First-Order Representation Languages for Goal-Conditioned RL

Simon Ståhlberg, Hector Geffner

RWTH Aachen University, Germany
simon.stahlberg@gmail.com, hector.geffner@ml.rwth-aachen.de

Abstract

First-order relational languages have been used in MDP planning and reinforcement learning (RL) for two main purposes: specifying MDPs in compact form, and representing and learning policies that are general and not tied to specific instances or state spaces. In this work, we instead consider the use of first-order languages in goal-conditioned RL and generalized planning. The question is how to learn goal-conditioned and general policies when the training instances are large and the goal cannot be reached by random exploration alone. The technique of Hindsight Experience Replay (HER) provides an answer to this question: it relabels unsuccessful trajectories as successful ones by replacing the original goal with one that was actually achieved. If the target policy must generalize across states and goals, trajectories that do not reach the original goal states can enable more data- and time-efficient learning. In this work, we show that further performance gains can be achieved when states and goals are represented by sets of atoms. We consider three versions: goals as full states, goals as subsets of the original goals, and goals as lifted versions of these subgoals. The result is that the latter two successfully learn general policies on large planning instances with sparse rewards by automatically creating a curriculum of easier goals of increasing complexity. The experiments illustrate the computational gains of these versions, their limitations, and opportunities for addressing them.

1 Introduction

Reinforcement learning (RL) provides a powerful framework for training agents to maximize rewards (Sutton and Barto 1998; Bertsekas and Tsitsiklis 1996). A significant challenge in RL arises in scenarios with large state spaces and sparse rewards, where random exploration often fails to achieve successful outcomes, leading to learning deadlock. To mitigate this issue, *Hindsight Experience Replay (HER)* (Andrychowicz et al. 2017) has been proposed as an effective technique. HER operates in the *goal-conditioned RL* (GCRL) setting, where policies are developed to generalize across both states and goals (Andrychowicz et al. 2017; Nasiriany et al. 2019; Eysenbach et al. 2019, 2022; Chane-Sane, Schmid, and Laptev 2021). The core concept in HER involves relabeling unsuccessful trajectories: instead of treating a trajectory as a failure to reach the intended goal g, it is reinterpreted as successfully reaching the goal g' that was actually achieved. By breaking the deadlock caused by sparse rewards, HER significantly accelerates learning, even in challenging tasks like solving the Rubik's cube (Agostinelli et al. 2019).

Interestingly, GCRL shares similarities with *generalized planning* (Martín and Geffner 2004; Srivastava, Immerman, and Zilberstein 2008; Hu and Giacomo 2011; Celorrio, Segovia-Aguas, and Jonsson 2019), which focuses on learning policies that generalize across different states and goals within a planning domain. This problem has been explored through various approaches, including SAT-based (Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021) and reinforcement learning approaches (Ståhlberg, Bonet, and Geffner 2022b, 2023). In the RL context, generalized planning resembles GCRL but differs in using relational, first-order languages to represent both states and goals, akin to STRIPS (Guestrin et al. 2003; Sanner and Boutilier 2009). Furthermore, the scope of generalization is explicitly defined by the lifted description of the planning domain (Haslum et al. 2019).

This paper introduces and evaluates three variants of HER adapted for the planning setting. The first, *state HER*, relabels goals as sets (conjunctions) of ground atoms representing full states. The second, *propositional HER*, limits relabeled goals to subsets of ground atoms found in the original goal. The third, *lifted HER*, uses lifted versions of propositional goals that maintain essential structural dependencies. Experimental results demonstrate that the latter two variants are particularly effective, automatically constructing a learning curriculum by progressively identifying more challenging trajectories (e.g., larger goal sizes and states farther from the initial state). Unlike previous RL approaches to generalized planning (Ståhlberg, Bonet, and Geffner 2023), which were limited to small instances requiring precomputation of optimal values $V^*(s)$ for uniform sampling, our approach is scalable to billions of states without such precomputation.

The remainder of this paper is organized as follows: we begin with a review of relevant background in classical planning and GCRL, followed by a detailed description of how states, actions, and goals are encoded for the relational GNN architecture, the RL algorithm used (a variant of DQN for GCRL), and the considered HER variants. We then present our experiments and discuss the results, interspersed with related work throughout the introduction.

2 Background

In this section, we present a brief overview of classical planning, goal-conditioned reinforcement learning (GCRL), and relational graph neural networks (R-GNN), which constitute the foundation of our approach.

Classical Planning

A planning instance is defined as a tuple $\langle \mathcal{P}, \hat{\mathcal{A}}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. In this paper, we use the hat notation, $\hat{X}$, to denote first-order (lifted) constructs, which may contain variables as well as constants. When the hat is omitted, X refers to the ground (propositional) version, where all variables have been replaced by constants. We denote $\langle \mathcal{P}, \hat{\mathcal{A}} \rangle$ as the *domain* and $\langle \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ as the *problem*, where:

- $\mathcal{P}$ represents a set of *predicate symbols*. For each symbol $P \in \mathcal{P}$, there is an associated *arity* n. An *atom* is denoted as $P(x_1, \ldots, x_n)$; if no term x_i is a variable (i.e., all terms are objects), then it is a *ground atom*.
- $\hat{\mathcal{A}}$ denotes a set of *action schemas*. Each action schema $\hat{a} = A(X_1, \ldots, X_k)$ is defined by a name A and a list of parameters $X_1, \ldots, X_k$ (variables). An action schema consists of a *precondition*, a set of atoms that must be true in the current state for the action to be applicable, and an *effect*, a set of atoms that describe how the state is modified when the action is applied.
- $\mathcal{O}$ denotes a set of *objects* (constants).
- $\mathcal{I}$ is the *initial state* and $\mathcal{G}$ is the *goal*. Both are *states*, which are sets of ground atoms over $\mathcal{P}$ and $\mathcal{O}$. A state s is considered a *goal state* if and only if $\mathcal{G} \subseteq s$.

A *ground action* is an instantiation of an action schema $\hat{a}$, where all variables are replaced by objects from $\mathcal{O}$, e.g., $a = A(o_1, \ldots, o_k)$. The objective in planning is to find a sequence of ground actions that, when applied to the initial state $\mathcal{I}$, leads to a state s satisfying $\mathcal{G} \subseteq s$. Each ground action is only applicable if its precondition holds in the current state, and its effect deterministically updates the state.

Goal-Conditioned Reinforcement Learning

We adopt a goal-conditioned RL (GCRL) framework tailored to the characteristics of classical planning. In our formulation, we assume that: the actions are *deterministic*, the environment is *fully observable*, and there is a *fixed initial state*. Additionally, we define a reward function $r(s, a, \mathcal{G})$, and a discount factor $\gamma \in [0, 1]$.

In GCRL, each episode begins with an initial state s_0 and a goal $\mathcal{G}$. In our setting, the initial state and the goal is always fixed to $s_0 = \mathcal{I}$ and $\mathcal{G}$, respectively. At each time step t, the agent selects an action $a_t = \pi(s_t, \mathcal{G}), \quad a_t \in \mathcal{A}[s_t]$, receives an immediate reward $r_t = r(s_t, a_t, \mathcal{G})$, and transitions to a new state s_{t+1}, which is obtained by applying a_t in s_t. The objective of the agent is to maximize its return, defined as the discounted sum of future rewards $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$.

In planning, the primary objective is to solve problems by reaching a goal state. This is typically modeled in one of two ways: a reward is given only upon reaching a goal state, or a constant negative reward is incurred until the problem is solved. We use the latter, setting $r(s, a, \mathcal{G}) = -1$. Goal states are terminal, meaning the episode ends once a goal is reached and no further rewards are accumulated. In states with no applicable ground actions, a dummy action is provided that leaves the state unchanged.

The Q-value function $Q(s, a, \mathcal{G})$ represents the expected return of taking action a in state s while pursuing the goal $\mathcal{G}$. A greedy policy selects the action with the highest return: $\pi_Q(s, \mathcal{G}) = \arg\max_{a \in \mathcal{A}[s]} Q(s, a, \mathcal{G})$. An *optimal policy* π^* satisfies $Q^{\pi^*}(s, a, \mathcal{G}) \geq Q^{\pi}(s, a, \mathcal{G})$ for all states s, ground actions a, goals $\mathcal{G}$, and any policy π. Notably, all optimal policies share the same action-value function, which in turn satisfies the Bellman optimality equation:

$$Q^*(s, a, \mathcal{G}) = r(s, a, \mathcal{G}) + \gamma \max_{a' \in \mathcal{A}[s']} Q^*(s', a', \mathcal{G}), \qquad (1)$$

where s' denotes the state resulting from applying a in s. This equation forms the foundation for Q-learning.

3 Learning General Policies

We begin by explaining the computation of Q-values for each applicable action in a given state relative to a goal. Then, we describe DQN with HER for GCRL.

Q-value Predictions

In classical planning, states are represented as sets of ground atoms, where sizes vary across states, and the number of applicable actions dynamically changes. In this paper, we use a relational graph neural network (R-GNN) architecture (Ståhlberg, Bonet, and Geffner 2022a) to learn Q-value functions in classical planning domains. The R-GNN operates on relational structures and can handle variable-size inputs, making it suitable for classical planning. The R-GNN architecture is a convenient choice as it can directly process sets of ground atoms. Previous work using this architecture (Ståhlberg, Bonet, and Geffner 2022a,b, 2023) focused on learning value functions, which involves generating successor states and computing their values. In contrast, our approach learns a Q-value function that directly estimates the value of executing a specific action in a given state.

For brevity, we omit a detailed description of the R-GNN architecture here. Interested readers can find description in (Ståhlberg, Bonet, and Geffner 2022a) or in the appendix. Conceptually, it functions as a black-box, taking sets $\mathcal{S}'$ and $\mathcal{O}'$ as inputs and generating embeddings $\boldsymbol{f}(o)$ for each object $o \in \mathcal{O}'$. We simply need to define the input sets $\mathcal{S}'$ and $\mathcal{O}'$ to encompass the state, applicable actions, and the goal. Additionally, we design the input so that the Q-values for all applicable ground actions are computed in parallel.

It is crucial that the input encoding distinguishes between the state, applicable actions and the goal. To achieve this, we introduce new predicate symbols and objects in the input. The state uses original predicate symbols $\mathcal{P}$, applicable actions are represented by new predicate symbols P_A for each action schema A, and the goal by new predicate symbols $P_{\mathcal{G}}$ for each goal predicate $P \in \mathcal{P}$. Additionally, new objects o_a are introduced for each applicable action a. The input $\mathcal{S}'$ and $\mathcal{O}'$ are defined as $\mathcal{S}' = \mathcal{S}_s \cup \mathcal{S}_{\mathcal{A}[s]} \cup \mathcal{S}_{\mathcal{G}}$, where:

- $\mathcal{S}_s = s$ is the set of atoms true in the given state;

- $\mathcal{S}_{\mathcal{A}[s]} = \{P_A(o_a, o_1, \ldots, o_n) : a = A(o_1, \ldots, o_n) \in \mathcal{A}[s]\}$, with P_A and o_a representing new predicate symbols and objects, respectively; and
- $\mathcal{S}_{\mathcal{G}} = \{P_{\mathcal{G}}(o_1, \ldots, o_n) : P(o_1, \ldots, o_n) \in \mathcal{G}\}$, with $P_{\mathcal{G}}$ representing a new predicate symbol.

The set of objects is $\mathcal{O}' = \mathcal{O} \cup \{o_a : a \in \mathcal{A}[s]\}$, the union of objects in the state and new objects representing applicable actions. Notably, the atom $P_A(o_a, \ldots)$ for action a includes terms of a, indirectly connecting o_a to state and goal atoms.

Initialization of the R-GNN requires knowledge of all predicate symbols. No new predicate symbol depends on a specific problem instance, ensuring a fixed set of new symbols for each domain. Consequently, the model can be trained on one instance set and applied to others within the same domain. The output of R-GNN$(\mathcal{S}', \mathcal{O}')$ is a set of embeddings $\boldsymbol{f}(o)$ for each object $o \in \mathcal{O}'$. To compute the Q-value for a specific action, the embedding of the corresponding action object o_a is used alongside a state summary. The Q-value function is defined as:

$$Q(s, a, \mathcal{G}) = \text{MLP}\left(\boldsymbol{f}(o_a), \sum_{o \in \mathcal{O}} \boldsymbol{f}(o)\right). \tag{2}$$

Here, the sum encompasses embeddings of original state objects (excluding action objects), ensuring the Q-value is conditioned on both action and overall state context, where the goal is implicit. Although using only $\boldsymbol{f}(o_a)$ is possible, it empirically leads to poorer generalization.

Deep Q-Network for GCRL

The presented Q-value function is differentiable and trainable using standard deep reinforcement learning methods. In this section, we briefly outline the training process for the Q-value function using a variant of the DQN algorithm (Mnih et al. 2015); a more detailed description is provided in the appendix. DQN involves two primary stages: (1) *experience generation* by interacting with the environment, and (2) *Q-value function training* using the gathered experience. These stages are interleaved, allowing the agent to learn from its interactions while exploring the environment.

Experience Generation The initial step involves collecting experience tuples $\langle s_t, a_t, r_t, s_{t+1}, \mathcal{G}\rangle$ through policy execution in the environment. These tuples are stored in a *replay buffer*, a finite-memory structure holding recent experiences. Here, individual transition steps are stored in the replay buffer rather than entire trajectories. This decorrelation is safe in our context as classical planning adheres to the Markov property. This step also entails an exploration-exploitation trade-off; we use Boltzmann exploration where the probability of choosing action a in state s with goal $\mathcal{G}$ is:

$$\pi(a \mid s, \mathcal{G}) = \frac{\exp\left(Q(s, a, \mathcal{G})/T\right)}{\sum_{a' \in \mathcal{A}[s]} \exp\left(Q(s, a', \mathcal{G})/T\right)}, \tag{3}$$

with $T > 0$ regulating exploration versus exploitation, with higher T increasing exploration.

Optimization The subsequent step optimizes $Q(s_t, a_t, \mathcal{G})$ via *mini-batch gradient descent* on the loss:

$$\mathcal{L} = \left(Q(s_t, a_t, \mathcal{G}) - y_t\right)^2, \tag{4}$$

$$y_t = r_t + \gamma \max_{a' \in \mathcal{A}[s_{t+1}]} Q(s_{t+1}, a', \mathcal{G}), \tag{5}$$

where y_t is the target value and γ is the discount factor. The value of y_t is computed using a target network that is updated every k episodes using the weights of the main network, with k set to 1 in our experiments. The mini-batch is sampled from the replay buffer.

Experience Refinement Though the prior steps suffice for training a Q-value function, experience quality can be improved. Hindsight Experience Replay (HER) is one approach, generating extra experience by reinterpreting trajectories with different goals (Andrychowicz et al. 2017). Here, the refinement step is a function REFINE operating on a trajectory τ and a problem, yielding refined trajectories $\tau_1, \ldots, \tau_m$. The input includes the problem for problem-specific strategies, like those discussed later. The refined trajectories are stored in the replay buffer.

4 Hindsight Experience Replay Variations

Hindsight Experience Replay (HER) is a technique designed to improve sample efficiency in environments with sparse rewards. The core idea is to reinterpret unsuccessful episodes by substituting the original goal with an alternative goal that was achieved during the episode. This allows the agent to learn from every experience, regardless of whether the original goal was met. More precisely, an experience tuple is stored as $\langle s, a, r, s', \mathcal{G}\rangle$, where $\mathcal{G}$ is the original goal. HER then generates additional transitions by: sampling a new goal $\mathcal{G}'$ from states encountered later in the same episode; recalculating the reward using $\mathcal{G}'$; and augmenting the replay buffer with these hindsight experiences. This procedure is implemented by the aforementioned REFINE function.

We introduce three variants of HER tailored to the classical planning setting: *state HER*, *propositional HER*, and *lifted HER*. State HER closely mirrors the original HER algorithm, whereas propositional and lifted HER exploit the structure of the goal to generate more informative hindsight experiences. The only difference between the three variants is the definition of the HINDSIGHTGOAL function, which determines the hindsight goal for each subtrajectory. In all three cases, the algorithm greedily extracts the longest possible non-overlapping, cycle-free subtrajectories that achieve the hindsight goal only at their final states. Although it is possible to allow overlapping subtrajectories, this may reduce the diversity of experiences in the replay buffer.

State HER

In standard HER (Andrychowicz et al. 2017), the hindsight goal is defined as the state achieved at the end of the subtrajectory. Formally,

$$\text{HINDSIGHTGOAL}(s, \mathcal{G}) = s. \tag{6}$$

This approach treats the entire final state, including both fluent and static atoms, as the new goal. While static atoms are always satisfied and could be omitted for conciseness, we retain them here to remain consistent with the original HER formulation.

Propositional HER

Propositional HER differs from state HER by exploiting the structure of the original goal rather than ignoring it entirely.

Algorithm 1: Generation of lifted goals.

Input: Grounded goal $\mathcal{G}$
Output: Lifted goals $\hat{\boldsymbol{G}}$
Initialize $\hat{\boldsymbol{G}} \leftarrow \emptyset$
Construct a goal-dependency graph $G(\mathcal{G}) = (V, E)$:
Identify connected components $C_1, \ldots, C_k$ of G
Let C_i^* be the set of all subgraphs of C_i containing at most one connected component
Compute the cartesian product $C^* = C_1^* \times \cdots \times C_k^*$, and combine subgraphs
for each $(V', E') \in C^*, V' \neq \emptyset$ **do**
    Let $\hat{\mathcal{G}}$ be V' with objects replaced by variables
    For each pair of variables in $\hat{\mathcal{G}}$, add $X \neq Y$ to $\hat{\mathcal{G}}$
    Add $\hat{\mathcal{G}}$ to $\hat{\boldsymbol{G}}$
end for

Instead of substituting the goal with the final state, propositional HER constructs a new goal by selecting the largest subset of the original goal that is achieved at the end of the subtrajectory. For instance, in the Blocks domain, if the original goal is to build a tower of blocks but only a partial stack is achieved, the hindsight goal becomes the maximal subset of goal atoms satisfied in the final state. Formally, the hindsight goal is defined as:

$$\textsc{HindsightGoal}(s, \mathcal{G}) = \arg\max_{\mathcal{G}' \subseteq \mathcal{G}} \left\{ |\mathcal{G}'| : \mathcal{G}' \subseteq s \right\}. \quad (7)$$

This method is especially useful in domains where goals are expressed as conjunctions of atoms, letting the agent to learn from partial achievements. However, in cases where the goal consists of a single atom, propositional HER offers no advantage over the original goal, as no proper subset exists.

Lifted HER

The limitations of propositional HER can be addressed by lifting goals to the first-order level, enabling the generation of *analogous* hindsight goals that capture structural similarities rather than requiring exact matches. For example, in the Blocks domain, a trajectory may construct a tower using blocks different from those specified in the original goal. Such a trajectory is still valuable, as it demonstrates the ability to achieve the underlying goal structure. To achieve this, lifted HER lifts the original goal to a first-order representation by replacing constants with variables, allowing for alternative groundings. However, this process requires making implicit constraints explicit, such as ensuring variables are assigned to distinct objects and that the goal forms a similar structure (e.g., a tower).

The first step is to construct a goal-dependency graph for $\mathcal{G}$, which captures the relationships between atoms in the grounded goal based on shared objects. For example, consider a goal that requires building a single tower in the Blocks domain. We want to avoid generating lifted subgoals such as $\textsc{On}(X, Y) \wedge \textsc{On}(Z, W)$, where all four variables can be bound to different objects. In practice, such subgoals are likely to correspond to disconnected structures, which are not meaningful in the context of the original goal. The goal-dependency graph prevents the enumeration of these structurally irrelevant subgoals by ensuring that only connected subgraphs, corresponding to coherent goal structures, are considered during lifted goal generation.

Definition 1. *A goal-dependency graph $G(\mathcal{G}) = (V, E)$ is an undirected graph where each vertex $v \in V$ corresponds to an atom in $\mathcal{G}$, and an edge $(p, p') \in E$ exists if and only if p and p' share at least one object in its arguments.*

The second step is to identify subgraphs of $G(\mathcal{G})$ that do not split any *existing* connected components into multiple components. In the Blocks example, this means that the subgraph $\textsc{On}(a, b) \wedge \textsc{On}(b, c)$ is valid, while $\textsc{On}(a, b) \wedge \textsc{On}(c, d)$ is not as it splits the goal into two disconnected components. Note that $G(\mathcal{G})$ may contain multiple connected components, we generate valid subgoals for each connected component separately and later combine them.

The third step is to lift the propositional subgoals to first-order representations. This is done by replacing the constants in the subgraph with variables, and adding inequality constraints to ensure that the variables are distinct. For example, the propositional subgoal $\textsc{On}(a, b) \wedge \textsc{On}(b, c)$ is lifted to $\textsc{On}(X, Y) \wedge \textsc{On}(Y, Z) \wedge (X \neq Y) \wedge (Y \neq Z)$.

Algorithm 1 outlines the procedure for generating lifted goals. Given a grounded goal $\mathcal{G}$, the algorithm produces a set of lifted goal schemas $\hat{\boldsymbol{G}}$. We define $\textsc{Ground}(\hat{\mathcal{G}}, s)$ as a grounding of the goal schema $\hat{\mathcal{G}}$ that is satisfied in the state s, or $\perp$ if no such grounding exists. In other words, this operation returns a propositional instantiation of $\hat{\mathcal{G}}$ that holds in s. The HINDSIGHTGOAL function for lifted HER is then defined as follows:

$$\hat{\mathcal{G}}^* = \arg\max_{\hat{\mathcal{G}} \in \hat{\boldsymbol{G}}} \left\{ |\hat{\mathcal{G}}| : \textsc{Ground}(\hat{\mathcal{G}}, s) \neq \perp \right\} \quad (8)$$

$$\textsc{HindsightGoal}(s, \mathcal{G}) = \textsc{Ground}(\hat{\mathcal{G}}^*, s) \quad (9)$$

If there is no grounding, then $\textsc{HindsightGoal}(s, \mathcal{G}) = \perp$.

Example

Consider a goal in the Blocks domain that requires stacking blocks b_1, b_2, b_3, and b_4 in the specific order, with b_1 on top and b_4 at the bottom. Propositional HER would generate the following subgoals:

- $\textsc{On}(b_1, b_2) \wedge \textsc{On}(b_2, b_3) \wedge \textsc{On}(b_3, b_4)$,
- $\textsc{On}(b_1, b_2) \wedge \textsc{On}(b_2, b_3)$, $\textsc{On}(b_2, b_3) \wedge \textsc{On}(b_3, b_4)$,
- $\textsc{On}(b_1, b_2)$, $\textsc{On}(b_2, b_3)$, $\textsc{On}(b_3, b_4)$.

In contrast, lifted HER generates subgoal schemas that generalize beyond the specific blocks b_1, b_2, b_3, and b_4. The resulting lifted subgoals are:

- $\textsc{On}(X, Y) \wedge \textsc{On}(Y, Z) \wedge \textsc{On}(Z, W) \wedge X \neq Y \wedge \ldots$,
- $\textsc{On}(X, Y) \wedge \textsc{On}(Y, Z) \wedge X \neq Y \wedge \ldots$,
- $\textsc{On}(X, Y) \wedge X \neq Y$.

In state HER, the hindsight goal is the entire final state of the subtrajectory, including atoms such as $\textsc{Clear}(b_1)$ and $\textsc{On-Table}(b_4)$ that are unrelated to the original goal. Including these extra atoms might result in a goal that is harder to achieve than the original, since all blocks must be arranged in the specified configuration.

5 Experiments

We first describe the benchmark used in the experiments, then explain how the training and testing were set up. Finally, we discuss the results of the experiments.

Domain	Train	Validation	Test
Blocks	2 – 14 blk	15 – 19 blk	20 – 50 blk
Childsnack	1 – 6 chd	7 – 12 chd	13 – 50 chd
Childsnack-AF	1 – 10 chd	11 – 19 chd	20 – 119 chd
Delivery	2 – 5 grd, 1 – 5 pkg	6 – 9 grd, 1 – 7 pkg	10 – 25 grd, 6 – 10 pkg
Gripper	1 – 19 bal	20 – 29 bal	30 – 129 bal
Hiking	1 – 6 cpl, 1 – 4 loc	6 – 10 cpl, 2 – 5 loc	10 – 26 cpl, 5 – 7 loc
Miconic	1 – 19 ppl, 2 – 20 flr	20 – 29 ppl, 21 – 30 flr	30 – 80 ppl, 31 – 81 flr
Reward	2 – 12 grd, 1 – 7 rwd	13 – 15 grd, 8 – 13 rwd	16 – 29 grd, 8 – 14 rwd
Spanner	1 – 14 nut	15 – 24 nut	25 – 50 nut
Visitall	2 – 12 grd	13 – 15 grd	16 – 30 grd

Table 1: Overview of the training, validation, and test sets for each domain. The training set is used to fit the models, the validation set to select hyperparameters and the best checkpoint, and the test set to assess final performance. Instances are generated using one or two scaling parameters with even distribution. The parameters are as follows: balls (bal), children (chd), couples (cpl), floors (flr), grid size (grd), locations (loc), nuts (nut), packages (pkg), people (ppl), and rewards (rwd). The suffix "AF" denotes allergy-free instances (i.e., without gluten-intolerant children).

		LAMA			
			Plan Length		
Domain	#	Cov.	Total	Median	Mean
Blocks	100	100	23250	218.0	232.5
Childsnack	100	24	2361	89.0	98.4
Childsnack-AF	100	65	16965	261	261.0
Delivery	100	99	27337	276	276.1
Gripper	100	100	23800	238.0	238.0
Hiking	100	26	2038	77.5	78.4
Miconic	100	100	19556	195.5	195.6
Reward	100	99	12054	116	121.8
Spanner	100	0	0	0	0.0
Visitall	100	100	53004	475.0	530.0
Total	1500	1212	312029	213.5	257.4

Table 2: The coverage and the plan length (total, median, and mean) for each domain for LAMA. The plan length is for the first solution found. LAMA solved 80.8% of the instances.

Domains

The domains used in the experiments are those from Drexler et al. (2024), where C_2 expressiveness is sufficient to distinguish all non-isomorphic states. Models based on R-GNN architecture cannot fit the training data if there are value conflicts; however, the absence of value conflicts does not guarantee that a general policy can be learned. To conserve space, we refer the reader to Drexler et al. (2024) for the domain descriptions. The datasets used are shown in Table 1.

In addition, we also evaluate on a **Maze** domain. In this domain, the task is to navigate a grid to reach a specific location. This domain illustrates the limitaions of propositional HER, as it fails to relabel trajectories in this domain.

Setup

We now describe the experimental setup for training and evaluation. This includes details of the architecture, hyperparameters, and the training and testing procedures. The methods were implemented using PyTorch, and used a lifted successor generator to ground goals (Ståhlberg 2023).[1]

Model In contrast to the original RNN (Ståhlberg, Bonet, and Geffner 2022a), we use residual updates and layer normalization. Furthermore, we use a readout from a random layer in addition to the readout from the last layer (Bansal et al. 2022), imposing the same loss on both. This was necessary to train networks with 100 layers.

Hyperparameters The embedding size was set to 32, using hard maximum for aggregation. The learning rate started at 10^{-3} and decayed linearly to 10^{-6} over 300 episodes. The Boltzmann temperature decayed from 1.0 to 0.1 linearly over 600 episodes. The discount factor was 0.999. Each episode included 32 optimization steps with a batch size of 32. Huber loss (Huber 1964) (delta = 1.0) was used instead of MSE. The replay buffer size was 1000, with prioritized experience replay (Schaul et al. 2016) (priority exponent 0.6, priority weight 0.4). We generated 4 trajectories per episode, each up to 100 steps. The target network was updated after each episode using the main network's weights.

Training We trained 10 models per method and domain, each with a unique random seed, to compute mean and standard deviation of coverage. Training was performed on a system with an Intel Xeon Platinum 8352M CPU, 16 GB RAM, and an NVIDIA A10 GPU with 24 GB VRAM, lasting up to 12 hours per model. Model selection was based on validation coverage; in case of ties, shortest total solution length was used, and then randomly selected.

Testing We evaluated the learned Q-value functions using a greedy policy: at each step, the action with highest Q-value was selected, i.e., $a^* = \arg\max_{a \in \mathcal{A}[s]} Q(s, a, \mathcal{G})$. Only the current state s was used—no lookahead or backtracking. The action a^* updated the state s, repeated until $\mathcal{G} \subseteq s$ or after 1000 steps. Actions leading to previously visited states were excluded to avoid cycles.

Baselines

Learning general policies for planning domains with reinforcement learning (RL) is difficult because rewards are extremely sparse: agents usually receive no reward until the goal is reached, so random exploration almost never produces successful episodes and learning stalls. Consequently, standard RL methods such as REINFORCE (Williams 1992) and Actor-Critic (Sutton et al. 1999) generally fail to learn useful policies without a carefully designed curriculum.

To address reward sparsity, Gehring et al. (2022) propose domain-independent heuristic reward shaping. This technique is orthogonal to our approach and could be combined with it; we therefore omit it from our baselines.

[1] Code and data will be made available upon publication.

	Lifted HER				Propositional HER				State HER			
		Plan Length				Plan Length				Plan Length		
Domain	Cov.	Total	Median	Mean	Cov.	Total	Median	Mean	Cov.	Total	Median	Mean
Blocks	**100**	9672	98.0	96.7	**100**	10730	100.0	107.3	98	10876	112.0	111.0
Blocks-L	**100**	9692	97.0	96.9	**100**	9802	98.0	98.0	78	11456	95.0	146.9
Childsnack	56	4358	69.0	77.8	**100**	9263	90.5	92.6	59	6309	111	106.9
Childsnack-AF	**100**	23424	211.5	234.2	**100**	21150	211.5	211.5	53	8240	141	155.5
Delivery	**12**	3506	243.5	292.2	11	1837	137	167.0	0	0	0	0.0
Gripper	**100**	23800	238.0	238.0	**100**	23988	238.0	239.9	**100**	24400	238.0	244.0
Gripper-L	**100**	23800	238.0	238.0	**100**	23858	239.0	238.6	**100**	23950	238.0	239.5
Hiking	0	0	0	0.0	0	0	0	0.0	0	0	0	0.0
Miconic	**100**	15999	159.0	160.0	**100**	15837	158.5	158.4	**100**	17955	180.5	179.6
Miconic-L	**100**	15870	158.5	158.7	**100**	15929	159.0	159.3	0	0	0	0.0
Reward	58	5564	86.5	95.9	**72**	6596	87.5	91.6	0	0	0	0.0
Reward-L	**85**	7941	89	93.4	78	7139	89.0	91.5	26	2566	95.5	98.7
Spanner	**100**	9555	94.0	95.5	**100**	9555	94.0	95.5	74	6766	90.0	91.4
Visitall	**100**	45329	409.0	453.3	88	34546	363.0	392.6	49	32018	654	653.4
Visitall-L	68	26247	340.0	386.0	**87**	36119	368	415.2	28	13706	483.0	489.5
Total	1179	224757	136	190.6	**1236**	226349	136.5	183.1	765	158242	148	206.9

Table 3: The number of solved instances and the plan length (total, median, and mean) for each domain for the learned models. There are 100 test instances per domain and 15 domains, resulting in a total of 1500 instances. See Table 1 for the training, validation, and test sets. The suffix "L" indicates that training instances too large to be fully expanded were excluded. When generating a solution, a maximum of 1000 steps was allowed, and actions leading to previously visited states were explicitly excluded. Models trained with lifted HER solve 78.6% of instances, while those trained with propositional HER solve 82.4%. In contrast, models trained with state HER solve only 51.0% of instances. The quality of the solutions can be compared to the solutions produced by LAMA (Table 2).

Related work by Ståhlberg, Bonet, and Geffner (2023) learns general policies with RL but is limited to very small state spaces and thus does not scale to the larger instances we consider. Their method also assumes the ability to sample arbitrary initial states, whereas we always start from a fixed initial state.

Our primary baseline is *state HER*, an adaptation of HER (Andrychowicz et al. 2017) to planning. We compare our lifted and propositional HER variants against this baseline, and we evaluate the resulting plan quality against that of the LAMA planner (Richter, Westphal, and Helmert 2011).

Results

The main results are presented in Table 3, which reports the number of solved instances and plan lengths (total, median, and mean) for each domain. Lifted HER solves 78.6% of instances, while propositional HER achieves 82.4%. In comparison, state HER solves only 51.0%.

Table 2 shows the performance of the baseline planner, LAMA. Notably, the learned models often produce significantly shorter solutions than LAMA, indicating that they do not rely on random exploration. Moreover, the learned models solve more instances than LAMA (82.4% vs. 80.8%), despite being restricted to a greedy policy without backtracking, whereas LAMA is allowed to backtrack. Nevertheless, it is clear that the learned models learn general policies that can solve much larger instances, as shown in Table 1.

Failures

There are several domains where learning a general policy failed. In **Delivery**, all packages must be delivered to the same location, but the truck can carry only one package at a time. Random exploration can occasionally deliver a single package, but the likelihood of delivering multiple packages to the same location in a single trajectory is extremely low. As a result, the models typically learn to deliver just one package, failing to generalize to the full task.

In the **Hiking** domain, a similar issue arises: the probability of generating informative trajectories through random exploration is simply too low for effective learning.

For **Childsnack**, and to some extent **Spanner**, learning a general policy is problematic. Although coverage is high, this is due to selecting the best model across runs using the validation set. The issue appears to be the handling of dead-end states: every trajectory ending in a dead-end is relabeled as a success for the relabeled goal. In Childsnack, dead-ends occur when gluten-free ingredients run out; since some children have been fed, the state matches a subgoal. Similarly, in Spanner, if some nuts have been tightened, the trajectory is relabeled as a success. Consequently, there is no training data on how to avoid dead-ends for the original goal.

In **Reward** and **Visitall**, coverage decreases as the grid size increases. This may be due to the model's depth: with larger grids, distant cells may not communicate effectively, limiting the model's ability to tackle larger instances.

Training Curves

Figure 1 shows the number of episodes required by each method to learn a model that achieves a given coverage on the test set of the Blocks domain. Lifted HER learns models that reach nearly 100% coverage with fewer episodes than propositional HER, and exhibits lower variance, especially

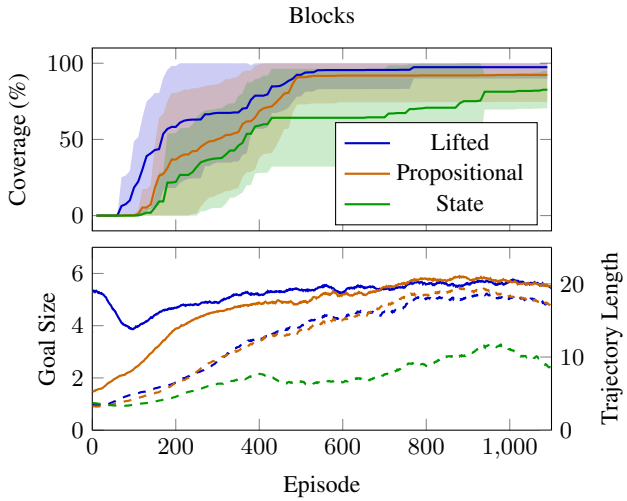

Figure 1: The x-axis shows the episode number during training. In the **upper plot**, the solid line indicates the mean coverage on the test set achieved by the best model up to each episode. In this plot, cycle avoidance is not used when solving the instance; if the policy enters a cycle, the episode is terminated. The shaded area represents the standard deviation. In the **bottom plot**, the solid line (left y-axis) indicates the mean goal size, while the dashed line (right y-axis) shows the mean trajectory length. In **both plots**, 10 different seeds are used.

in later episodes. In contrast, state HER requires substantially more episodes to achieve comparable coverage, and its variance remains high throughout training.

We note that lifted HER and propositional HER have the ability to automatically generate a curriculum of increasing difficulty. Figure 1 illustrates this by plotting the mean goal size and trajectory length during training. Both lifted HER and propositional HER show a clear increase in goal size and trajectory length over time, indicating that the models are exposed to progressively harder tasks. For lifted HER, the mean goal size initially decreases as it learns to deconstruct towers, but later increases as the model learns to solve the original task. State HER is partially omitted from this figure, as its goal sizes remain large and relatively constant throughout training (varying with instance size). However, state HER shows only a modest increase in mean trajectory length, indicating limited curriculum learning.

Due to space constraints, the figures for the remaining domains can be found in the appendix, illustrating that this effect is not limited to the Blocks domain.

Automatic curriculum learning is not a new concept (Portelas et al. 2020), although it often requires a specific mechanism to implement it. In our method, the curriculum emerges naturally through the relabeling process, which focuses on the largest subgoals that can be achieved at any given time. While this is partially present in state HER and in the original formulation of HER, it becomes significantly more evident in propositional and lifted HER.

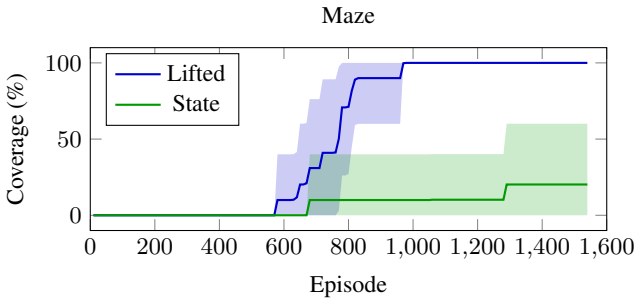

Figure 2: See Figure 1 for a description of the plot. Propositional HER is not shown, as not a single trajectory could be relabeled.

Propositional vs. Lifted HER

A noteworthy limitation of propositional HER is its reliance on achieving smaller subsets of the original goal. This limitation appears in the Maze domain, where the goal is to reach a specific location. The shortest path to the goal in the training set is approximately 65 steps, but with a horizon of 100 steps and the possibility of backtracking or taking dead-end branches, random exploration is unlikely to succeed. Indeed, after running propositional HER for 120 hours (across 10 runs), not a single successful trajectory was found.

In contrast, lifted HER relabels the goal to the actual location reached by the agent. This allows the model to learn a general policy, as demonstrated in Figure 2, where it consistently achieves 100% coverage. State HER can relabel the goal, but these goals include static atoms, which appear to hinder learning; as a result, coverage never reaches 100%.

Propositional HER also has difficulty with tasks such as picking up a particular item among countless others, or clearing a specific block in Blocks when there are hundreds of blocks. The probability of randomly generating a trajectory that achieves the precise goal is extremely low. Lifted HER overcomes this by relabeling which object or block the agent was supposed to target, enabling it to start learning.

6 Conclusions

In this paper, we introduced three variants of Hindsight Experience Replay (HER) for generalized planning: state HER, propositional HER, and lifted HER. These variants adapt HER to the planning setting, where states and goals are represented using first-order languages. State HER relabels goals as full states, propositional HER restricts relabeled goals to subsets of ground atoms that appear in the original problem goal, and lifted HER uses lifted versions of these propositional goals that preserve suitable structural dependencies. We showed that these HER variants can significantly improve the data efficiency and scalability of RL approaches to generalized planning. In particular, they enable general policies to be learned in domains where it is not possible to precompute optimal values for all states. We also demonstrated that these HER variants can automatically construct a more diverse and effective curriculum for learning, leading to faster convergence and better performance.

Acknowledgements

The research has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, Germany, by the European Research Council (ERC), Grant agreement No. 885107, and by the Excellence Strategy of the Federal Government and the NRW Länder, Germany.

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1: 356–363.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2017. Hindsight Experience Replay. In *Proc. of the 31st Conf. on Neural Information Processing Systems (NIPS 2017)*, volume 30, 5048–5058.

Bansal, A.; Schwarzschild, A.; Borgnia, E.; Emam, Z.; Huang, F.; Goldblum, M.; and Goldstein, T. 2022. End-to-end Algorithm Synthesis with Recurrent Networks: Extrapolation without Overthinking. In *Proc. of the 36th Annual Conf. on Neural Information Processing Systems (NeurIPS 2022)*.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proc. of the 33rd AAAI Conf. on Artificial Intelligence (AAAI 2019)*, 2703–2710.

Celorrio, S. J.; Segovia-Aguas, J.; and Jonsson, A. 2019. A Review of Generalized Planning. *Knowledge Engineering Review*, 34.

Chane-Sane, E.; Schmid, C.; and Laptev, I. 2021. Goal-Conditioned Reinforcement Learning with Imagined Subgoals. In *Proc. of the 38th International Conf. on Machine Learning (ICML 2021)*, volume 139, 1430–1440.

Drexler, D.; Ståhlberg, S.; Bonet, B.; and Geffner, H. 2024. Symmetries and Expressive Requirements for Learning General Policies. In *Proc. of the 21st International Conf. on Principles of Knowledge Representation and Reasoning (KR 2024)*.

Eysenbach, B.; Gupta, A.; Ibarz, J.; and Levine, S. 2019. Diversity is All You Need: Learning Skills without a Reward Function. In *Proc. of the 7th International Conf. on Learning Representations (ICLR 2019)*.

Eysenbach, B.; Zhang, T.; Levine, S.; and Salakhutdinov, R. 2022. Contrastive Learning as Goal-Conditioned Reinforcement Learning. In *Proc. of the 36th Annual Conf. on Neural Information Processing Systems (NeurIPS 2022)*, volume 35, 35603–35620.

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In *Proc. of the 35th AAAI Conf. on Artificial Intelligence (AAAI 2021)*, 11801–11808.

Gehring, C.; Asai, M.; Chitnis, R.; Silver, T.; Kaelbling, L. P.; Sohrabi, S.; and Katz, M. 2022. Reinforcement Learning for Classical Planning: Viewing Heuristics as Dense Reward Generators. In *Proc. of the 32nd International Conf. on Automated Planning and Scheduling (ICAPS 2022)*.

Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003. Generalizing Plans to New Environments in Relational MDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 1003–1010.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool.

Hu, Y.; and Giacomo, G. D. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. In *Proc. of the 22nd International Joint Conf. on Artificial Intelligence (IJCAI 2011)*, 918–923.

Huber, P. J. 1964. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1): 73–101.

Martín, M.; and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Applied Intelligence*, 20(1): 9–19.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533.

Nasiriany, S.; Pong, V. H.; Lin, S.; and Levine, S. 2019. Planning with Goal-Conditioned Policies. In *Proc. of the 33rd Annual Conf. on Neural Information Processing Systems (NeurIPS 2019)*, volume 32, 14814–14825.

Portelas, R.; Colas, C.; Weng, L.; Hofmann, K.; and Oudeyer, P. 2020. Automatic Curriculum Learning For Deep RL: A Short Survey. In *Proc. of the 29th International Joint Conf. on Artificial Intelligence (IJCAI 2020)*, 4819–4825.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 Planner Abstracts*, 50–54.

Sanner, S.; and Boutilier, C. 2009. Practical Solution Techniques for First-Order MDPs. *Artificial Intelligence*, 173(5-6): 748–788.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized Experience Replay. In *Proc. of the 4th International Conf. on Learning Representations (ICLR 2016)*.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2008. Learning Generalized Plans Using Abstract Counting. In *Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI 2008)*, 991–997.

Ståhlberg, S. 2023. Lifted Successor Generation by Maximum Clique Enumeration. In *Proc. of the 26th European Conf. on Artificial Intelligence (ECAI 2023)*, 2194–2201.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proc. of the 32nd International Conf. on Automated Planning and Scheduling (ICAPS 2022)*, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proc. of the 19th International Conf. on Principles of Knowledge Representation and Reasoning (KR 2022)*, 474–483.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In *Proc. of the 20th International Conf. on Principles of Knowledge Representation and Reasoning (KR 2023)*.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proc. of the 13th Annual Conf. on Neural Information Processing Systems (NIPS 1999)*, 1057–1063.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4): 229–256.

First-Order Representation Languages for Goal-Conditioned RL: Appendix

Algorithm 1: R-GNN

Input: Ground atoms $\mathcal{S}$, objects $\mathcal{O}$
Output: Embeddings $\boldsymbol{f}_L(o)$ for each $o \in \mathcal{O}$
Initialize $\boldsymbol{f}_0(o) \sim 0^k$ for each $o \in \mathcal{O}$
for $i \in \{0, \dots, L-1\}$ **do**
  for each atom $q := P(o_1, \dots, o_m) \in \mathcal{S}$ **do**
    $\boldsymbol{m}_{q,o_j} := [\text{MLP}_P(\boldsymbol{f}_i(o_1), \dots, \boldsymbol{f}_i(o_m))]_j$
  end for
  for each $o \in \mathcal{O}$ **do**
    $\boldsymbol{m}_o := \max\big(\{\!\{\boldsymbol{m}_{q,o} : o \in q, q \in \mathcal{S}\}\!\}\big)$
    $\boldsymbol{f}_{i+1}(o) := \boldsymbol{f}_i(o) + \text{LN}\big(\text{MLP}_U\big(\boldsymbol{f}_i(o), \boldsymbol{m}_o\big)\big)$
  end for
end for

A Introduction

This appendix complements the paper *First-Order Representation Languages for Goal-Conditioned RL*. It provides further details on three core components: relational graph neural networks, deep Q-networks with experience refinement, and automated curriculum learning. The appendix includes pseudo-code and plots that illustrate the impact of curriculum learning observed in our experiments. Additionally, a miscellaneous section presents further details.

B Relational Graph Neural Network

Graph Neural Networks (GNNs) operate on graphs, whereas planning states are represented as relational structures with predicates of varying arities. To bridge this gap, the Relational Graph Neural Network (R-GNN) (Ståhlberg, Bonet, and Geffner 2022) adapts GNNs to handle such relational structures by exchanging information between objects based on the ground atoms that are true in the state. In a R-GNN, each object $o \in \mathcal{O}$ is associated with an embedding $\boldsymbol{f}_i(o) \in \mathbb{R}^k$, where k is the embedding dimension and i denotes the iteration. Initially, all embeddings are set to zero vectors: $\boldsymbol{f}_0(o) = 0^k$. The R-GNN updates these embeddings iteratively as follows:

$$\boldsymbol{f}_{i+1}(o) := \boldsymbol{f}_i(o) + \text{comb}_i\big(\boldsymbol{f}_i(o), \boldsymbol{m}_o\big)\,, \tag{1}$$

where $\boldsymbol{m}_o$ is the aggregated message received by object o, and $\text{comb}_i(\cdot)$ is a combination function. The aggregated message is defined as:

$$\boldsymbol{m}_o := \text{agg}\big(\{\!\{\boldsymbol{m}_{q,o} : o \in q,\ q \in \mathcal{S}\}\!\}\big)\,, \tag{2}$$

where $\boldsymbol{m}_{q,o}$ is the message sent from atom q to object o, computed by:

$$\boldsymbol{m}_{q,o_j} := [\text{comb}_P(\boldsymbol{f}_i(o_1), \dots, \boldsymbol{f}_i(o_m))]_j\,, \tag{3}$$

with $q = P(o_1, \dots, o_m)$ and $o_j = o$. The notation $[\cdot]_j$ selects the j-th output, corresponding to the position of o in q. Messages are computed for each true atom, and $\boldsymbol{m}_o$ aggregates all messages received by o.

Algorithm 1 outlines the procedure. In our implementation, we use the $\max$ function for aggregation, and all combination functions $\text{comb}_i(\cdot)$ are implemented as multilayer perceptrons (MLPs) consisting of a non-linear layer with Mish activation (Misra 2020) followed by a linear layer. Unlike (Ståhlberg, Bonet, and Geffner 2022), we use a residual connection (He et al. 2016) to combine the current embedding $\boldsymbol{f}_i(o)$ with the aggregated message.

Algorithm 2: Deep Q-Network (DQN). The pseudo-code omits mini-batches for clarity; in practice, mini-batches are used for both problems and replay buffer transitions. Line 9 is not part of the original DQN algorithm.

Input: Problems $\mathbf{P}_1, \dots, \mathbf{P}_n$, and refinement strategy REFINE
Output: Policy π_θ
Initialize policy parameters θ randomly
Initialize target network parameters $\theta_{\text{target}} \leftarrow \theta$
Initialize replay buffer $\mathcal{D}$
for each episode **do**
  Sample $\mathbf{P}_i = \langle \mathcal{P}, \hat{\mathcal{A}}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle, \quad i \sim \text{Uniform}(1, n)$
  Generate trajectory τ using π_θ from $\mathcal{I}$
  Refine experience $\tau_1, \dots, \tau_m = \text{REFINE}(\tau, \mathbf{P}_i)$
  Store all transitions from $\tau_1, \dots, \tau_m$ in $\mathcal{D}$
  Sample transition $(s_t, a_t, r_t, s_{t+1}, \mathcal{G})$ from $\mathcal{D}$
  Compute target:

$$y_t = r_t + \gamma \max_{a' \in \mathcal{A}[s_{t+1}]} Q(s_{t+1}, a', \mathcal{G}; \theta_{\text{target}})$$

  Update policy parameters θ by minimizing:

$$\mathcal{L} = (Q(s_t, a_t, \mathcal{G}; \theta) - y_t)^2$$

  Update $\theta_{\text{target}} \leftarrow \theta$ every k episodes
end for

Algorithm 3: Template for the REFINE function. The selection and construction of hindsight goals is delegated to the HINDSIGHTGOAL function. This procedure greedily extracts the longest non-overlapping, cycle-free subtrajectories where the hindsight goal is achieved only at the end.

Param.: Function HINDSIGHTGOAL : $\mathcal{S} \times \mathcal{S} \to \mathcal{S} \cup \{\bot\}$
Input: Trajectory $\tau = \langle s_0, a_0, r_0, s_1, \ldots, s_n \rangle$, and goal $\mathcal{G}$
Output: Trajectories $\tau_1, \ldots, \tau_m$
$end \leftarrow n$
while $end > 0$ **do**
  $start \leftarrow end - 1$
  $\mathcal{G}' = \text{HINDSIGHTGOAL}(s_{end}, \mathcal{G})$
  if $\mathcal{G}' \neq \bot$ **then**
    $\tau' \leftarrow \langle\rangle$
    $visited \leftarrow \{s_{end}\}$
    while $(start \geq 0) \wedge (\mathcal{G}' \not\subseteq s_{start}) \wedge (s_{start} \notin visited)$ **do**
      $r'_{start} \leftarrow \ldots$ {Recompute reward w.r.t. $\mathcal{G}'$}
      $\tau' \leftarrow \langle \langle s_{start}, a_{start}, r'_{start}, s_{start+1}, \mathcal{G}' \rangle \rangle \oplus \tau'$
      $visited \leftarrow visited \cup \{s_{start}\}$
      $start \leftarrow start - 1$
    end while
    if $\tau' \neq \langle\rangle$ **then**
      output τ'
    end if
  end if
  $end \leftarrow start$
end while

C DQN with Experience Refinement

In addition to the description in the main text, Algorithm 2 presents the corresponding pseudo-code for the DQN procedure. Here, on Line 9, the refinement step is applied to the trajectory τ and problem $\mathbf{P}$, yielding refined trajectories $\tau_1, \ldots, \tau_m$. The refinement step is detailed in Algorithm 3, which serves as a template for the REFINE function used in our experiments. This REFINE function acts as a generic placeholder for a task-specific refinement strategy that modifies the trajectory generated by the policy. The main text introduces three such refinement methods: state HER, propositional HER, and lifted HER. These three only differ in how they implement the HINDSIGHTGOAL function.

D Curriculum Learning Curves

The main text presented curves for the Blocks domain, illustrating how the size of goals and trajectory lengths evolve during training. To illustrate that this effect is not limited to the Blocks domain, we provide additional plots in this section. We note that state HER is omitted from these plots, as its goal size remains a large constant, which hinders the visualization of trends in the other HER variants.

Figures 1 and 2 illustrate the average goal size and trajectory length for different HER approaches across a variety of domains. The trend discussed in the main text is clearly evident not only in the Blocks domain but also in other environments such as Childsnack, Childsnack-allergy-free, Gripper, Miconic, Reward, and Visitall. However, this effect is not consistently observed in all domains–for example, Delivery, Hiking, and Spanner show limited or no such progression.

Figure 3 offers additional insights into trajectory lengths for selected domains, this time including state HER. In domains like Miconic, Reward, and to a lesser extent Visitall, lifted HER and propositional HER exhibit increasing trajectory lengths over the course of training, while state HER remains relatively unchanged. This suggests that lifted and propositional HER are more effective in creating a curriculum. The reason lifted HER performs worse than propositional HER in Visitall is due to the restrictions imposed on the size of lifted goals, as discussed below, which indirectly limit the length of the trajectories.

E Miscellaneous Details

Grounding Unlike state HER or propositional HER, lifted HER involves grounding lifted goals, which is a computationally intensive process. The complexity increases exponentially with the number of variables involved. To maintain feasibility, we cap the size of lifted goals (excluding inequality constraints) at a maximum of 10 atoms. In addition, we limit the number of lifted goals considered per size, as the Cartesian product of subgraphs can result in an exponential number of possible combinations. For further efficiency, we restrict our search to a single grounding per lifted goal within the state, in order to avoid generating an exponential number of groundings. These limitations are essential to ensure that the algorithm remains viable for real-world use.

References

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proc. of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, 770–778. IEEE Computer Society.

Misra, D. 2020. Mish: A Self Regularized Non-Monotonic Activation Function. In *Proceedings of the 31st British Machine Vision Conference (BMVC 2020)*. BMVA Press.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proc. of the 32nd International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 629–637. AAAI Press.

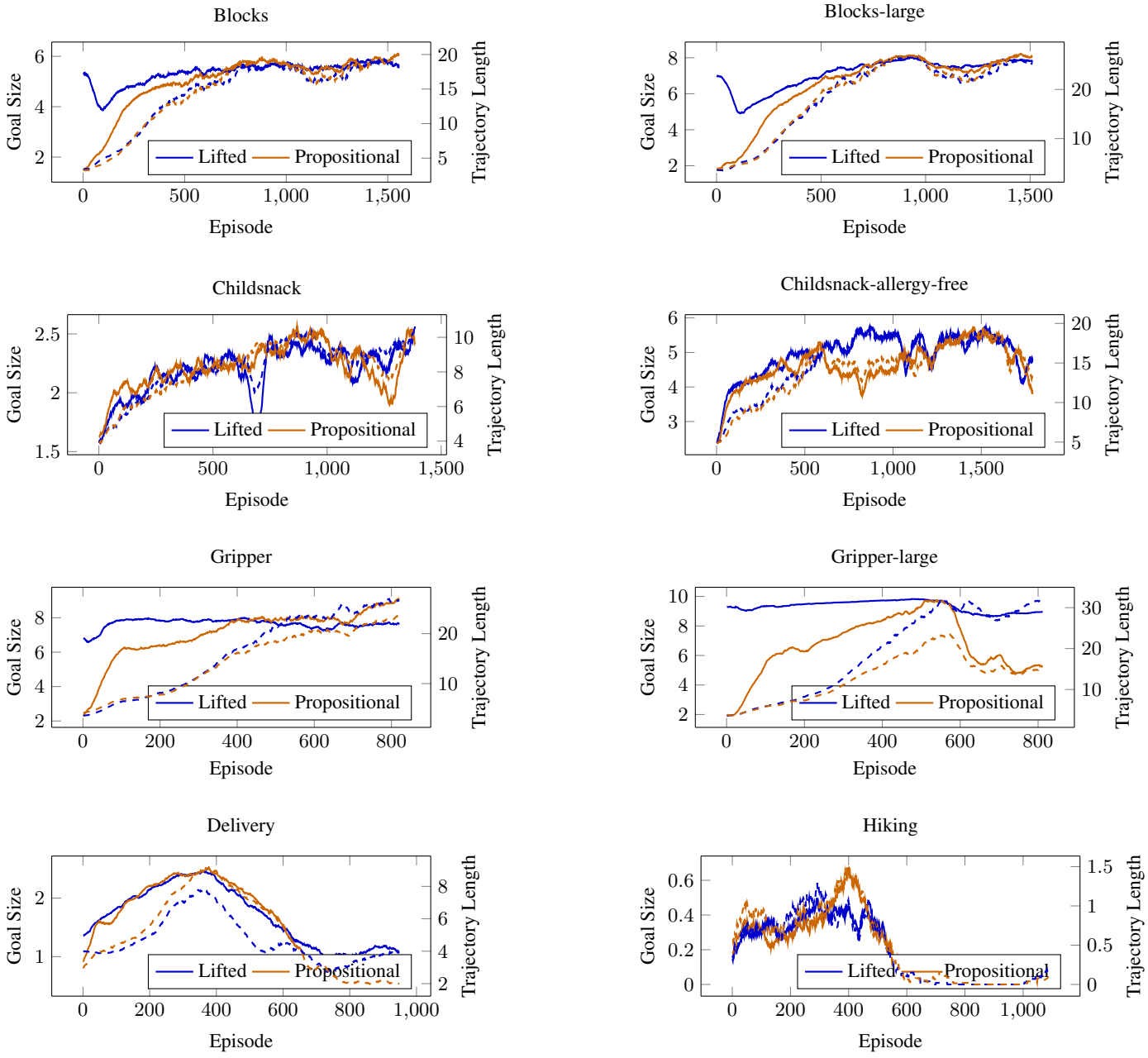

Figure 1: The x-axis shows the episode number during training. The solid line (left y-axis) indicates the mean goal size, while the dashed line (right y-axis) shows the mean trajectory length. 10 different seeds are used to compute the mean. State HER is not shown as it tends to yield a constant goal size, making it difficult to visualize the other variations. Curves are smoothed using a moving average to make the figures more visible.

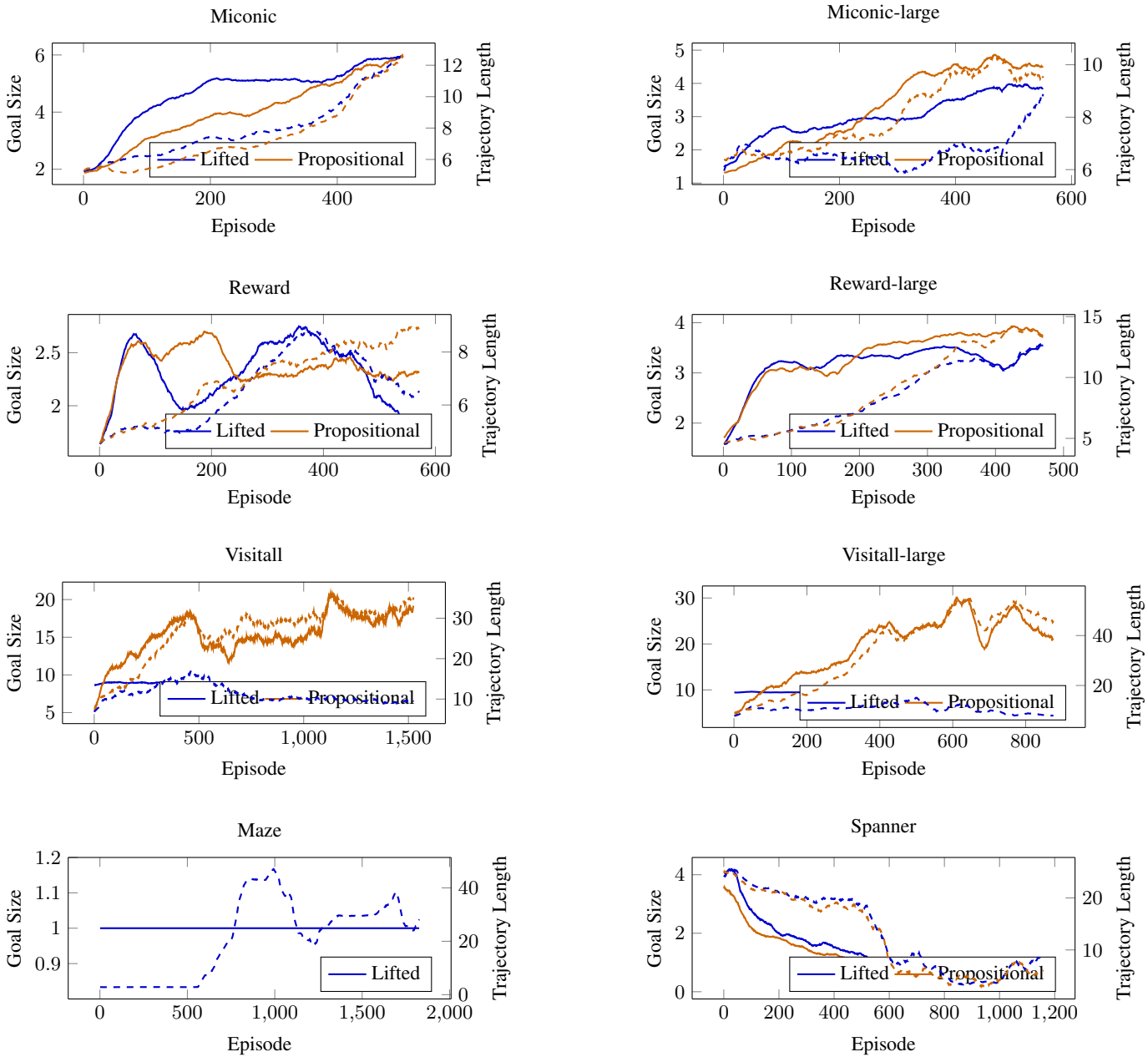

Figure 2: The x-axis shows the episode number during training. The solid line (left y-axis) indicates the mean goal size, while the dashed line (right y-axis) shows the mean trajectory length. 10 different seeds are used to compute the mean. State HER is not shown as it tends to yield a constant goal size, making it difficult to visualize the other variations. Curves are smoothed using a moving average to make the figures more visible.

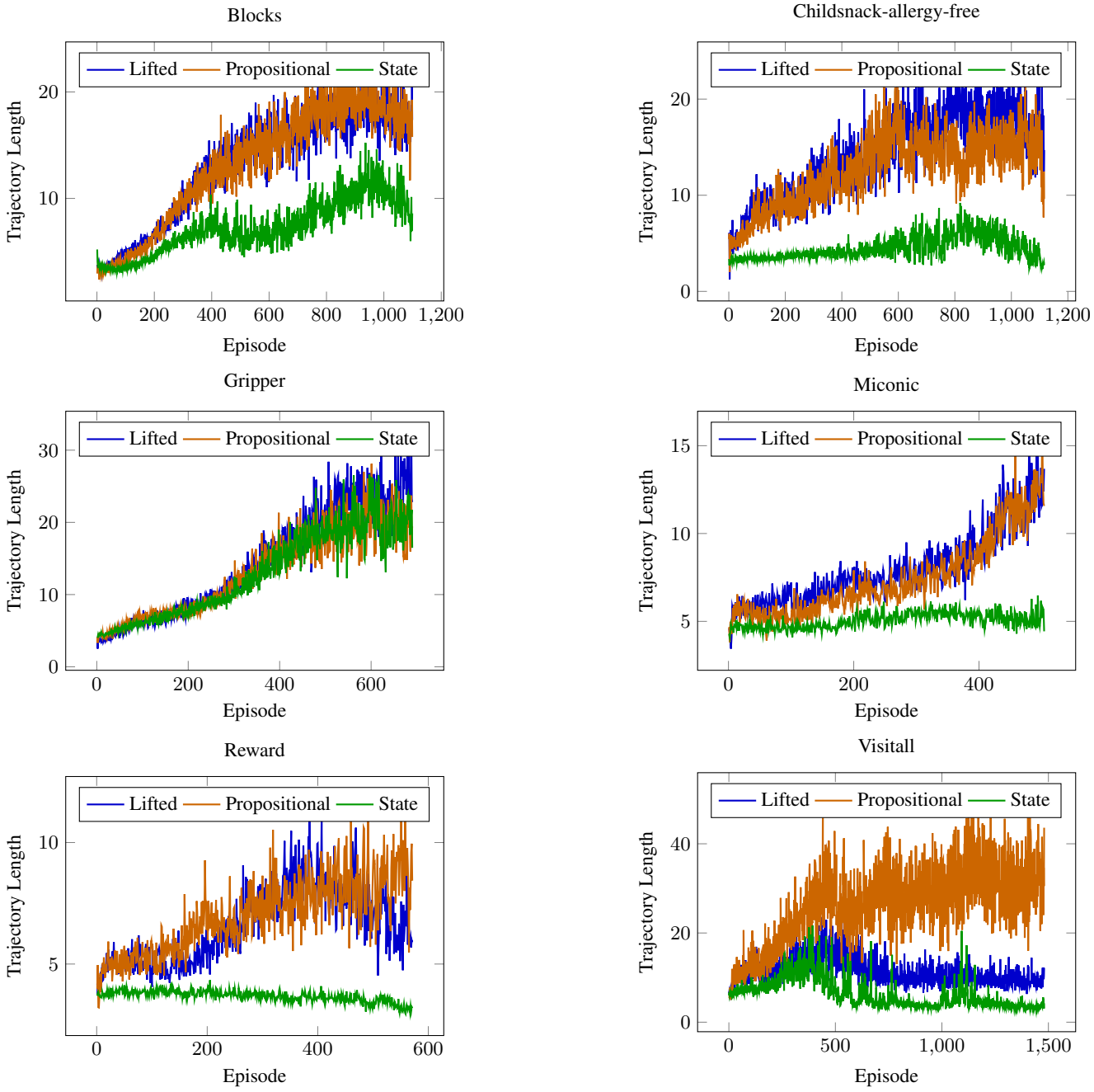

Figure 3: The x-axis shows the episode number during training. The line shows the mean trajectory length for selected domains. 10 different seeds are used to compute the mean. Unlike Figure 1 and 2, the curves are not smoothed and contain state HER as well.