LEARNING OBJECT-CENTERED AUTOTELIC BEHAVIORS WITH GRAPH NEURAL NETWORKS

Ahmed Akakzia
Sorbonne Université
ahmed.akakzia@isir.upmc.fr

Olivier Sigaud
Sorbonne Université

ABSTRACT

Although humans live in an open-ended world and endlessly face new challenges, they do not have to learn from scratch each time they face the next one. Rather, they have access to a handful of previously learned skills, which they rapidly adapt to new situations. In artificial intelligence, autotelic agents — which are intrinsically motivated to represent and set their own goals — exhibit promising skill adaptation capabilities. However, these capabilities are highly constrained by their policy and goal space representations. In this paper, we propose to investigate the impact of these representations on the learning and transfer capabilities of autotelic agents. We study different implementations of autotelic agents using four types of Graph Neural Networks policy representations and two types of goal spaces, either geometric or predicate-based. By testing agents on unseen goals, we show that combining object-centered architectures that are expressive enough with semantic relational goals helps learning to reach more difficult goals. We also release our graph-based implementations to encourage further research in this direction.

1 INTRODUCTION

A central challenge in artificial intelligence (AI) consists in designing artificial agents capable of solving an unrestricted set of tasks in a continual and open-ended skill learning process. In principle, these processes should be domain-agnostic. Reinforcement learning (RL) seems to be an adequate paradigm to solve a single sequential decision problem from a reward signal (Sutton et al., 1999). Nevertheless, this signal is usually predetermined and highly grounded to its designer's aspirations. Thus, the extension of the RL framework to an open-ended sequences of unpredictable tasks raises difficult questions.

Recently, a promising line of research has been interested in the design of *autotelic agents*, borrowing older ideas from (Steels, 2004). These agents are intrinsically motivated to represent, set and pursue their own goals. Usually, they do not depend on any external reinforcement signal, since they autonomously reward themselves over the completion of their own goals. Autotelic agents are known to be open-ended learners. Through RL, they manage to acquire goal-directed behaviors which can transfer to domains sharing similar goal spaces. However, this transfer is deeply bound to their representational capabilities.

From that perspective, a key challenge consists in endowing autotelic agents with appropriate inductive biases to enhance their representational power. To enable efficient transfer, such biases should express a set of general and structured features. On the one hand, the design of the autotelic agents' goal spaces should leverage the power of structured semantic representations. Namely, recent works in AI (Akakzia et al., 2021; Alomari et al., 2017; Kulick et al., 2013; Tellex et al., 2011) introduced symbolic high-level object-centered representations to explicitly capture abstract spatial relations such as proximity and aboveness, where the latter is used to refer to the quality of being directly above. By contrast, other works use plain spatial target coordinates specific to each of the available objects (Colas et al., 2019; Li et al., 2019; Lanier et al., 2019).

On the other hand, although neural networks are flexible tools to learn latent representations, their raw usage is insufficient to capture disentangled representations from high-dimensional structured input. Recently, Graph Neural Networks (GNNs) have been introduced to implement relational inductive biases in neural networks. They mainly rely on shared networks to transfer features among the input components. Besides, they follow efficient computation

schemes: through their neighborhood aggregation and graph-level pooling schemes, they easily capture the existing relationships between nodes.

Contributions. In this paper, we provide a systematic study of the use of GNNs in autotelic learning within a multi-object manipulation domain. More specifically, we investigate 4 variants of GNNs: *full graph networks*, *interaction networks*, *relation networks* and *deep sets*. Furthermore, we consider two different types of goal spaces: 1) *semantic goals* based on binary predicates describing spatial relations between physical objects; 2) *continuous goals* corresponding to specific target positions for each object. Finally, we assess the transfer capabilities of the best performing GNN-based agents by introducing three sets of held-out semantic goals defining three different scenarios: 1) transfer to combinations of configurations (such as a stack and a pyramid); 2) transfer from goals based on pair-wise relations to goals based on triple-wise relations (pyramids); 3) transfer to higher order stacking of objects.

Our results show that

- Semantic goal spaces induce a higher level of abstraction than continuous goal spaces, enabling lighter GNN-based architecture to perform on par with the ones that use the whole computational scheme.
- Performing the edge update step is sufficient for good transfer to combinations of previously seen goals.
- Node updates promote the transfer from goals based on pair-wise relations to goals based on triple-wise relations, as information flow not only between pairs, but also between all the nodes.
- Relation networks outperform full graph and interaction networks in transferring to higher order stacks on objects.

These results suggest that coupling semantic goal spaces with sufficiently representative graph-based networks helps to learn more complex goals and yields better transfer capabilities. Finally, we release our implementations of the considered GNN-based architectures in multi-object manipulation domain to encourage further research in this direction[1]

2 Related work

This paper relies on several previous works from different areas of research within AI. Namely, we consider recent findings in automatic curriculum learning, semantic goal representations, graph neural networks and graph-based autotelic learning, and combinations of several of these aspects.

Automatic Curriculum Learning. Adaptability is a key characteristic enabling humans to display an exceptional capacity to learn (Elman, 1993) and works in AI attempted to leverage similar automatic curriculum learning (ACL) schemes in artificial agents (Portelas et al., 2020). Most of these approaches leverage forms of intrinsic motivations to power their exploration and learning progress (LP) (Bellemare et al., 2016; Achiam & Sastry, 2017; Nair et al., 2018; Burda et al., 2018; Pathak et al., 2019; Colas et al., 2019; Pong et al., 2019). In this paper, our agents borrow the LP-based curriculum learning algorithm introduced in Colas et al. (2019) when targeting continuous goals, but we show this in not necessary when targeting semantic goals.

Semantic Goal Representations. Studies in developmental psychology suggest that notions such as proximity, animacy and containment are innately grounded in the perceptual world of the infant (Mandler, 2012). Inspired by this line of thought, recent works in AI introduced symbolic high-level representations to explicitly capture abstract spatial relations (Tellex et al., 2011; Kulick et al., 2013; Alomari et al., 2017; Akakzia et al., 2021). We borrow the semantic goal representations used in Akakzia et al. (2021) and based on the predicates *close* and *above*. Such semantic representations are more abstract the classic goal-as-state representations, as they account for the underlying relations between objects independently of their perceived states, such as their geometric positions.

Graph Neural Networks. GNNs are powerful tools to implement strong inductive biases that focus on structured representations (Battaglia et al., 2018). At the price of more computations, they efficiently foster combinatorial generalization and improve sample efficiency over standard architectures in different machine learning domains (Gilmer et al., 2017; Scarselli et al., 2005; Zaheer et al., 2017; Li et al., 2019). GNNs parse the stream of input features into several objects, called *nodes*. They also capture the relational features between pairs of these objects which they store in the corresponding *edges*. They usually involve three computational schemes: 1) **Edge updates** using the initial features of the edge and both features of the nodes involved within that edge; 2) **Node updates** using the initial features

[1] https://github.com/akakzia/rlgraph$_2$.

of the node and the aggregated features of the edges that enter that nodes; 3) **Graph output** using an aggregation of either all the nodes or the edges features. The first two steps involve shared networks, which enable transfer between the different nodes and edges. Depending on the order and the nature of the computational steps, there exist many variants of GNNs. In this paper, we only consider 4 of these variants: *full graph networks* (Battaglia et al., 2018), *interaction networks* (Battaglia et al., 2016), *relation networks* (Santoro et al., 2017) and *deep sets* (Zaheer et al., 2017). In general, these variants are shown to outperform flat architectures when combined with object-centered representations. Details about the implementations of these variants are provided in Section 3.3.

Graph-based Autotelic RL. GNNs have been used to solve RL problems (Zambaldi et al., 2018; Li et al., 2019; Colas et al., 2020a; Akakzia et al., 2021). By contrast to Li et al. (2019); Colas et al. (2020a); Akakzia et al. (2021) — which explicitly associate a node to each object in an object manipulation domain — the approach in Zambaldi et al. (2018) attempts to solve the StarCraft II mini-games (Vinyals et al., 2017) without object-centered inductive bias. In the latter, the nodes do not correspond to specific objects, but rather to randomly scattered boxes of pixels. In this paper, we rather join the former group.

Structured Policies and Representations. Close to our work, (Bapst et al., 2019) study the combination of structured representations and graph-based policies and show that it outperforms setups that use less induced structure. On the one hand, like us, they consider both continuous and semantic settings. In the former, while they add a node for each target goal, we encode continuous goal features within the edges of our graphs. In the latter, while they use an additional conversion layer to feed the policy with the converted geometric features, we directly use binary semantic predicates as inputs to both our critics and actors. On the other hand, by contrast to their graph computation scheme which involves an encode-process-decode architecture (Battaglia et al., 2016), our graphs are simpler as they do not use any form of recurrence. In fact, through only one step of computation involving one edge update and one node update, inputs are converted into either actions (for the actor) or q-values (for the critic). Finally, in this paper, we consider many types of GNNs which use different computation schemes and we aim at assessing their transfer capabilities to previously held-out goals.

3 METHODS

In this section, we state the problem we address in this paper, then we introduce the object manipulation environment and the two goal spaces that we consider (Section 3.2). Finally, we present the graph-based implementations of our autotelic agents (Section 3.3)

3.1 PROBLEM STATEMENT

We address hard exploration problems where an agent is expected to learn a large diversity of complex behaviors. We cast the problem into the framework of goal-conditioned reinforcement learning (Colas et al., 2020b) and particularly consider autotelic agents which can represent, set and pursue their own goals. These agents learn from a sparse reward signal, i.e. they are only rewarded for reaching the goal they have set. More formally, these sparse reward autotelic agents are facing a rewardless Markov Decision Process MDP $= \{S, A, T\}$ where states $s \in S$ and actions $a \in A$ are continuous valued and the transition function $T : S \times A \rightarrow \Pi(S)$ defines the probability of reaching any state after performing an action from a state. Agents themselves are implemented as a goal-conditioned policy $\pi(a|s, g)$ and are rewarded with a function $r(s, g)$ which determines whether state s satisfies goal g. The key feature of autotelic agents is that they choose on which goal g to work at any moment. In this paper, goals are either semantic, i.e. they are represented as a set of binary predicates describing the features of the scene that matter for the agent's tasks, or continuous, i.e they directly correspond to a subset of the features perceived by the agent.

3.2 ENVIRONMENT AND GOAL SPACES

The Fetch Manipulate Environment. All agents studied in this paper evolve in the *Fetch Manipulate* domain from Akakzia et al. (2021), which is a variant of the standard *Fetch* domains (Plappert et al., 2018). We extend it to a 5-object setup: the agent is a 4-DoF robotic arm facing 5 colored objects on a table. It perceives features of its body and of the surrounding objects. These features include geometric positions, orientations and velocities.

Semantic versus continuous Goals. From the perceived features, agents using semantic goals build high-level binary representations that assert the presence (1) or absence (0) of the binary spatial relations *above* and *close* between objects. As the latter is symmetric (*close(A, B) = close(B, A)*), we only consider 10 combinations of objects for this

predicate. However, we consider all the 20 ordered pairs of objects for the *above* predicate. This yields semantic goal vectors of 30 dimensions. The resulting configuration space contains 2^{30} elements, among which ~ 75.000 are physically reachable. These semantic representations are inspired by the work of Mandler (2012) on a minimal set of spatial primitives children seem to be born with, or to develop early in life. Initially empty, the set of discovered semantic goals gets gradually filled each time an agent encounters new configurations.

By contrast with semantic goals, continuous goals directly use the perceived features, i.e. goals correspond to precise target positions for each available object. To succeed, agents have to place every object in its corresponding target position, see Figure 1 for an illustration. These goal spaces are used in many works attempting to solve multi-object manipulation problems (Colas et al., 2019; Li et al., 2019; Lanier et al., 2019).

Figure 1: Illustration of objects and targets.

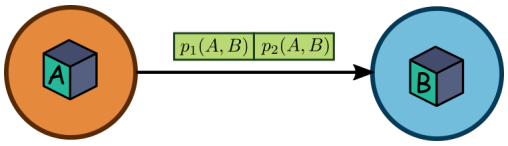
Autotelic learning. All studied agents autonomously select and attempt to master goals from the set of discovered goals. Agents using semantic goals simply reward themselves for each object for which all the predicates involving that object are verified. An episode ends successfully if all predicates about all objects are verified before a time limit. At the beginning of an episode, the blocks are procedurally placed on the table so that they are never initially stacked.

By contrast, the autotelic learning process of agents using continuous goals is more involved. First, we assume that these agents are initially aware that they can construct stacks using the available objects, and that the maximum number of objects stacked corresponds to the number of available objects. Second, at the beginning of each episode, these agents autonomously select how many objects they want to stack (from 0 up to 5 in this paper). Accordingly, target positions are generated for each object. These agents reward themselves for each object placed correctly within a range of its corresponding target position. An episode ends successfully if all objects are placed correctly before a time limit. To further accelerate the learning process, we consider biased initializations as part of a way to adapt the difficulty of the task to the learner's skills: at the beginning of each episode, and with a probability of 0.2, blocks are arranged into a stack of up to 5 objects. We call this non-trivial scene reset. To stabilize the learning process, we use an automatic LP-based curriculum (Colas et al., 2019): based on their learning progress estimations, agents can choose to target goals with no stacks, a stack of 2, 3, 4 or 5 objects where all target positions that are not involved in stacks are automatically generated directly on the table. As shown in Appendix B.1, calling upon this additional ACL process is necessary for learning to work when using continuous goals.

3.3 GRAPH-BASED AUTOTELIC LEARNING

In this section, we describe the implementation of the intrinsically motivated goal-conditioned RL module using GNNs. This module is powered by the Soft-Actor Critic algorithm (SAC) (Haarnoja et al., 2018) where both the critic and the policy networks are GNNs. We use the Multi-criteria Hindsight Experience Replay algorithm (MC-HER) to facilitate transfer between goals (Lanier et al., 2019). MC-HER extends the Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) strategy to multi-object scenarios, enabling further transfer between partial features of the goal vector.

3.3.1 GRAPH STRUCTURE



(a) Input edge features as target predicates where p_1 =close and p_2 =above.

(b) Input edge features as target geometric positions.

Figure 2: Illustration of a single directed edge for semantic goals (a) and continuous goals (b).

All our agents use a fully connected graph structure: every object corresponds to a node, and all nodes are connected. First, each node holds the features of a particular object in the scene. Second, each edge linking a source and a recipient node holds partial features of the goal. As illustrated in Figure 2, for semantic goals, these features correspond to the predicates that involve both the source and the recipient node, while for continuous goals, they correspond to the target position of the block corresponding to the source node. Finally, the global features correspond either to the agent's

body state (in the case of the policy) or to a concatenation of the agent's body state and the action (in the case of the critic). We respectively denote the node features, edges features and global features with X, E and U.

3.3.2 GRAPH COMPUTATIONS

Although all our agents rely on the same graph structure, they use different computation schemes. In this paper, we focus on four particular types of GNNs: full graph networks (GN), interaction networks (IN), relation networks (RN) and deep sets (DS). Figure 3 illustrates the different computation steps for each architecture.

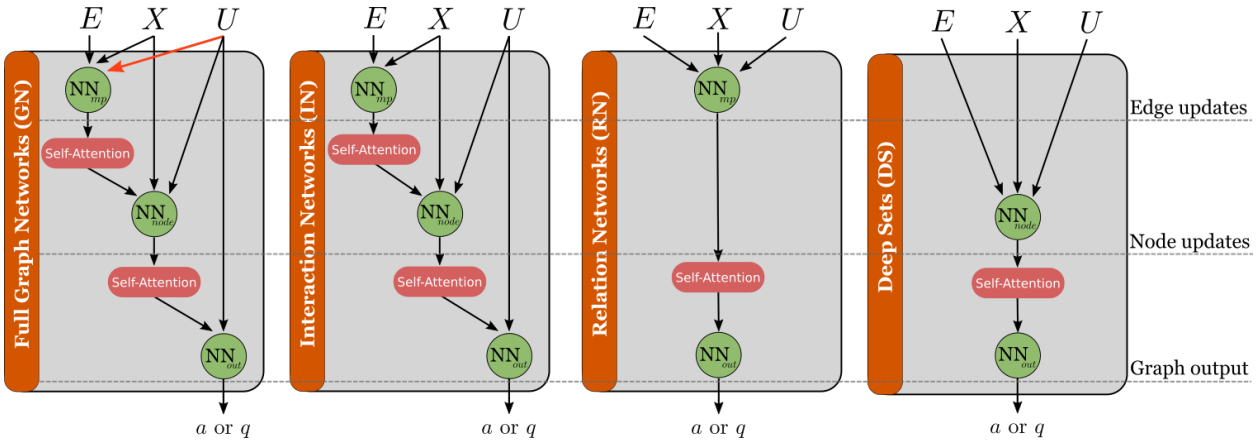


Figure 3: Illustration of the different computational schemes for (from left to right) GN, IN, RN and DS. E, X and U respectively correspond to the edge features, node features and global features. Note that GN uses U to update edges features (red arrow), while IN does not and RN only updates edges features, while DS only updates nodes features.

Full Graph Network (GN). As its name suggests, this architecture uses the whole computation scheme within a standard graph network block. See Figure 3 for an illustration. First, an *edge update* step is performed. A shared network NN_{mp} is used to compute the update features of each edge. It takes as input the concatenated input features of each edge (goal features), the involved source and recipient nodes (object features) and the global features. Second, a *node update* step is performed for each node using a second shared network NN_{node}. It takes as input the concatenated input features of the considered node, the global features and an aggregation of the updated features of the incoming edges. Third, the *graph output* step is performed, where the updated features of the nodes are pooled, concatenated with the global features and fed to a readout network NN_{out}. The output quantity corresponds to either the action (in the case of the actor) or the q-value (in the case of the critic). In this paper, we use self-attention to compute the weighing scores used in all the aggregation steps (Vaswani et al., 2017; Veličković et al., 2017).

Interaction Network (IN). This architecture resembles the one described in the GN architecture. The only difference is that, during the edge update step, the global features are not used as inputs to the shared network NN_{mp}.

Relation Network (RN). This architecture entirely bypasses the node update step. It only performs the edge update step using the shared network NN_{mp}, which takes as inputs the initial node, edge and global features. The output vector is aggregated using a self-attention module, then fed to a readout network NN_{out}.

Deep Sets (DS). This architecture entirely bypasses the edge update step. It only performs node updates using the shared network NN_{node}. The latter takes as input the node, edge and global features, outputs a vector is later fed to a self-attention module to compute attention scores. Finally, the aggregated vector is fed to a readout network NN_{out}.

3.3.3 PSEUDO CODE

The autotelic learning mechanisms with respectively semantic and continuous goals are presented in Algorithms 1 and 2. Algorithms 3 and 4 further describe how goals are sampled and updated in Algorithm 1.

Algorithm 1 Learning Semantic Goals

Require Env E, number of trajectories per step n, replay function R_{mcher}
Initialize policy Π, Uniform goal sampler $\mathcal{G}^s_{unif}$, buffer B.
$L_{discovered}$ = []
loop
  $L_{goals} \leftarrow$ SAMPLE GOALS($L_{discovered}$, n)
  $L_{trajectories} \leftarrow E$.rollout($\Pi$, g)
  $L_{discovered} \leftarrow$ UPDATE GOALS($L_{trajectories}$)
  B.update($L_{trajectories}$)
  $\mathcal{D}_{train} \leftarrow B$.sample($R_{mcher}$)
  Π.update($\mathcal{D}_{train}$)
return Π

Algorithm 2 Learning Continuous Goals

Require Env E, Goal classes C_g, number of trajectories per step n, replay function R_{mcher}
Initialize policy Π, LP-based goal sampler $\mathcal{G}^s_{LP}$, buffer B.
loop
  $L_{classes} \leftarrow \mathcal{G}^s_{LP}$.sample_classes($C_g$, n)
  $L_{goals} \leftarrow generate\_positions(L_{classes})$
  $L_{trajectories} \leftarrow E$.rollout($g$)
  $\mathcal{G}^s_{LP}$.update($L_{trajectories}$)
  B.update($L_{trajectories}$)
  $\mathcal{D}_{train} \leftarrow B$.sample($R_{mcher}$)
  Π.update($\mathcal{D}_{train}$)
return Π

Algorithm 3 SAMPLE GOALS

Require discovered goals list $L_{discovered}$, number of goals to sample n,
$L_{samples}$ = []
for i in [1, .., n] **do**
  if $L_{discovered}$ is empty **then**
    $L_{samples}$.append(zeros)
  else
    $g \leftarrow L_{discovered}$.sample_uniform()
return $L_{samples}$

Algorithm 4 UPDATE GOALS

Require Trajectory list L_τ, discovered goals list $L_{discovered}$ $\mathcal{G}^s_{LP}$, buffer B.
for τ in L_τ **do**
  $(s, a, s', g_{achieved}, g_{desired}) \leftarrow \tau$
  Last_$g_{achieved} \leftarrow g_{achieved}[-1]$
  if Last_$g_{achieved}$ not in $L_{discovered}$ **then**
    $L_{discovered}$.append(Last_$g_{achieved}$)
return $L_{discovered}$

4 EXPERIMENTS AND RESULTS

We first describe the experimental setup used in this paper. Then, we present the results obtained when training autotelic agents with semantic and continuous goals. Finally, we assess the transfer capabilities on the different architectures with three different scenarios. Additional studies and ablations are provided in the appendices.

4.1 EXPERIMENTAL SETUP

We train 4 graph-based autotelic agents in the *Fetch Manipulate* domain with 5 objects using the graph architectures described in Section 3.3. We consider both the semantic and continuous goal spaces introduced in Section 3.2.

Evaluation Classes. To evaluate the agents, we define several evaluation classes for both semantic and continuous goals. First, for semantic goals, we consider classes of configurations where exactly i pairs of blocks are close (C_i), configurations containing stacks of size i (S_i), configurations containing pyramids of size 3 (P_3) and combinations of these. These classes are disjoint and their union does not cover the entire semantic configuration space, but they are representative enough and they enable fair comparisons between the agents. Second, for continuous goals, we consider classes of configurations where there are no stacks and where there is a stack of size i ($\widetilde{S}_i$, where the symbol $\sim$ is for continuous).

Evaluation Metrics. Evaluations are performed each 50 cycles. During one cycle, the agents perform 2 rollouts of 200 timesteps with 2 goals sampled autonomously. At test time, the per-class performance of the agent is computed on 24 goals of each evaluation class (264 semantic goals and 120 continuous goals). The measure of the agent's global success rate (SR) is the average of all the per-class successes. Testing is conducted offline and with deterministic policies.

Baseline. For both semantic and continuous goals, we consider a flat baseline, where all the perceived features are concatenated and directly fed to the neural networks. We call Semantic-Flat (S-FLAT) and Continuous-Flat (C-FLAT) the flat baseline using respectively semantic and continuous goals.

Networks Capacity. Independently of their computational scheme, we make sure all the agents have the same networks capacity in terms of number of parameters to be optimized. As full graph networks use the highest number of parameters in principle, we provide the other agents with sufficient budget to match them. Concretely, we add an additional node updater and edge updater for respectively deep sets and relation networks architecture, and we make the flat networks sufficiently deep.

Transfer Scenarios. To assert the transfer capabilities of the different agents, we investigate how GNN-based agents with semantic goal spaces to goal configurations used as test goals though they have never trained on them before. To this end, we consider three different scenarios: **1) Transfer to combinations of constructions**, where agents are prevented from training on any goal including combinations of known constructions. The corresponding set of test goals includes the classes $S_2\&S_2$, $P_3\&S_2$ and $S_2\&S_3$; **2) Transfer to pyramids**, where agents are prevented from training on any goal including pyramids. The underlying set of test goals includes the classes P_3, and $P_3\&S_2$; **3) Transfer to higher stacks**, where agents are prevented from training on goals including stacks of size 3 and more. The set of corresponding test goals includes S_3. See Table 1 for details.

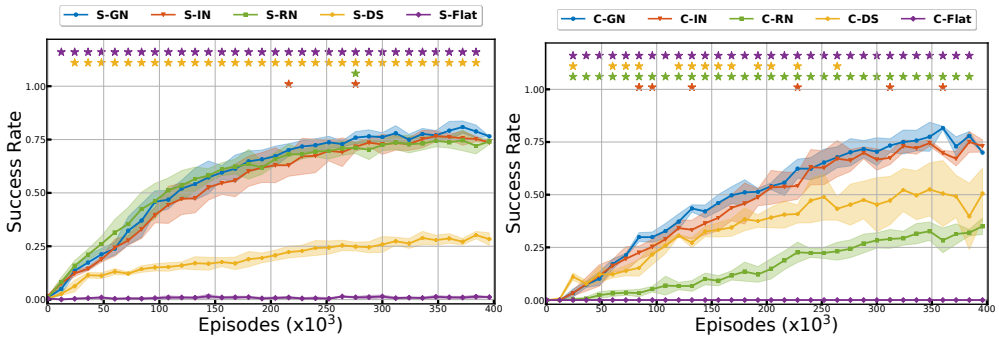
Table 1: Testing classes and their sizes.

Scenario-Class	Size
1 - S_2 & S_2	60
1 - S_2 & S_3	120
(1, 2) - P_3 & S_2	60
2 - P_3	30
3 - S_3	60

In practice, we make sure none of the testing goals are sampled during training by simply preventing any episode where a test goal was encountered (at any time step) from getting stored in the replay buffer. This also prevents HER 's future strategy from selecting these goals during replay. The set of training goals for each scenario may include all the other possibly encountered configurations.

4.2 GLOBAL PERFORMANCE METRICS

In this section, we study the global performance of the different graph-based autotelic agents. Figure 4 presents the average SR across evaluation classes for both semantic goals (Figure 4a) and continuous goals (Figure 4b).



(a) (b)

Figure 4: Global SR across training episodes with (a) Semantic (S) goals and (b) Continuous (C) goals for the considered agents. Mean $\pm$ standard deviations are computed over 5 seeds. Stars highlight statistical differences w.r.t S-GN agents (Welch's t-test with null hypothesis $\mathcal{H}_0$: no difference in the means, $\alpha = 0.05$).

Semantic Goals. The S-FLAT agents are not able to learn to reach any semantic configuration as their global SR does not increase during training (Figure 4a, purple curve). This is not surprising since simple MLP networks that take as input high dimensional concatenations of multiple objects struggle in disentangling the learned representations. By contrast, all the GNN-based agents are able to increase their global SR, this is consistent with previous results showing that object-centered graph-based architectures are better suited for multi-object manipulation domains (Li et al., 2019). On the one hand, the global SR of S-DS agents gets stuck at around 25% (Figure 4a, orange curve). This result highlights the importance of the edge update step, as it allows to focus on pairwise relations embedded within the semantic relational predicates which are used as input features to the edges. On the other hand, S-GN, S-IN and S-RN agents have similar performance across training episodes (Figure 4a, blue, red and green curves), and show statistical differences only rarely (see stars on Figure 4a). This result suggests that, when dealing with semantic relational goals, *the edge update step is crucial*. In fact, S-RN agents — which have a lighter computational scheme but exclusively perform pairwise edge updates — manage to perfectly catch with the more heavy architecture S-GN and S-IN.

Continuous Goals. Similar to semantic goals, the C-FLAT agents fail to learn any interesting behavior. However, interestingly, only GNN-based architectures that use the full computational scheme — that is, C-GN and C-IN — have the best performance. On the one hand, by contrast to the semantic goals setup, C-RN agents get stuck at around 30% of the maximum global SR (Figure 4b, green curve). This suggests that the edge update step is not sufficient to capture interesting features when geometric target goals are encoded within the input edge features. On the other hand, C-DS agents perform better than their C-RN counterparts, with an average global SR of 50% (Figure 4b, orange curve). This is probably due to the fact that the node update step — which is conducted by C-DS but not C-RN — enables information about every target geometric goal to flow to every node in the graph, which helps agents increase their performance. However, C-DS agents show higher variance compared to all the other GNN-based agents. This instability is probably explained by the role of the edge update step — which is bypassed by C-DS — in disentangling useful pairwise features that help stabilize the learning. This is made even more likely as agents that perform the whole computational scheme provide the best results and the least instabilities (Figure 4b, blue and red curves).

4.2.1 Per Class Performance Metrics

The global performance metrics in Section 4.2 show that the average SR across evaluation classes gets stuck at around 75% for both semantic and continuous agents. To investigate this, we zoom on the per class performance metrics.

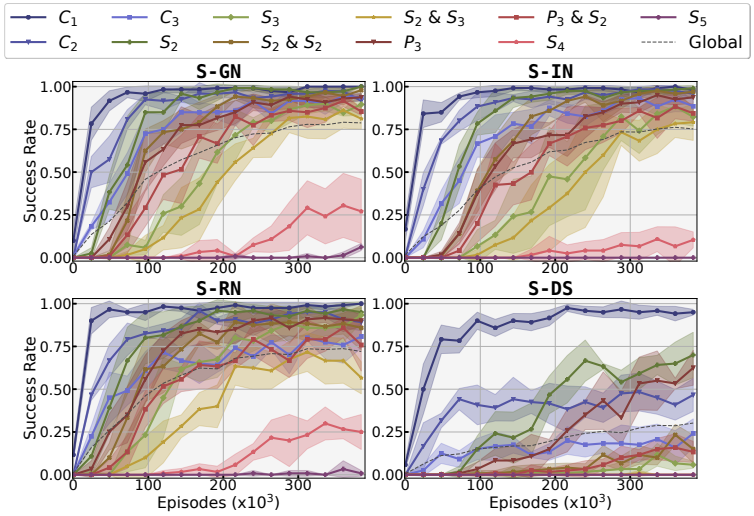


Figure 5: Local SR for each class across training episodes with continuous goals. Mean ± standard deviations are computed over 5 seeds.

Semantic Goals. Figure 5 shows the per-class performance of S-GN, S-IN, S-RN and S-DS. First, and as the global performance metrics suggest (Section 4.2), S-GN, S-IN and S-RN show very similar local performance. They are all able to master all the classes (with at least 65% of SR) except for S_4 and S_5. This failure occurs because the learned policies are sub-optimal. In fact, when rewarding themselves for each object placed correctly, the critics would most likely be *greedy*: incremental rewards should come fast, even if this means not constructing stacks in the trivial order (from base upwards). As a result, agents would start by constructing the upper part of a stack, then placing it on the base object. This is not a problem for S_3 since robotic arms can pick and place a stack of two blocks. However, in S_4, it is impossible to pick and place a stack of three blocks. See Figure 6 for an illustrative example. Second, the S-DS agents struggle with classes that involve many constraints to be satisfied. This is because they lack enough representational power to disentangle pairwise relations between objects.

Continuous Goals. Figure 7 shows the per-class performance of C-GN, C-IN and C-DS. All agents first master the easy classes, before moving up to the more diffficul ones. This results from these agents leveraging automatic curriculum learning, using their LP estimation as a proxy to choose goals that are at an affordable level of complexity. However, as opposed to semantic goals, there is less interference between classes and transfer is poorer (per-class SR increases sequentially). On the one hand, C-DS agents show a lot of instabilities beyond the $\widetilde{S}_2$ class. This further supports the idea that the node update step alone in deep sets does not provide enough representational power. On the other hand, both C-GN and C-IN manage to reach goals in all the evaluation classes, from no stacks at all to stacks of 5 objects. However, they are both unable to maximize their per-class performance. This suggests that learning policies

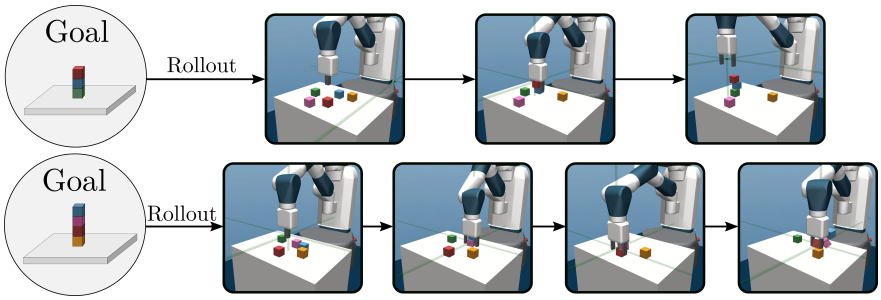


Figure 6: Example of sub-optimal behavior with semantic goals when targeting a goal in S_3 (up) and in S_4 (down). The agent tries to pick and place a stack of three objects and fails (down).

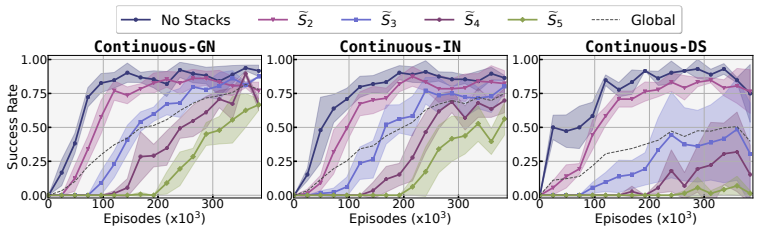


Figure 7: Local SR for each class across training episodes with continuous goals. Mean $\pm$ standard deviations are computed over 5 seeds.

that can achieve all evaluation classes at the same time with continuous goals is difficult and requires more training budget.

4.2.2 TRANSFER CAPABILITIES

To investigate the transfer capabilities of the GNN-based agents, we consider the best performing architectures with semantic goal spaces: S-GN, S-IN and S-RN (See Section 4.2). Tables 2 and 3 respectively show the global and the per class performance metrics for the considered agents. The values presented in these tables correspond to the evaluation of the training policies on the held-out goals once the training is stabilized.

Table 2: Global SR metrics, averaged over 5 seeds.

Agents	Scenario 1	Scenario 2	Scenario 3
S-GN	$\mathbf{0.93 \pm 0.04}$	0.78 ± 0.09	0.28 ± 0.14
S-IN	0.89 ± 0.01	$\mathbf{0.82 \pm 0.09}$	0.47 ± 0.13
S-RN	0.89 ± 0.02	0.68 ± 0.12	$\mathbf{0.64 \pm 0.10}$

Table 3: Per class SR metrics, averaged over 5 seeds.

Scenario-Class	S-GN	S-IN	S-RN
1 - S_2 & S_2	$\mathbf{0.97 \pm 0.02}$	0.92 ± 0.04	0.96 ± 0.03
1 - S_2 & S_3	$\mathbf{0.89 \pm 0.04}$	0.87 ± 0.01	0.82 ± 0.07
1 - P_3 & S_2	$\mathbf{0.93 \pm 0.05}$	0.88 ± 0.04	0.90 ± 0.05
2 - P_3	0.80 ± 0.09	$\mathbf{0.85 \pm 0.10}$	0.67 ± 0.14
2 - P_3 & S_2	0.75 ± 0.12	$\mathbf{0.80 \pm 0.10}$	0.68 ± 0.11
3 - S_3	0.28 ± 0.14	0.47 ± 0.13	$\mathbf{0.64 \pm 0.10}$

Transfer Scenario 1. All the considered agents are good at transferring to combinations of constructions that they have encountered separately during training (Column 2 of Table 3). This probably results from both the representational power of GNNs and the self-attention aggregation schemes during both edge and node updates. On the one hand, all these graph-based architectures are permutation-invariant. On the other hand, for a goal within one of the combination classes, each construction is independent from the other. Consequently, agents would attend to one of them, accomplish it, and then focus on the other, as they learned to construct them separately during training.

Transfer Scenario 2. During training within this scenario, agents never encounter nor train on configurations involving pyramids. In other words, an object is never placed simultaneously on top of two other objects that are close to each other. First, we compare S-RN to both S-GN and S-IN. S-RN has lower overall transfer capabilities in this scenario compared to S-GN and S-RN (Column 1 of Table 2). They struggle in transferring to held-out goals from both P_3 and P_3&S_2 classes (Table 3). On the one hand, S-RN agents bypass the node update step which aggregates, for a particular node, the flowing information from the other nodes. Consequently, they tend to focus more on pairwise

relations and less on the global relations of a particular configuration. On the other hand, a pyramid involves three objects. In fact, to be able to put an object i on top of j and k, the latter two need to be close to each other. As a result, exclusive edge updates without allowing information to flow between the different nodes is insufficient. Second, we compare S-GN to S-IN. As the former concatenates the global features to the input features of the edge update, the size of its corresponding network would be greater. This would probably lead to overfitting on the training data and transferring less to new situations.

Transfer Scenario 3. During training, agents never encountered a configuration involving a stack of 3 objects. Column 3 of Table 2 shows that S-RN outperforms both S-GN and S-IN in this scenario. This can be explained in a threefold fashion. First, we defined the *above* predicate as being *directly above*. Hence, a goal from the class S_3 would involve only two above predicates of the form *above*(i, j) and *above*(j, k) — which are fed to two different edges — to be true. Second, S-RN agents focus on pairwise relations that exist within edges. In other words, if they have learned to stack two blocks, they are capable to sequentially perform two independent stacks of two. Third, a stack of three objects can be constructed independently from the order of the stacks (i.e. first B above C then A above B or first A above B then A/B on C). By contrast S-GN and S-IN tend to transfer less as they overfit more on the training data due to their heavier computational scheme.

5 Conclusion and Future Work

In this paper, we studied several GNN-based goal-conditioned architectures for both the policy and critic in multi-object manipulation domains. More specifically, we considered four different computational schemes: full graph networks, interaction networks, relation networks and deep sets. We evaluated our agents using two different goal space structures: 1) continuous geometric goal spaces corresponding to per-object target positions; 2) semantic relational goal spaces based on the binary predicates close and above. Finally, we studied the transfer capabilities in three different scenarios by assessing the agents performance on different sets of held-out semantic goals.

Semantic and Continuous Goal Spaces. Autotelic agents benefit from the abstract structured representation within semantic goal spaces. In fact, our results show that, when dealing with continuous goals, full graph and interaction networks, which adapt the full graph computational scheme, are the best performing agents. For this particular type of goal space, information about all the objects needs to flow to each node in order to learn more complex goals. By contrast, with semantic goal spaces, performing the edge update step is sufficient to capture useful pair-wise relations between objects. In fact, relation networks perform in par with full graph and interaction networks. Finally, additional ingredients such as non-trivial scene resets and LP-based goal selection were necessary for agents to learn complex continuous goals. However, their counterparts that use semantic goals are more flexible as they do not need these additional ingredients.

Transfer Capabilities with Semantic Goals. Our transfer study suggests three main results. First, full graph, interaction and relation networks are all able to transfer to combinations of previously seen goals. Second, relation networks struggle in adapting to new goals that require reasoning about triplets of objects, such as building pyramids. These agents bypass the node update step, which seems to be crucial to pool information about all the nodes in the graph. Finally, relation networks outperform the other GNN-based architecture in transferring to goals of higher stacks. In fact, their light computational scheme enables them to overfit less on the training data. Consequently, they are more flexible in sequentially combining pair-wise skills.

This study helps understand the impact of key design choices towards open-ended autotelic agents capable of efficient transfer between abstract goals. However, the agents studied here only leverage their physical interactions with the environment. This does not account for the extraordinary human capacities to learn from social interactions (Vygotsky, 1978; Bruner, 1973; Tomasello, 2009). We believe adding social learning mechanisms as suggested in (Sigaud et al., 2021) is a promising line of research towards more capable open-ended agents.

Acknowledgments

This work was performed using HPC resources from GENCI-IDRIS (Grant 2020-A0091011875).

References

Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding language to autonomously-acquired skills via goal generation. In *ICLR 2021*, 2021.

Muhannad Alomari, Paul Duckworth, David C. Hogg, and Anthony G. Cohn. Natural language acquisition and grounding for embodied robotic systems. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. *arXiv preprint arXiv:1707.01495*, 2017.

Victor Bapst, Alvaro Sanchez-Gonzalez, Carl Doersch, Kimberly Stachenfeld, Pushmeet Kohli, Peter Battaglia, and Jessica B. Hamrick. Structured agents for physical construction. In *International Conference on Machine Learning*, pp. 464–474. PMLR, 2019.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

Jerome S. Bruner. Organization of early skilled action. *Child development*, pp. 1–11, 1973.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Cédric Colas, Pierre Fournier, Mohamed Chetouani, Olivier Sigaud, and Pierre-Yves Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *International conference on machine learning*, pp. 1331–1340. PMLR, 2019.

Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Ford Dominey, and Pierre-Yves Oudeyer. Language as a cognitive tool to imagine goals in curiosity-driven exploration. *arXiv preprint arXiv:2002.09253*, 2020a.

Cédric Colas, Tristan Karch, Olivier Sigaud, and Pierre-Yves Oudeyer. Intrinsically motivated goal-conditioned reinforcement learning: a short survey. *arXiv preprint arXiv:2012.09830*, 2020b.

Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1): 71–99, 1993.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Johannes Kulick, Marc Toussaint, Tobias Lang, and Manuel Lopes. Active learning for teaching a robot grounded relational symbols. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

John B Lanier, Stephen McAleer, and Pierre Baldi. Curiosity-driven multi-criteria hindsight experience replay. *arXiv preprint arXiv:1906.03710*, 2019.

Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. *arXiv preprint arXiv:1912.11032*, 2019.

Jean M Mandler. On the spatial foundations of the conceptual system and its enrichment. *Cognitive science*, 36(3): 421–451, 2012.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *arXiv preprint arXiv:1807.04742*, 2018.

Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pp. 5062–5071. PMLR, 2019.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep RL: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pp. 666–672. IEEE, 2005.

Olivier Sigaud, Hugo Caselles-Dupré, Cédric Colas, Ahmed Akakzia, Pierre-Yves Oudeyer, and Mohamed Chetouani. Towards teachable autonomous agents. *arXiv preprint arXiv:2105.11977*, 2021.

Luc Steels. The autotelic principle. In *Embodied artificial intelligence*, pp. 231–242. Springer, 2004.

Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Approaching the symbol grounding problem with probabilistic graphical models. *AI magazine*, 32(4):64–76, 2011.

Michael Tomasello. *Constructing a language*. Harvard university press, 2009.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

L. S. Vygotsky. Tool and Symbol in Child Development. In *Mind in Society*, chapter Tool and Symbol in Child Development, pp. 19–30. Harvard University Press, 1978. ISBN 0674576292. doi: 10.2307/j.ctvjf9vz4.6.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

A APPENDIX

A.1 IMPLEMENTATION DETAILS

In this part, we present details necessary to reproduce our results. We further open-source our code at https://github.com/akakzia/rlgraph$_2$.

GNN-*based networks.* Our four graph-based architectures use at most two shared networks, NN_{edge} and NN_{node}, respectively for computing updated edge features and node features. Both are 1-hidden-layer networks of hidden size 256. Taking the output dimension to be equal to $3\times$ the input dimension for the shared networks showed better results. All networks use ReLU activation and the Xavier initialization. For edge-wise and node-wise aggregation, we use a one-headed self-attention module. Finally, to produce the output, all architecture use a readout network NN_{out}. The latter is also a 1-hidden-layer network of hidden size 256. For optimization, we use Adam with learning rates 10^{-3}. The list of hyperparameters is provided in Table 4.

Parallel implementation of SAC-HER. All our experiments are based on a *Message Passing Interface* (Dalcin et al., 2011) to exploit multiple processors. Each of the 24 parallel workers maintains its own replay buffer of size 10^6 and performs its own updates. To synchronize experience between different workers, updates are summed over the 24 actors and the updated actor and critic networks are broadcast to all workers. Each worker alternates between 2 data collection episodes and 30 updates with batch size 256. To form an epoch, this cycle is repeated 50 times and followed by the offline evaluation of the agent.

Table 4: Hyperparameters used in this paper.

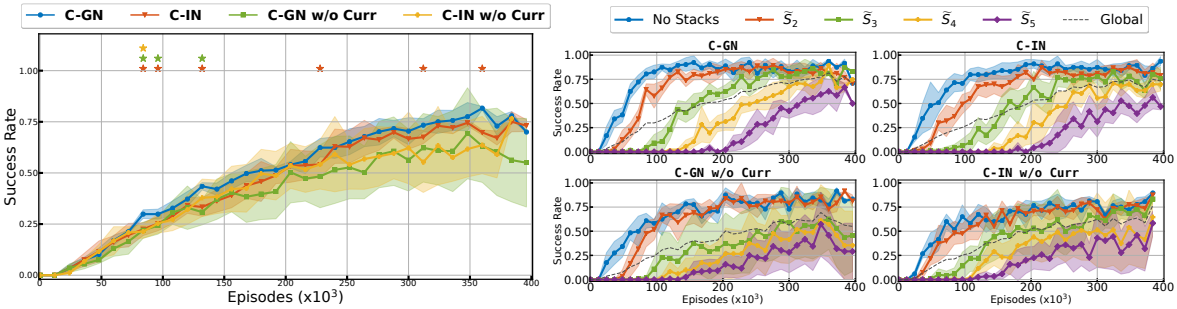
Hyperparam.	Description	Values.
nb_mpis	Number of workers	24
nb_cycles	Number of repeated cycles per epoch	50
$nb\_rollouts\_per\_mpi$	Number of rollouts per worker	2
$rollouts\_length$	Number of episode steps per rollout	200
$nb\_updates$	Number of updates per cycle	30
$replay\_strategy$	HER replay strategy	$future$
k_replay	Ratio of HER data to data from normal experience	4
$batch\_size$	Size of the batch during updates	256
γ	Discount factor to model uncertainty about future decisions	0.99
τ	Polyak coefficient for target critics smoothing	0.95
lr_actor	Actor learning rate	10^{-3}
lr_critic	Critic learning rate	10^{-3}
α	Entropy coefficient used in SAC	0.2
$biased\_init$	Probability of following non-trivial scene reset scheme	0.2
$self\_eval\_curriculum$	Probability to perform self evaluations	0.1
$curriculum\_queue\_length$	Window over which LP estimations are made	1000

B ADDITIONAL RESULTS

In this section, we present additional results which complement the ones presented in the main paper. More specifically, we study the relative importance of curriculum learning when using continuous goals (Appendix B.1) and of self-attention when using semantic goals (Appendix B.2).

B.1 CURRICULUM ABLATION

To study the relative importance of the LP-based curriculum learning mechanism used with continuous goals, we introduce ablations of C-GN and C-IN which uniformly sample a class of goals without any particular prioritization. We only consider architectures based on GN and IN in this ablation study since they show the best results in Section 4.2. Figure 8 presents the global performance metrics for C-GN, C-IN and their ablation counterparts. Autotelic agents using continuous goals but no curriculum clearly show an increased variance in their global performance. Figure 8 zooms on the local performance on each class for the considered agents. Compared to C-GN and C-IN, the shaded areas in the ablations are larger, suggesting that the learning process of the latter agents is not stable. Precisely, this



(a) Global SR. Stars highlight statistical differences w.r.t C-GN agents.

(b) Per class SR.

Figure 8: Performance metrics for C-GN, C-IN and their curriculum ablations. Mean ± standard deviations are computed over 5 seeds.

is true in stacks of size 3 or higher. In fact, ablations face catastrophic forgetting as they engage with harder goals. The curriculum learning mechanism helps stabilize the learning process by focusing on goals of moderate level of complexity, including the ones that the agents are likely to forget during training. Note that this issue is specific to continuous goals, which shows that they are not well suited to transfer between different goals.

B.2 Self-attention ablation

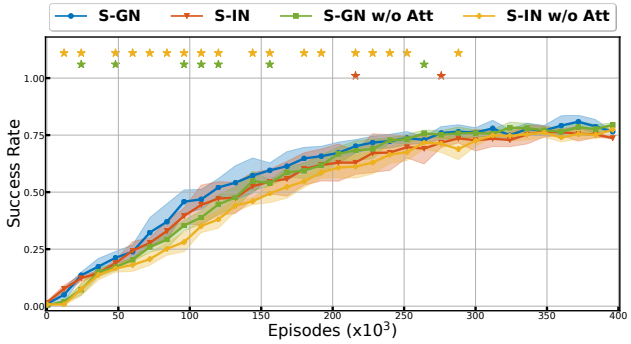
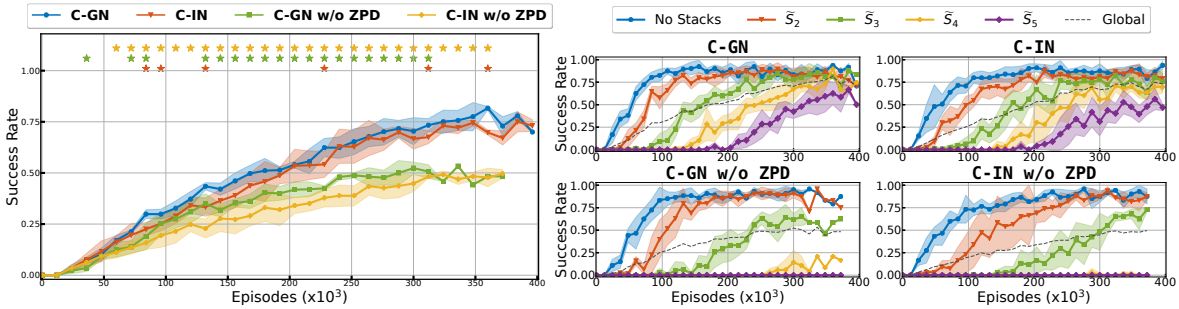


Figure 9: Global SR across training episodes for S-GN, S-IN and their self-attention ablations counterparts. Mean ± standard deviations are computed over 5 seeds. Stars highlight statistical differences w.r.t S-GN agents (Welch's t-test with null hypothesis $\mathcal{H}_0$: no difference in the means, $\alpha = 0.05$.

We propose to remove the self-attention aggregation schemes from S-GN and S-IN,– the two best performing agents,– and introduce the corresponding ablations which use an unweighted sum when performing the pooling over edges or nodes. Figure 9 presents the global SR for these agents across training episodes. The differences only appear at the beginning of training. In fact, the global performance metrics in the ablations increases slower than their corresponding full-versions (blue vs green; red vs orange). However, all agents seem to behave similarly by the end of training. This suggests that self-attention improves sample efficiency, yielding GNN-based agents that can faster capture actionable relational features within their graphs.

B.3 Non-trivial scene reset ablation

To assess the importance of the non-trivial scene reset scheme for continuous goals, we consider the C-GN and C-IN agents — the best performing GNN-based architectures so far — and remove the biased initialization scheme: blocks are placed without any initial stacks in the resulting ablations. Figure 10 shows performance metrics for these agents. The global SR of both ablations increases slower than that of C-GN and C-IN (Figure 10b). Besides, it gets stuck at around 50% of the maximal performance while their corresponding full versions manage to reach 75%. Zooming on the per-class performance metrics shows the considerable decrease in the capability to reach complex goals when



(a) Global SR. Stars highlight statistical differences w.r.t C-GN agents.

(b) Per class SR.

Figure 10: Performance metrics for C-GN, C-IN and their ablations where non-trivial scene reset is removed. Mean $\pm$ standard deviations are computed over 5 seeds.

removing the non-trivial reset scheme (Figure 10b): the ablations struggle to transfer between easy goals ($\widetilde{S}_2$ and $\widetilde{S}_3$) and harder ones ($\widetilde{S}_4$ and $\widetilde{S}_5$).