a linear dynamics latent space, which allows for fast planning. Chen et al. (2018); Amos et al. (2017) perform efficient planning by using a convex model formulation and convex optimization. Alternatively, the replay buffer can be used as a nonparametric model to select waypoints (Eysenbach et al., 2019) or in an MPC fashion (Blundell et al., 2016). None of these methods perform differentiation through the planning algorithm to learn better latent representations.

Differentiation through planning Embedding differentiable planners has been proposed in previous works, e.g. in the continuous case with CEM (Amos & Yarats, 2020; Bharadhwaj et al., 2020). Wu et al. (2020) use a (differentiable) recurrent neural network as a planner. Tamar et al. (2016) use a differentiable approximation of the value iteration algorithm to embed it in a neural network. Silver et al. (2017) differentiate through a few steps of value prediction in a learned MDP to match the externally observed rewards. Srinivas et al. (2018) use a differentiable forward dynamics model in latent space. Karkus et al. (2019) suggest a neural network architecture embedding MDP and POMDP solvers and during the backward pass, they substitute the algorithms by learned approximations. In comparison, we do not perform any relaxation or approximation of the planner itself and we learn interpretable time-dependent costs of the latent planning graph based on expert demonstrations by differentiating through the planner. Similarly to our work, Yonetani et al. (2020) embed an A* algorithm into a neural network, but in comparison, their method does not operate with time-dependent costs, subgoal selection and does not provide a policy for closed-loop control.

Inverse reinforcement learning and imitation learning Uncovering the expert's objective function from demonstrations has been a central topic in reinforcement learning (Ng & Russell, 2000). Our method is connected to inverse reinforcement learning in the sense that we learn the objective function that the expert optimizes to extract an optimal policy, also called apprenticeship learning (Abbeel & Ng, 2004; Neu & Szepesvári, 2012; Aghasadeghi & Bretl, 2011). What separates our approach is that the inferred costs are inherently part of the learned neuro-algorithmic policy in conjunction with the applied planner on the costs.

Our method is an offline imitation learning method, but since we propose an end-to-end trainable policy, it is naturally extendable to the online case with a method such as

DAgger (Ross et al., 2011) or other online reinforcement learning methods augmented with expert datasets (Reddy et al., 2019; Ho & Ermon, 2016).

Offline model-based reinforcement learning Model-based methods have shown promise by facilitating better generalization (Janner et al., 2019). These approaches fall into two camps: using models to extract a policy in a Dyna-style approach (Sutton, 1991; Janner et al., 2019; Sutton et al., 2008; Yao et al., 2009; Kaiser et al., 2020), or incorporating the model in a planning loop, i.e. model-predictive control (Finn & Levine, 2017; Racanière et al., 2017; Oh et al., 2017; Silver et al., 2017; Pinneri et al., 2021). In this work, we consider the latter case where an implicit transition model is "hidden" within the predicted time-dependent costs.

Combinatorial algorithms in end-to-end trainable networks We suggest a hybrid policy consisting of a neural network and an accompanying expert (shortest path) discrete solver that is trainable end-to-end. Incorporating expert discrete solvers into end-to-end trainable architectures is a topic with exciting recent developments. For the simpler setup of comparing to ground-truth values on the solver output, numerous frameworks have been suggested such as the "predict-and-optimize" framework and its variants (Elmachtoub & Grigas, 2021; Demirovic et al., 2019; Mandi et al., 2020). Moreover, specializations for concrete cases 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) have been proposed.

We are interested in the harder case of providing an *entirely hybrid* architecture which may use the solver at intermediate levels and is trainable end-to-end. For this case, two approaches have recently emerged (Vlastelica et al., 2020; Berthet et al., 2020). Vlastelica et al. (2020) introduce an efficient implicit piecewise linear interpolation scheme, while Berthet et al. (2020) introduce an estimation of a Gaussian smoothing of the piecewise constant function. The approach from Vlastelica et al. (2020) is especially appealing, since it allows for uses in which the solver is the computational bottleneck. By formulating the control problem as a time-dependent shortest path problem (TDSP), we show that the framework of Vlastelica et al. (2020) is applicable in specific control settings.

3. Markov Decision Processes and Shortest Paths

We consider the problem formulation in the context of a Markov Decision Process (MDP). We concern ourselves with policies that solve a time-dependent shortest path problem on a latent graph representation related to that MDP. We start with a class of MDPs that can be directly mapped to a shortest path problem and construct this mapping. Then we consider conditions that allow for a reduced latent graph where the optimal policy follows a time-dependent shortest path problem.

First, we consider discrete MDPs with deterministic transitions. In addition, we follow 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 $s \in \mathcal{S} \to s' \in \mathcal{S}$ when taking the action $a \in \mathcal{A}$, $g \in \mathcal{S}$ 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.*

In episodic reinforcement learning, the objective is to find a policy that maximizes 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, which is also the terminal state.

Definition 2 *A discrete and deterministic goal-conditioned Markov Decision Process (ddgcMDP) $\ddot{\mathcal{M}}$ is a gcMDP with discrete and finite state space $\mathcal{S}$ and action space $\mathcal{A}$ with deterministic transitions, i.e. $p(s' \mid a, s)$ is one-hot.*

Let us consider the following graph representation (G, v_s, v_g): a weighted graph $G = (V, E, C)$ together with start vertex v_s and goal vertex v_g, where V is the vertex set, $E \subset (V \times V)$ is the edge set, and $C \in \mathbb{R}_+^{|E|}$ is the cost matrix with positive entries. We write $C(e)$ for the cost of edge e. In addition, we consider an inverse model $\psi \colon E \to \mathcal{A}$, associating an edge to an action.

A direct translation of a ddgcMDP to a graph is given by a bijective correspondence $\phi \colon \mathcal{S} \to V$ between states and vertices, for each possible transition (s, a, s') there is an edge $e = (\phi(s), \phi(s'))$ with $\psi(e) = a$, and the goal vertex is $v_g = \phi(g)$. The cost matrix C takes the role of the reward function with $C(\phi(s), \phi(s')) = c_{\max} - r(s, a, s', g)$ where $c_{\max}$ is an upper bound on the reward ensuring positive cost values. To deal with variable or infinite episode lengths, g can be seen as an absorbing state where $r(g, \cdot, g, g) = c_{\max}$, incurring a cost of 0 at the goal. Due to the deterministic nature of the ddgcMDP, the optimal policy yields a path with maximal return (sum of rewards) from start s_0 to goal g, which now coincides with the shortest path according to C by definition.

3.1. Factorized MDPs and Time-dependent Shortest Path

In many interesting environments, the state space is exponentially large, e.g. due to independent dynamics of entities. As an illustrative example, consider the game environment shown in Fig. 1 and 3 – an agent with moving obstacles. For such environments, the corresponding graph would become intractably large. We now define conditions under which the size of the graph can be drastically reduced, without losing the property that its shortest path solves the MDP. To this end, we assume the state space can be factorized as $\mathcal{S} = \mathcal{S}_A \times \mathcal{S}_E$, where $\mathcal{S}_A$ is affected only by the actions and $\mathcal{S}_E$ by the environment dynamics independent of the actions. We write the decomposition of each state as $s = (s^a, s^e)$. The mapping from state space to graph vertices $\phi\colon \mathcal{S} \to V$ is bijective only w.r.t. $\mathcal{S}_A$ and ignores $\mathcal{S}_E$, i.e. $\forall s^a \in \mathcal{S}_A\ \exists v \in V\colon \phi(s^a, s^e) = v\,, \forall s^e \in \mathcal{S}_E$. For brevity, we write $\phi(s^a)$. For instance, using $\mathcal{S}_A$ as the agent's positions on a discrete grid results in just as many vertices as there are positions of the agent.

Next, we show how a solution to this factorized ddgcMDP can be found by solving a time-dependent shortest path (TDSP) problem on the reduced graph. The cost matrix C is now a time-dependent quantity, i.e. $C \in \mathbb{R}_+^{H\times|E|}$ that assigns every edge e a positive cost value for each time $t \in \{1, 2, \dots, H\}$ of the planning horizon/episode. To ease the notation, we write $C^t(e)$ for the cost of edge e at time t. The TDSP problem is defined as reaching the goal vertex v_g within at most H steps when starting at time step 1 in the start vertex v_s.

Two situations need to be modeled by the cost:[1] a) the environment dynamics can make an edge e become unavailable (for instance, an obstacle is moving in the way), then the cost should be infinite: $C^t(e) = \infty$, and b) the environment dynamics changes the reward, thus for all other edges we have

$$C^t(\phi(s^a), \phi(s'^a)) = c_{\max} - r((s^a, s^e_t), a, (s'^a, s^e_t)) \quad (1)$$

where $a = \psi(\phi(s^a), \phi(s'^a))$. Again, with this construction, the time-dependent shortest path solution coincides with the optimal policy of the specific ddgcMDP. We can also deal with stochastic environment dynamics as long as the action's effect on the state stays deterministic. This changes the reward term in Eq. 1 to an expectation over environment dynamics:

$$\mathop{\mathbb{E}}_{s^e_t}[r((s^a, s^e_t), a, (s'^a, s^e_t))].$$

In our architecture, described below, the policy generates a latent graph at every observation/state and solves a ddgcMDP to optimality at every time step following a model predictive control approach using receding horizon planning.

[1]Note that learning the *exact* costs is not necessary, since we are only interested in optimal trajectories.

4. Shortest Path Algorithm and its Differentiation

We aim to predict the cost matrix C via a deep neural network and train the entire system end-to-end via gradient descent on expert trajectories, as illustrated in Fig. 1. 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).

Time-dependent shortest path with vertex costs. In our neuro-algorithmic policy, we use the TIME-DEPENDENT-SHORTEST-PATH (TDSP) formulation based on vertex-costs instead of edge-costs, due to the reduction in cost matrix size by a factor of $|\mathcal{A}|$. The TDSP solver has as input the graph $G(V, E, C)$ with time-dependent cost matrix $C \in \mathbb{R}_+^{H\times|V|}$ and a pair of vertices $v_s, v_g \in V$ (start and goal). We write $C^t(v)$ for the cost of vertex v at time t.

This version of the shortest path problem can be solved by executing the Dijkstra's shortest path algorithm[2] (Dijkstra, 1959) on an augmented graph. In particular, we set

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

where the cost of vertex $(v_i, t) \in V^*$ is simply $C^t(i)$ and $E^{\circlearrowleft}$ 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 the vertex (v_g, H) from $(v_s, 1)$ with the minimum traversal cost.

4.1. 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, see Fig. 2. The *exact* gradient of this linear interpolation is computed efficiently via evaluating the solver on only one additional instance (see Algo. 1).

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 $G = (V, E, C)$ as described above, we define $Y \in \{0, 1\}^{H\times|V|}$ an indicator matrix of visited vertices. In

[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.

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

```
function ForwardPass($C$, $v_s$, $v_g$)
    $Y$ := TDSP($C, v_s, v_s$)            // Run Dijkstra's algo.
    save $Y$, $C$, $v_s$, $v_e$          // Needed for backward pass
    return $Y$

function BackwardPass($\nabla L(Y)$, $\lambda$)
    load $Y$, $C$, $v_s$, $v_e$
    $C_\lambda := C + \lambda \nabla L(Y)$      // Calculate modified costs
    $Y_\lambda$ := TDSP($C_\lambda, v_s, v_g$)    // Run Dijkstra's algo.
    return $\frac{1}{\lambda}(Y_\lambda - Y)$
```

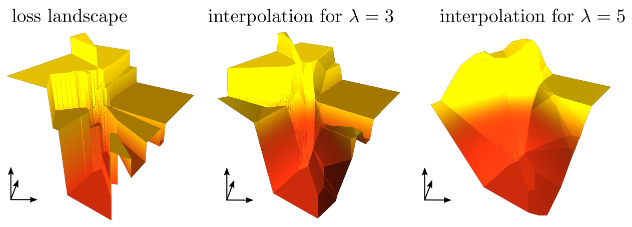

loss landscape
interpolation for $\lambda = 3$
interpolation for $\lambda = 5$

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).

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^*) from start to goal will be denoted as $\mathcal{Y} := \mathrm{Adm}(G, v_s, v_g)$. The time-dependent shortest path optimization problem can be then rewritten as

$$\mathrm{TDSP}(C, v_s, v_g) = \arg\min_{Y \in \mathcal{Y}} \sum_{(i,t)} Y_i^t C_i^t, \tag{2}$$

where $\mathcal{Y} = \mathrm{Adm}(G, v_s, v_g)$. This is an inner-product objective and thus the theory from (Vlastelica et al., 2020) applies and allows us to learn the cost-matrix generating network with gradient descent, as described below.

4.2. Cost Margin

The supervision signal for the costs C of the latent graph is only indirectly given via the shortest path solutions. At training time, there is no incentive to make these costs robust to small misestimations. Thus, inducing a margin on costs can be beneficial for generalization. Similarly to Rolínek et al. (2020), where it was used in the context of rank-based metric learning, we induce a margin α by increasing the cost of the ground-truth path and decreasing the rest in the training stage of the algorithm:

$$c_i^t \leftarrow \begin{cases} c_i^t + \frac{\alpha}{2} & \text{if } (v_i, t) \in Y^* \\ c_i^t - \frac{\alpha}{2} & \text{if } (v_i, t) \notin Y^* \end{cases} \quad \forall\, (v_i, t) \in V^*. \tag{3}$$

5. Neuro-algorithmic Policy Framework

We propose the Neuro-algorithmic Policy (NAP) framework, which is an end-to-end trainable deep policy architecture embedding an algorithmic component using the afore-mentioned techniques. Following the definitions from Sec. 3, we concern ourselves with learning the mapping $\varphi_\theta : \mathcal{S} \mapsto \mathbb{R}^{|V| \times T} \times V \times V$, i.e. mapping from MDP states to cost matrices and respective start and end vertices for the TDSP problem[3] of planning horizon H. This enables us to construct the policy

$$\pi_\theta := \psi \circ \mathrm{TDSP} \circ \varphi_\theta. \tag{4}$$

The mapping φ_θ can be decomposed into φ_θ^c (*cost-predictor*), φ_θ^s (*start-predictor*), φ_θ^g (*goal-predictor*), i.e. mappings from state to costs, start vertex and goal vertex. In practice, instead of learning φ_θ^s and φ_θ^g directly, we learn the conditional probability densities $p_\theta^s(v|s)$ and $p_\theta^g(v|s)$.

In this work, we examine the application of neuro-algorithmic policies to the imitation learning setting, where we have access to trajectories τ sampled from the expert policy distribution $\eta(\pi^*)$. Given a fixed planning horizon H, the objective that we consider consists of three main parts, the latent cost term, start vertex term and goal vertex term. The latent cost objective is defined as

$$J^C(\theta, H) = \mathop{\mathbb{E}}_{\tau_{\mathrm{t:t+H}} \sim \eta(\pi^*)} \Big[\mathrm{d^H}(\mathrm{TDSP}(\varphi_\theta(\tau_t)), \phi'(\tau)) \Big], \tag{5}$$

where $\mathrm{d^H}(\cdot, \cdot)$ denotes the Hamming distance between predicted and expert paths in the latent graph, and $\phi' : \mathcal{S} \mapsto V$ the mapping of the expert-visited states to latent graph vertices. The second part of the objective is a standard cross-entropy term for the start and goal vertices that allows us to train a *start-* and *goal-predictor*:

$$\begin{aligned} J^P(\theta, H) = \mathop{\mathbb{E}}_{\tau_{\mathrm{t:t+H}} \sim \eta(\pi^*)} \Big[& -\log p_\theta^s(\phi'(\tau_t)|\tau_t) \\ & - \log p_\theta^g(\phi'(\tau_{t+H})|\tau_t) \Big]. \end{aligned} \tag{6}$$

We assume access to ϕ' at training time in order to map the expert to the latent graph for calculation of J^C and J^P. Finally, we optimize for the sum of J^C and J^P.

We utilize a concrete architecture consisting of two main components: a backbone ResNet18 architecture (without the final fully connected layers (a detailed description is available in Sec. C in the supplementary) 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 tensor C for the planning horizon H with the *cost-predictor*, the start vertex v_s and goal vertex v_g with the *goal-predictor*, explained below.

[3]We hold the graph structure fixed, namely the set of edges E and learn the costs C. Therefore we replace G with costs C in the TDSP solver to simplify the notation.

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

5.1. Global and Local Goal Prediction

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 goal vertices (v_s, v_g). To this end, we employ a second ResNet18 similar to the *cost-predictor* that approximates $p^s(v|s)$ and $p^g(v|s)$, i.e. the start and goal conditional densities.

At training time, given the expert trajectories, we have access to the mapping ϕ' that maps the expert trajectory to the latent graph. In the *global* setting, the last position of the expert is the goal v_g, corresponding to, for instance, the jewel in CRASH JEWEL HUNT which is also the terminal state, see Fig. 3.

In the *local* setting, we expect the end vertex to be an intermediate goal (for instance "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 $t + H$ and $t + 2H$.

The local setting allows to limit the complexity of our method, which grows with the planning horizon. This is also a trade-off between the combinatorial complexity solved by the $TDSP$ solver and the goal predictor. Ideally, the planning horizon H 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 akin to hierarchical methods 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.

6. Experiments

To validate our hypothesis that embedding planners into neural network architectures leads to better generalization in control problems, we consider several procedurally generated environments (from the ProcGen suite (Cobbe et al., 2020) and CRASH JEWEL HUNT) with considerable variation between *levels*.

We compare with the following baselines: a standard behavior cloning (BC) baseline using a ResNet18 architecture trained with a cross-entropy classification loss on the same dataset as our method; the PPO algorithm as implemented in Cobbe et al. (2020) and data-regularized actor-critic (DrAC) (Raileanu et al., 2020). A comparison to DrAC is especially

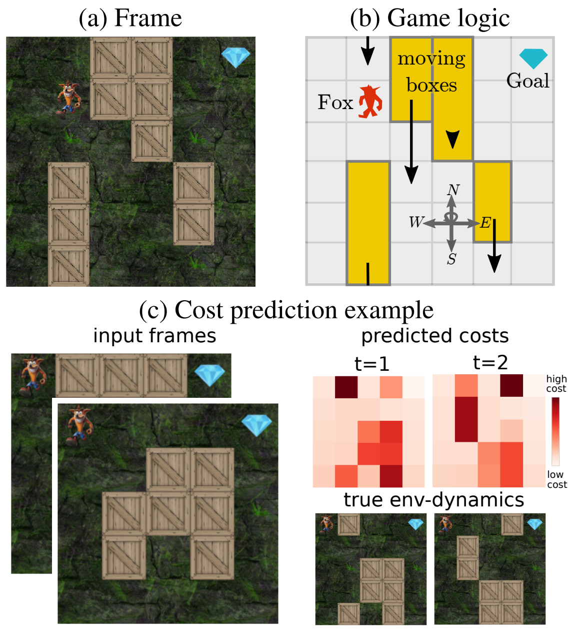


Figure 3: The CRASH JEWEL HUNT environment. The goal for the fox, see (a), is to obtain the jewel in the right most column, while avoiding the moving wooden boxes (arrows in (b)). When the agent collides with a wooden box it instantly fails to solve the task. We observe that the predictions of the costs in (c) are highly interpretable, corresponding to (future) movements of the boxes.

interesting, since the method claims to improve generalization by applying optimized data augmentations. As there are multiple variations of data augmentation suggested by Raileanu et al. (2020), we run all of them and select the best result as DrAC* for performance comparison to our method.[4] We also ablate the *start-* and *goal-predictor* and replace with the ground truth vertices, this serves as an upper baseline for NAP which we denote with NAP*. More details on the training procedure and the hyperparameters can be found in the supplementary Sec. D.

For the experimental validation, we aim to anwser the following questions: **(i)** Can NAP be trained to perform well in procedurally generated environments? **(ii)** Can NAP generalize in a low data regime, i.e. after seeing only few different levels? **(iii)** Can we also solve non-goal-reaching environments?

6.1. CRASH JEWEL HUNT

For proof of concept, we first consider an environment we constructed to test NAP, called CRASH JEWEL HUNT which can be seen in Fig. 3. The environment corresponds to a

[4]We defer a more detailed description of DrAC along with performance plots to Sec. F

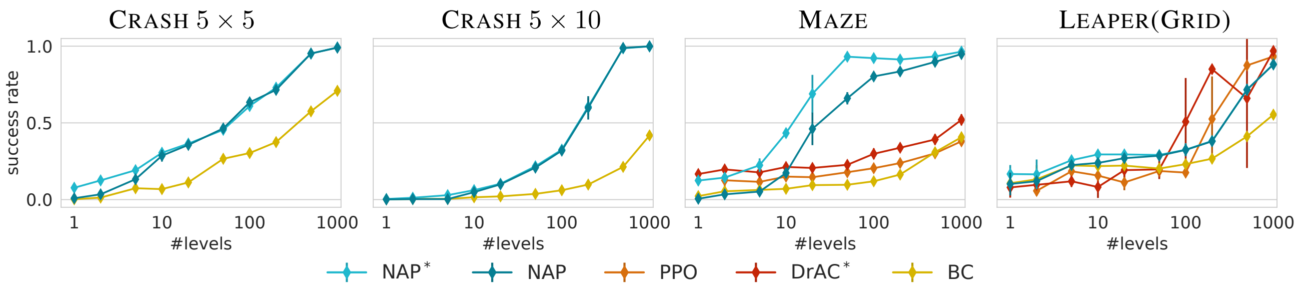


Figure 4: Performance on unseen test levels for a different number of training levels. We observe that NAP already shows signs of generalization after only being trained on 100 levels.

grid-world of dimensions *height* $\times$ *width* 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. 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. More concretely, we vary the velocities, sizes and initial positions of the boxes as well as the start and goal positions.

6.2. ProcGen Benchmark

In addition to the jewel hunt environment, we evaluate our method on the hard version MAZE, LEAPER and CHASER environments from the ProcGen suite (Cobbe et al., 2020). We have chosen these environments because their structure adheres to our assumptions. For LEAPER, we modified the environment such that grid-world dynamics applies and denote it as LEAPER(GRID).

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 contacting 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, most of which carry the risk of dying. For this reason, we used three human expert trajectories per level.

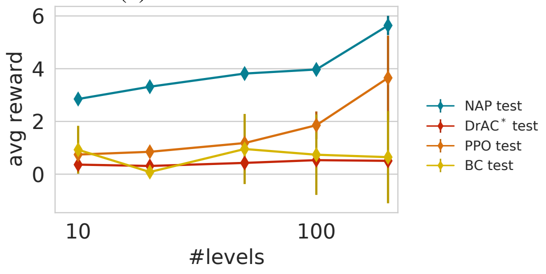
6.3. Results

We train our method (NAP) and the imitation learning baseline until saturation on a 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., 2020) and provide also two subsequent frames and 200M time steps for training. For the DrAC baselines we use the code from Raileanu et al. (2020). For our method, we also report performance of a version with access to the true start and end-point prediction (NAP*), with the exception of the CHASER where true goals are not well-defined.

In Fig. 4, we show the performance of the methods when exposed to a different number of levels at training time. As reported in Cobbe et al. (2020), the baselines have a large generalization gap and 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 MAZE we obtain near 80% success rate already with just 100 levels which is never reached by PPO or DrAC even

(a) Performance on CHASER



(b) Short-horizon plans

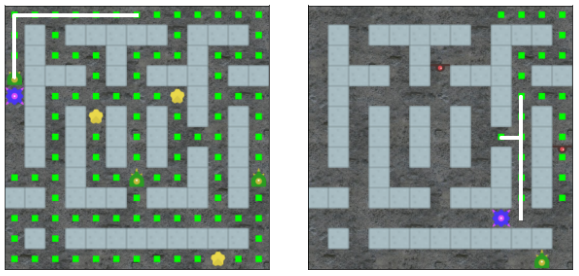


Figure 5: Performance of NAP against PPO on the CHASER environment (a) trained on 10, 20, 50, 100 and 200 levels. In (b) we show the short-horizon plans (white) of the agent (blue) at step 5 and 110 in the environment.

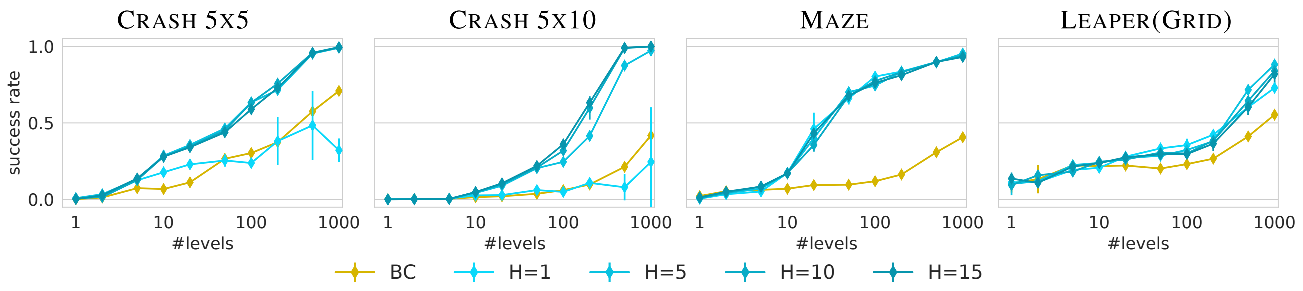


Figure 6: Test success rate of our method with different horizon lengths. The solver assumes that the last horizon step costs remain to infinity. In this sense, the horizon of length 1 corresponds to a static solver.

after seeing a 1000 levels. Our NAP trained on 1000 levels reaches the same performance as PPO trained on 200 000 levels of the MAZE environment. For CRASH JEWEL HUNT 5×5, already with 30 trajectories a third of the 1000 test-levels can be solved, while the behavior cloning baseline manages less than 50 out of the 1000.

In CHASER we learn a subgoal predictor from expert trajectories that predicts a subgoal for the TDSP algorithm at each time step, since there is no clear goal state specified by the environment. The performance plots in Fig. 5 show that NAP outperforms the baselines. Additionally, in Fig. 5(b) we observe that using NAP the agent is able to perform detours to collect orbs, indicating that the learned costs in the latent planning graph reflect the actual task.

6.4. 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. 6, our method outperforms the behavior cloning baseline in all of the environments, the gap between the methods being correlated with the complexity of the environment. It can also be seen that making the planning horizon smaller in environments with dynamics hurts performance.

On the other hand, for environments with no dynamics, such as the maze environment, there is no benefit in using a longer planning horizon, as expected. Nevertheless, there is still strong performance gain in generalization when using NAP as opposed to vanilla imitation learning from expert trajectories thanks to the introduced combinatorial inductive bias.

6.5. Path Optimality

We have observed that introducing combinatorial structural biases with a method such as NAP can benefit generalization greatly. Another interesting question is how optimal are the paths taken by NAP. Even though NAP has an advantage in having access to expert optimal or near-optimal trajectories, this does not necessarily translate to being able to act optimally in unseen situations.

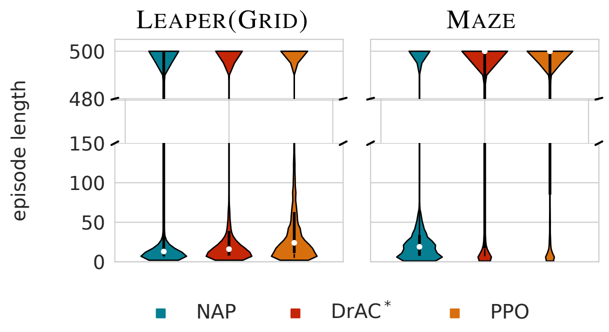


Figure 7: Distributions of test level episode lengths after training on 500 levels for the LEAPER(GRID) and MAZE environments over 3 seeds. We observe that NAP tends to take shorter paths than DrAC* and PPO. We also observe that the median episode length for the baselines in the MAZE task is located in the upper part, which corresponds to unsolved levels.

For this analysis we have plotted the episode length distribution across runs for NAP, DrAC* and PPO, which can be seen in Fig. 7. This shows us that NAP generalizes with more optimal (shorter) paths in comparison to DrAC* and PPO, even when compared only to levels solved by all methods. Even in the LEAPER(GRID) environment, where DrAC* outperforms NAP by measure of success rate, NAP still outperforms DrAC* in terms of path lengths.

7. Discussion

We have shown that hybrid neuro-algorithmic policies consisting of deep feature extraction and a shortest path solver – made differentiable via blackbox differentiation (Vlastelica et al., 2020) – *enable learning policies that generalize to unseen environment settings in the low-data regime*. Hybrid architectures are a stepping stone towards the usage of better inductive biases that enable stronger generalization. In NAP, the inductive bias that we impose is the topology of the latent planning graph in conjunction with a planning algorithm. Introducing the shortest-path solver as a module shifts the combinatorial complexity of the planning problem to efficient algorithmic implementations while facilitating the learning of good representations for planning.

Although there is a clear benefit in using NAP, the method comes with certain caveats. We assume that the topological structure (i.e. that there is an underlying grid structure with a set of 5 actions) of the latent planning graph is known a priori. 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. Any results allowing to abandon some of these assumptions will vastly increase the applicability of this method and should be of immediate interest.

Acknowledgment

We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Marin Vlastelica. Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. We acknowledge the support from the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B).

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine learning*, ICML, pp. 1, 2004.

Aghasadeghi, N. and Bretl, T. Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals. In *International Conference on Intelligent Robots and Systems*, IROS, pp. 1561–1566. IEEE, 2011.

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

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning*, ICML, pp. 146–155, 2017.

Asai, M. and Fukunaga, A. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154*, 2017.

Asai, M. and Kajino, H. Towards stable symbol grounding with zero-suppressed state autoencoder. In *International Conference on Automated Planning and Scheduling*, volume 29, pp. 592–600, 2019.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable perturbed optimizers. *arXiv preprint arXiv:2002.08676*, 2020.

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

Blaes, S., Vlastelica, M., Zhu, J., and Martius, G. Control what you can: Intrinsically motivated task-planning agent. In *Advances in Neural Information Processing Systems 32*, NeurIPS, pp. 12541–12552. Curran Associates, Inc., 2019.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Burget, F., Bennewitz, M., and Burgard, W. Bi 2 rrt*: An efficient sampling-based path planning framework for task-constrained mobile manipulation. In *Conference on Intelligent Robots and Systems*, IROS, pp. 3714–3721. IEEE, 2016.

Chen, Y., Shi, Y., and Zhang, B. Optimal control via neural networks: A convex approach. In *International Conference on Learning Representations*, ICLR, 2018.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*, ICML, pp. 2048–2056. PMLR, 2020.

Daw, N. D., Niv, Y., and Dayan, P. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8 (12):1704–1711, 2005.

Demirovic, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., and Guns, T. Predict+optimise with ranking objectives: Exhaustively learning linear functions. In *International Joint Conference on Artificial Intelligence*, IJCAI, pp. 1078–1085, 2019. doi: 10.24963/ijcai.2019/151.

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

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

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". *Management Science*, 2021. doi: 10.1287/mnsc.2020.3922. in press, arXiv preprint 1710.08005.

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.

Ferber, A., Wilder, B., Dilkina, B., and Tambe, M. MIPaaL: Mixed Integer Program as a Layer. In *Conference on Artificial Intelligence*, AAAI, pp. 1504–1511, 2020.

Finn, C. and Levine, S. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793. IEEE, 2017.

Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004. IEEE, 2014.

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*, pp. 2555–2565, Long Beach, California, USA, 2019.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pp. 4565–4573, 2016.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pp. 12519–12530, 2019.

Kaiser, L., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for Atari. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1xCPJHtDB.

Karkus, P., Ma, X., Hsu, D., Kaelbling, L. P., Lee, W. S., and Lozano-Pérez, T. Differentiable algorithm networks for composable robot learning. In *Robotics: Science and Systems XV*, RSS, 2019. doi: 10.15607/RSS.2019.XV.039.

Kumar, R., Mandalika, A., Choudhury, S., and Srinivasa, S. Lego: Leveraging experience in roadmap generation for sampling-based planning. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1488–1495, 2019.

Kuo, Y.-L., Barbu, A., and Katz, B. Deep sequential models for sampling-based planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6490–6497. IEEE, 2018.

Li, Y., He, H., Wu, J., Katabi, D., and Torralba, A. Learning compositional koopman operators for model-based control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1ldzA4tPr.

Mandi, J., Demirovic, E., Stuckey, P. J., and Guns, T. Smart predict-and-optimize for hard combinatorial optimization problems. *AAAI Conference on Artificial Intelligence*, 34 (02):1603–1610, 2020. doi: 10.1609/aaai.v34i02.5521.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Neu, G. and Szepesvári, C. Apprenticeship learning using inverse reinforcement learning and gradient methods. *arXiv preprint arXiv:1206.5264*, 2012.

Ng, A. Y. and Russell, S. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, ICML, 2000.

Niculae, V., Martins, A. F., Blondel, M., and Cardie, C. Sparsemap: Differentiable sparse structured inference. *arXiv preprint arXiv:1802.04223*, 2018.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *Advances in Neural Information Processing Systems*, pp. 6118–6128, 2017.

Pinneri, C., Sawant, S., Blaes, S., and Martius, G. Extracting strong policies for robotics tasks from zero-order trajectory optimizers. In *International Conference on Learning Representations*, ICLR, 2021.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 30:5690–5701, 2017.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

Reddy, S., Dragan, A. D., and Levine, S. Sqil: Imitation learning via reinforcement learning with sparse rewards. *arXiv preprint arXiv:1905.11108*, 2019.

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.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, AISTATS, pp. 627–635, 2011.

Savinov, N., Dosovitskiy, A., and Koltun, V. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018.

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. *Nature*, pp. 604–609, 2020. doi: 10.1038/s41586-020-03051-4.

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*, 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, 2018.

Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pp. 528–536, 2008.

Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. Value iteration networks. In *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

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.

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. SAT-Net: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, volume 97 of *ICML*, pp. 6545–6554, 2019.

Wu, G., Say, B., and Sanner, S. Scalable planning with deep neural network learned transition models. *Journal of Artificial Intelligence Research*, 68:571–606, 2020.

Yao, H., Bhatnagar, S., Diao, D., Sutton, R. S., and Szepesvári, C. Multi-step dyna planning for policy evaluation and control. In *Advances in Neural Information Processing Systems*, pp. 2187–2195, 2009.

Yonetani, R., Taniai, T., Barekatain, M., Nishimura, M., and Kanezaki, A. Path planning using neural A* search. *arXiv preprint arXiv:2009.07476*, 2020.