Object-Oriented Curriculum Generation for Reinforcement Learning

Felipe Leno Da Silva
University of São Paulo
São Paulo, Brazil
f.leno@usp.br

Anna Helena Reali Costa
University of São Paulo
São Paulo, Brazil
anna.reali@usp.br

ABSTRACT

Autonomously learning a complex task takes a very long time for Reinforcement Learning (RL) agents. One way to learn faster is by dividing a complex task into several simple subtasks and organizing them into a *Curriculum* that guides Transfer Learning (TL) methods to reuse knowledge in a convenient sequence. However, previous works do not take into account the TL method to build specialized *Curricula*, leaving the burden of a careful subtask selection to a human. We here contribute novel procedures for: (i) dividing the target task into simpler ones under minimal human supervision; (ii) automatically generating *Curricula* based on *object-oriented* task descriptions; and (iii) using generated *Curricula* for reusing knowledge across tasks. Our experiments show that our proposal achieves a better performance using both manually given and generated subtasks when compared to the state-of-the-art technique in two different domains.

KEYWORDS

Reinforcement Learning; Transfer Learning; Curriculum Learning

ACM Reference Format:
Felipe Leno Da Silva and Anna Helena Reali Costa. 2018. Object-Oriented Curriculum Generation for Reinforcement Learning. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. Andre, S. Koenig (eds.), Stockholm, Sweden, July 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Although Reinforcement Learning (RL)[29] has been used for autonomous task training, learning how to deliver a good performance takes a very long time, especially in the challenging tasks for which autonomous agents are starting to be employed [17]. As the classical RL algorithms are not scalable enough to be directly applied to such difficult problems, a growing body of literature studies how to reuse past knowledge to accelerate the learning process [34].

More recently, inspired by the *Curriculum Learning* approach initially applied for Supervised Learning methods [2], *Curricula* started to be employed for RL agents. The main idea of *Curriculum Learning* is to decompose a hard learning task (target task) into several simple ones (source tasks). Then, the learning agent can master source tasks and reuse the gathered knowledge to solve a target task (hopefully) faster than directly learning in it. For some domains, learning in smaller tasks might also be more cost effective (e.g., a robot learning in a simulated environment and then transferring knowledge for actuating in the real world [10]), hence using the *Curriculum* may be beneficial even when the agent does not learn faster but needs fewer steps in the target task.

This paper deals with the challenge of autonomously building *Curricula* that are useful to the agent. We here generate a *Curriculum* taking into account the TL method to better reuse the gathered knowledge. The *Object-Oriented* representation [6] is used to both generate a *Curriculum* in the form of a graph and to reuse knowledge, profiting from the generalization provided by the task description. The main contributions of this paper are: (i) a procedure for automatic construction of the set of source tasks under minimal human supervision; (ii) a method for automatically generating *Curricula* by using an intuitively-given *Object-Oriented* representation; (iii) a procedure for using large *Curriculum* graphs through a biased *Random Walk* approach. Moreover, we perform experimental evaluations in two domains, including a challenging Robot Soccer simulation.

Our experiments show that our proposal generates useful *Curricula*, which achieve advantages over the ones generated by the state-of-the-art method [30]. Our source task generation procedure was also successful in building a set of source tasks that lead to a useful *Curriculum*. An earlier version of this paper proposed using the *object-oriented* representation to build *Curricula* [23], but here we improve the experimental evaluation and propose a novel method for using those *Curricula*.

The remainder of this paper is organized as follows: Section 2 presents the background knowledge required for understanding our work; Section 3 presents our proposal adapting the *Curriculum* generation procedure to make better use of the *Object-Oriented* description; then, Section 4 describes our source task generation procedure; Section 5 presents our experimental setup and evaluation; Section 6 relates our proposal with the most relevant works in the literature; finally, Section 7 concludes the paper and points towards future works.

2 BACKGROUND

We here present the related concepts in the RL, *Object-Oriented* RL, and *Curriculum Learning* areas.

2.1 Reinforcement Learning

A *Markov Decision Process* (MDP) is the most widely adopted model for RL problems. An MDP is composed of $\langle S, A, T, R \rangle$, where S is a (possibly infinite) set of environment states, A is a set of available actions, T is the transition function, and R is the reward function. At each perception-action cycle, the agent observes the current state s, chooses one action a, and receives one reward $r = R(s, a, s')$, $s' = T(s, a)$. Each executed step results in a tuple $\langle s, a, s', r \rangle$, which

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. Andre, S. Koenig (eds.),
, July 2018, Stockholm, Sweden.

is the only feedback the agent has to learn how to solve the task (i.e., maximize the long-term sum of expected rewards). Since in learning problems R and T are unknown, the agent has to explore the environment by (initially) choosing random actions, until gathering enough knowledge to induce a policy $\pi : S \rightarrow A$, that returns one action to be applied in each state. A possible way to learn in this setting is by applying the Q-Learning [39] or SARSA [28] algorithms, which iteratively update an estimate of action qualities for each state $Q : S \times A \rightarrow \mathbb{R}$. This estimate eventually converges to the optimal Q-function: $Q^*(s,a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, where γ is a discount factor. Q can be used to define an optimal policy $\pi^*(s) = \arg\max_a Q^*(s,a)$ (the task solution). However, learning Q takes very long even for simple tasks, and much effort has been devoted to accelerate the learning process.

2.2 Object-Oriented MDPs

The use of task descriptions that allow abstraction can help to accelerate the learning process. The *Object-Oriented* representation [6] enables generalization opportunities through an intuitively-given task description. In *Object-Oriented MDPs* (OO-MDP) the state space is abstracted through the description of a set of classes $C = \{C_1, \ldots, C_c\}$, where each class C_i is composed of a set of *attributes* denoted as $Att(C_i) = \{C_i.b_1, \ldots, C_i.b_b\}$. Each attribute b_j has a *domain*, $Dom(C_i.b_j)$, specifying the set of values this attribute can assume. $O = \{o_1, \ldots, o_o\}$ is the set of objects that exist in a particular environment, where each object o_i is an instance of one class $C_i = C(o_i)$, so that o_i is described by the set of attributes from its class, $o_i :: Att(C(o_i))$. Now, the MDP state is observed as the union of all object states, $s = \bigcup_{o \in O} o.state$, where an object state is the set of values assumed by each of its attributes at a given time, $o_i.state = \left(\prod_{b \in Att(C(o_i))} o_i.b\right)$. Note that the definition of *object* here is not exactly the same as in OO programming. For RL, there is no class hierarchy. Also, the equality of objects in the point of view of the agent is usually computed comparing if two objects belong to the same class and have the same attribute values.

The generalization provided by such task descriptions is especially helpful to *Transfer Learning* (TL) approaches [14], which aim at reusing previous knowledge to solve a task faster [34]. In the next section we describe *Curriculum Learning*, an emerging method to accelerate learning by the use of TL techniques.

2.3 Curriculum Learning for RL

Curriculum Learning typically decomposes a hard (target) task $\mathcal{T}_t$ into several easier (source) tasks $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$. If proper task sequence (*Curriculum*) and TL methods are available, the whole set of tasks may be solved faster than when directly learning $\mathcal{T}_t$, due to the combination of a quick acquisition of knowledge in easier tasks and knowledge reuse.

In Narvekar *et al.*'s description [18], the *Curriculum* is a sequence of tasks within a single domain $\mathcal{D}$. $\mathcal{D}$ has a set of *degrees of freedom* $\mathcal{F}$. Any possible *MDP* that belongs to $\mathcal{D}$ can be built by a generator given a set of values of $\mathcal{F}$, $\tau : \mathcal{D} \times \mathcal{F} \rightarrow \mathcal{T}$. The learning agent is assumed to have a simulator to freely learn in the simpler tasks, and a sequence of source tasks is given by a human[1]. The agent then learns for some time in each of the source tasks specified by the *Curriculum* before trying to solve target task $\mathcal{T}_t$. The steps to build and to use a *Curriculum* are summarized in Algorithm 1. Given the target task $\mathcal{T}_t$, a set of candidates source tasks $\boldsymbol{\mathcal{T}}$ is defined (manually defined in all works so far). Then, the *Curriculum* $\mathbb{C}$ is built or given by a human, specifying a sequence of tasks $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t), \mathcal{T}_i \in \boldsymbol{\mathcal{T}}_{\mathbb{C}}, \boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{T}}_{\mathbb{C}}$. All tasks are then solved in order and previous knowledge might be reused for each new task.

Algorithm 1 *Curriculum* generation and use

Require: target task $\mathcal{T}_t$.
1: $\boldsymbol{\mathcal{T}} \leftarrow createTasks(\mathcal{T}_t)$
2: $\mathbb{C} \leftarrow buildCurriculum(\boldsymbol{\mathcal{T}}, \mathcal{T}_t)$
3: $learn(\mathcal{T}_t, \mathbb{C})$ ▷ use $\mathbb{C}$ to learn $\mathcal{T}_t$

Svetlik *et al.* [30] propose building the *Curriculum* as a graph, rather than an ordered list. The main idea of their proposal is to build a graph of tasks according to a *transfer potential* metric, which estimates how much a source task would benefit the learning process of another. They calculate transfer potential as:

$$v(\mathcal{T}_i, \mathcal{T}_j) = \frac{|Q_{\mathcal{T}_i} \cap Q_{\mathcal{T}_j}|}{1 + |S_{\mathcal{T}_j}| - |S_{\mathcal{T}_i}|}, \quad (1)$$

where $v(\mathcal{T}_i, \mathcal{T}_j)$ defines the transfer potential between two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, $|Q_{\mathcal{T}_i} \cap Q_{\mathcal{T}_j}|$ is the number of Q-values those two tasks have in common[2], and $|S_{\mathcal{T}_i}|$ and $|S_{\mathcal{T}_j}|$ are, respectively, the size of the state spaces in $\mathcal{T}_i$ and $\mathcal{T}_j$. A *Curriculum* graph $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, where $\boldsymbol{\mathcal{V}}$ is a set of vertexes (tasks) and $\boldsymbol{\mathcal{E}}$ is the set of edges that define the task sequence, is generated by including all source tasks that have a transfer potential to $\mathcal{T}_t$ higher than a threshold parameter ϵ. Then, $\mathbb{C}$ is used for learning in Algorithm 2 (which fits as a *learn* function in line 3 of Alg 1). First, an unsolved task with indegree *zero* is selected (line 4) and used for training until a stopping criterion is met (line 5). Then, all the edges from the selected task are removed from the graph (line 6) and another untrained task is selected until the target task is solved (that will be the last one). Notice that multiple tasks with indegree zero may exist (line 4); hence more than one task sequence might be generated from the same *Curriculum* depending on the tie-breaking strategy.

Algorithm 2 *learn* in [30]

Require: *Curriculum* $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, target task $\mathcal{T}_t$.
1: $\boldsymbol{L} \leftarrow \emptyset$ ▷ Learned tasks
2: **while** $\boldsymbol{\mathcal{V}} - \boldsymbol{L} \neq \emptyset$ **do** ▷ While there are unsolved tasks
3: $\quad \boldsymbol{T} \leftarrow \boldsymbol{\mathcal{V}} - \boldsymbol{L}$
4: $\quad$ Select a task $\mathcal{T}_i \in \boldsymbol{T}$ with $indegree = 0$
5: $\quad$ Learn $\mathcal{T}_i$ reusing previous knowledge
6: $\quad$ Remove edges from $\mathcal{T}_i$ in $\boldsymbol{\mathcal{E}}$
7: $\quad \boldsymbol{L} \leftarrow \boldsymbol{L} \cup \mathcal{T}_i$

Notice that every task included in the *Curriculum* must be learned before proceeding to the target task. Therefore, if a big *Curriculum* is given, the agent will probably spend too much time learning

[1] Narvekar *et al.* [18] present several heuristic procedures to define a *Curriculum*, but domain-specific human knowledge is required for all of them.

[2] This computation is domain-specific and must be defined by the designer.

source tasks. Moreover, while previous works have shown that building *Curricula* may be beneficial to learning agents, they do not evaluate the advantage of generalization when transferring knowledge between *Curriculum* tasks. To the best of our knowledge the set of source tasks is manually given in all works so far, and an automated source task generation procedure was listed as one of the open problems for *Curriculum* approaches [30]. In the next Sections we describe our proposal to automatically generate a *Curriculum*, how to reuse knowledge taking advantage of a relational task description, and our source task generation procedure.

3 OBJECT-ORIENTED CURRICULUM GENERATION

As discussed in Section 2, OO-MDPs can be used to provide generalization opportunities. We here contribute a method to take advantage of this generalization to facilitate *Curriculum* construction and use.

Since all tasks within a *Curriculum* are in the same domain, we have a single set of classes for all tasks, but each task may have a different set of objects and initial state. Thus, for our purposes we define an *Object-Oriented* task $\mathcal{T}_i$ as:

$$\mathcal{T}_i = \langle \boldsymbol{C}, \boldsymbol{O}^{\mathcal{T}_i}, S_0^{\mathcal{T}_i}, A^{\mathcal{T}_i}, T^{\mathcal{T}_i}, R^{\mathcal{T}_i} \rangle \quad (2)$$

where $\boldsymbol{C}$ is the set of classes, $\boldsymbol{O}^{\mathcal{T}_i}$ is the set of objects in $\mathcal{T}_i$, $S_0^{\mathcal{T}_i} : S \rightarrow [0, 1]$ is the probability of task $\mathcal{T}_i$ starting in each state, $A^{\mathcal{T}_i}$ is the set of possible actions for $\mathcal{T}_i$, $T^{\mathcal{T}_i}$ is the transition function, and $R^{\mathcal{T}_i}$ is the reward function. Notice that we suppressed the *degrees of freedom* set from Narvekar's original definition. If attributes not related to objects are required for $\mathcal{T}_i$, an *environment* class can be created for adding those attributes, hence the definition of *degrees of freedom* is no longer necessary. As in [30], we consider a *Curriculum* as a graph of tasks $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$. For building $\mathbb{C}$, we use a strategy similar to Svetlik's [30], adding in the *Curriculum* tasks with high transfer potential, which we here calculate based on the *Object-Oriented* description as:

$$v(\mathcal{T}_s, \mathcal{T}_t, \boldsymbol{\mathcal{T}}_{\mathbb{C}}) = \frac{sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_t)}{\max_{\mathcal{T}_i \in \boldsymbol{\mathcal{T}}_{\mathbb{C}}} sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_i)} \cdot \frac{|S_t|(|\boldsymbol{O}^{\mathcal{T}_t}| + 1)}{|S_s|(|\boldsymbol{O}^{\mathcal{T}_s}| + 1)} \quad (3)$$

where $\mathcal{T}_s$ is the task to measure the transfer potential, $\mathcal{T}_t$ is the target task, $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$ is the set of tasks already defined to be executed[3] before $\mathcal{T}_t$, $sim_{\mathbb{C}}(\mathcal{T}_i, \mathcal{T}_j)$ is a similarity value between tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, $|S_i|$ is the size of the state space in $\mathcal{T}_i$, and $\boldsymbol{O}^{\mathcal{T}_i}$ is the set of objects of task $\mathcal{T}_i$.

The intuition behind this equation is to prioritize the inclusion of tasks that are: (i) similar to the target task, increasing the usefulness of the learned policy; (ii) dissimilar from the previously included tasks, to reduce the amount of redundant knowledge; and that (iii) have a smaller state space than in the target task, to first train in smaller tasks. Here, the similarity between tasks is calculated as:

$$sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_t) = \sum_{C_i \in \boldsymbol{C}} \frac{|\boldsymbol{O}_{C_i}^{\mathcal{T}_s} \cap \boldsymbol{O}_{C_i}^{\mathcal{T}_t}|}{\max(|\boldsymbol{O}_{C_i}^{\mathcal{T}_s}|, |\boldsymbol{O}_{C_i}^{\mathcal{T}_t}|)} + \frac{|S_s \cap S_t|}{|S_t|} \quad (4)$$

[3] If $\boldsymbol{\mathcal{T}}_{\mathbb{C}} = \emptyset$, consider $\max_{\mathcal{T}_i \in \boldsymbol{\mathcal{T}}_{\mathbb{C}}} sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_i) = 1$.

where $\mathcal{T}_s$ and $\mathcal{T}_t$ are tasks from the same domain, C_i is one class from the set $\boldsymbol{C}$, $\boldsymbol{O}_{C_i}^{\mathcal{T}_s} \cap \boldsymbol{O}_{C_i}^{\mathcal{T}_t}$ is the set of objects that belong to class C_i and have the same attribute values in the initial state for both $\mathcal{T}_s$ and $\mathcal{T}_t$, and $|S_s \cap S_t|$ is an estimate of the number of states the two tasks have in common[4]. Notice that the real similarity between two tasks cannot be computed because it depends on the transition and reward functions that are unknown to the agent. The proposed equation gives a good indication of the true similarity if: (i) object attributes have the same semantic meaning in both tasks; and (ii) similar tasks in this domain have some objects in common. We believe that those are reasonable assumptions for the purpose of building *Curricula*.

We generate $\mathbb{C}$ by following Algorithm 3 (which fits as a *buildCurriculum* function in Alg. 1 line 2). The set of source tasks $\boldsymbol{\mathcal{T}}$ to be used as a parameter has been manually defined by humans in the works so far, but an automatically generated set could also be used (as in our proposed method detailed in the next section).

Algorithm 3 Automatic Curriculum Generation

Require: set of source tasks $\boldsymbol{\mathcal{T}}$, target task $\mathcal{T}_t$, threshold for task inclusion ϵ.

1: $\boldsymbol{\mathcal{V}} \leftarrow \emptyset, \boldsymbol{\mathcal{E}} \leftarrow \emptyset$ ▷ Initializing the Curriculum Graph
2: $(G, \boldsymbol{\mathcal{T}}_{\mathbb{C}}) = groupTasks(\boldsymbol{\mathcal{T}}, \mathcal{T}_t, \epsilon)$ ▷ Algorithm 4
3: $\boldsymbol{\mathcal{V}} \leftarrow \boldsymbol{\mathcal{T}}_{\mathbb{C}} \cup \mathcal{T}_t$
4: **for** $\forall g \in G$ **do** ▷ Intra-group Transfer
5: **for** $\forall \mathcal{T}_i \in g$ **do**
6: $\mathcal{T}_m = \arg\max_{T_j \in g | j > i} v(\mathcal{T}_j, \mathcal{T}_i, children(\mathcal{T}_i))$ ▷ Eq. (3)
7: **if** $v(\mathcal{T}_m, \mathcal{T}_i, children(\mathcal{T}_i)) > \epsilon$ **then**
8: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \langle \mathcal{T}_m, \mathcal{T}_i \rangle$
9: **for** $\forall g \in G$ **do** ▷ Inter-group Transfer
10: **for** $\forall g' : g' \in G$ and $g.features \subset g'.features$ **do**
11: **for** $\forall \mathcal{T}_i \in g$ **do**
12: $\mathcal{T}_m = \arg\max_{T_j \in g'} v(\mathcal{T}_j, \mathcal{T}_i, children(\mathcal{T}_i))$ ▷ Eq. (3)
13: **if** $v(\mathcal{T}_m, \mathcal{T}_i, children(\mathcal{T}_i)) > \epsilon$ **then**
14: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \langle \mathcal{T}_m, \mathcal{T}_i \rangle$
15: **for** $\forall \mathcal{T}_i : \mathcal{T}_i \in \boldsymbol{\mathcal{T}}_{\mathbb{C}}$ and $deg^+(\mathcal{T}_i) = 0$ **do** ▷ Add edges to target
16: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \langle \mathcal{T}_i, \mathcal{T}_t \rangle$
17: $\mathbb{C} \leftarrow \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$
18: return $\mathbb{C}$

The first step is to split the set of source tasks $\boldsymbol{\mathcal{T}}$ in groups according to their parameters (line 2). The grouping procedure has two purposes: (i) discarding tasks that are not expected to be beneficial for transfer and (ii) organizing $\boldsymbol{\mathcal{T}}$ into groups of tasks with the same parameters. Imagine that $\mathcal{T}_t$ is a robot soccer task against two opponents. A possible resulting set of groups would be: one group composed of tasks with no opponents; one composed of one-opponent tasks; and one with two-opponent tasks, where the tasks in which the initial states are very different from the one in $\mathcal{T}_t$ are discarded. Our grouping procedure is fully described in Algorithm 4 and the main idea is to build a set of candidate tasks $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$ discarding tasks with low *Object-Oriented* transfer potential (Equation (3)).

[4] This estimate is computed by evaluating the intersection of the object attribute $domains$ in the tasks. Thus, no additional knowledge is needed.

Algorithm 4 groupTasks

Require: set of source tasks $\mathcal{T}$, target task $\mathcal{T}_t$, threshold for task inclusion ϵ.

1: $\mathcal{T}_{\mathbb{C}} \leftarrow \emptyset$
2: $G \leftarrow \emptyset$
3: **for** $\forall \mathcal{T}_i \in \mathcal{T}$ **do**
4: **if** $v(\mathcal{T}_i, \mathcal{T}_t, \mathcal{T}_{\mathbb{C}}) > \epsilon$ **then** ▷ Eq. (3)
5: **if** $\nexists \boldsymbol{g} : \boldsymbol{g} \in G$ and $\boldsymbol{g}.features = \mathcal{T}_i.features$ **then**
6: $G \leftarrow G \cup group(\mathcal{T}_i.features)$ ▷ New group
7: $\boldsymbol{g} = \{\boldsymbol{g} \in G | \boldsymbol{g}.features = \mathcal{T}_i.features\}$
8: $\boldsymbol{g} \leftarrow \boldsymbol{g} \cup \mathcal{T}_i$
9: $\mathcal{T}_{\mathbb{C}} \leftarrow \mathcal{T}_{\mathbb{C}} \cup \mathcal{T}_i$
10: Sort tasks within each group ($\mathcal{T}_i \in \boldsymbol{g}$) by $v(\mathcal{T}_i, \mathcal{T}_t)$
11: return $(G, \mathcal{T}_{\mathbb{C}})$

Algorithm 4 returns a group for each possible task parameter in which there exists at least one task with transfer potential higher than threshold ϵ (Alg 4 line 4).

After the set of task groups G is built, we define the edges of the *Curriculum* as proposed by Svetlik *et al.* [30]. Firstly we search for pairs of tasks that have a high transfer potential between themselves within the same task group (Alg. 3 lines 4-8), then we search for pairs that belong to different groups, as long as the features of one task are contained in another's (lines 9-14). Here, operation $children(\mathcal{T}_i)$ returns the set of tasks already included in the *Curriculum* that have an edge to $\mathcal{T}_i$. In the example of a Robot Soccer task, $\boldsymbol{g}.features \subset \boldsymbol{g'}.features$ if the number of opponents in tasks inside $\boldsymbol{g}$ is smaller than for the tasks inside $\boldsymbol{g'}$. Hence, the knowledge from the simpler tasks will be transferred to the most complex ones. Finally, we create an edge from the tasks with outdegree $zero$[5] to the target task (line 16).

As an alternative for Algorithm 2, which is ineffective for big *Curricula*, we propose a novel procedure to use only portions of a *Curriculum* (Algorithm 5). We use a biased *Random Walk* to traverse the *Curriculum* graph $\mathbb{C}$ starting from the target task (a back-to-front path - line 5), without, however, visiting every vertex in $\mathbb{C}$. Then, we walk n_{steps} steps from $\mathcal{T}_t$, biasing each step according to the transfer potential of each of its *children*, i.e., tasks with high transfer potential are more likely to be selected (lines 5-9):

$$p(\mathcal{T}_i) = \frac{v(\mathcal{T}_i, curTask, \boldsymbol{T})}{\sum_{\mathcal{T}_j \in children(curTask)} v(\mathcal{T}_j, curTask, \boldsymbol{T})}. \quad (5)$$

Each of the selected tasks are labeled[6] for posterior use, and we keep a list of selected tasks for computing the transfer potential (lines 7 and 9). This procedure is repeated n_{traj} times. Finally, all labeled tasks are used for learning, finishing in the target task (lines 10-13).

Then, a procedure for reusing knowledge from the previously solved tasks must be defined. We use a method based on value function reuse as performed by Narvekar *et al.* [18], but taking advantage of the *Object-Oriented* representation. For that, we firstly map the current state to a set of similar states in the source tasks.

[5]outdegree $zero$ means that no edge is starting from the task, i.e., it still has no parents.
[6]A *label* here is binary mark on a vertex, posteriorly used for defining the tasks that were chosen for execution.

Algorithm 5 *learn* with biased *Random Walk*.

Require: *Curriculum* $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, target task $\mathcal{T}_t$, n_{traj}, n_{steps}.

1: $\boldsymbol{T} \leftarrow \emptyset$
2: **for** n_{traj} times **do**
3: $curTask \leftarrow \mathcal{T}_t$
4: $label(curTask)$
5: **for** n_{steps} times **do**
6: Draw $\mathcal{T}_i \in children(curTask)$ according to Eq. (5)
7: $label(\mathcal{T}_i)$
8: $curTask \leftarrow \mathcal{T}_i$
9: $\boldsymbol{T} \leftarrow \boldsymbol{T} \cup \mathcal{T}_i$
10: **while** $\exists labeled(\mathcal{T}_i), \forall \mathcal{T}_i \in \boldsymbol{T}$ **do**
11: Select a labeled task $\mathcal{T}_i$ with no labeled children
12: Learn $\mathcal{T}_i$ reusing previous knowledge
13: $unlabel(\mathcal{T}_i)$

Then, we reuse their value functions in the new Q-table. This mapping is calculated with Probabilistic Inter-TAsk Mappings (PITAM), as proposed in [24]. A PITAM is a mapping between states in two tasks $\mathscr{P}_{\mathcal{T}_s, \mathcal{T}_t} : S_{\mathcal{T}_t} \times S_{\mathcal{T}_s} \rightarrow [0, 1]$. For defining $\mathscr{P}$, a similarity metric sim_{PITAM} is calculated between the current state and all the state space in the source task by using the *Object-Oriented* description. Then, $\mathscr{P}$ is calculated by normalizing the similarity values into probabilities: $\sum_{s \in S_{\mathcal{T}_s}} \mathscr{P}_{\mathcal{T}_s, \mathcal{T}_t}(s_t, s) = 1$. We here define the similarity value as : $sim_{PITAM}(s_t, s_s) = |\boldsymbol{O}_{s_t} \cap \boldsymbol{O}_{s_s}|$, where $|\boldsymbol{O}_{s_t} \cap \boldsymbol{O}_{s_s}|$ denotes the number of objects that belong to the same class and have the same attribute values for two particular states s_t and s_s. That is, states that have no object in common have a mapping with zero probability, and the higher the number of common objects, the higher the mapping probability will be. Although calculating exact PITAM probabilities is computationally expensive, it is easy to use domain knowledge to prune the state space search (e.g., calculating similarity values only for states in which there are objects with similar attribute values). After the PITAM calculation, we initialize the Q-table in the new task $\mathcal{T}_t$ as:

$$Q_{\mathcal{T}_t}(s_t, a_t) \leftarrow \frac{\sum_{\mathcal{T}_s \in \mathbb{C}_{\mathcal{T}_t}} \sum_{s_s \in S_{\mathcal{T}_s}} \mathscr{P}_{\mathcal{T}_s, \mathcal{T}_t}(s_t, s_s) Q_{\mathcal{T}_s}(s_s, a_t)}{|\mathbb{C}_{\mathcal{T}_t}|}, \quad (6)$$

where $\mathbb{C}_{\mathcal{T}_t}$ is the set of already solved tasks that had an edge to $\mathcal{T}_t$ in the original *Curriculum* graph. Notice that the object attributes must be agent-centered to facilitate transfer. For example, if the agent is in a world represented by a grid, it is very hard to generalize states if the object observations are given by their absolute positions. However, if distances between the agent and objects are used as attributes (agent-centered representation), it is much easier to find state correspondences (i.e., find objects with the same attribute values in two states).

Using the *Curriculum* and this transfer procedure, the agent is expected to learn faster than when learning from scratch. In the next Section we describe our proposal to automatically generate source tasks ($\mathcal{T}$ in Algorithm 3).

4 OBJECT-ORIENTED AUTOMATIC SOURCE TASK GENERATION

The success of a *Curriculum* depends on: (i) a proper set of source tasks; (ii) a proper task ordering; (iii) the efficacy of the chosen TL algorithm; (iv) a good stopping criterion to identify when to switch tasks. Even though Svetlik *et al.* [30] had proposed a method to define the sequence, automatically defining the set of source tasks was still an open problem [18, 20, 30].

We here propose a method to generate source tasks by using the *Object-Oriented* representation, requiring less human effort than previous works in which this set must be manually given. The intuition of our proposal is to randomly select a portion of the objects from the target task to build each of the (smaller) source tasks, assuming that smaller tasks are easier to solve.

Algorithm 6 fully describes our source task generation procedure (which fits as a *createTasks* function in Alg. 1 line 1). At first we define the set $\boldsymbol{F}_{simple}$, which is a set of possible valuations for the *environment* objects in the target task (line 2). We assume that all domains have an *environment* class C_{Env}, which has as attributes domain features not related to other classes (e.g., the grid size in a *Gridworld*). The $simplify$ function creates environment objects corresponding to simplified versions of the target task, for example, if the environment object is the size of a grid, a possible simplification would be a set of values corresponding to smaller grids. Then, we define the set of objects belonging to the class with fewer objects among the remaining classes C_{min} (lines 3–4). We then create o_{min} tasks, each of them containing from 0 to o_{min} objects from C_{min} (line 6), one possible set of values from $\boldsymbol{F}_{simple}$ (line 7), and a random number of objects from the other classes (line 11). The initial state for this new source task if defined through the *initState* function. A possible way to implement this function is to draw random attribute values for all objects, or copy the values from the target task initial state (when applicable). The action space, transition function, and reward functions for the new task are then defined through the *actionSpace*, $transitionFunction$, and $rewardFunction$ functions[7], and the task is finally added to the set of source tasks $\boldsymbol{\mathcal{T}}$. This process is repeated multiple times according to a parameter n_{rep}.

This procedure can be used to generate a set of source tasks when a human is unavailable or unwilling to provide it. However, this procedure is not valid for all possible domains, because we change the number of objects without necessarily having knowledge about the transition function. When using this procedure, the designer must ensure that solvable tasks are generated. Thus, it might be necessary to set a minimum number of objects of a given class, to create the initial state in a domain-dependent way, or to add a final human-guided step to reject unsolvable tasks (which is still much easier than hand-coding the entire set).

In the next Section we present our experimental evaluation.

5 EXPERIMENTAL EVALUATION

We here describe our experiments to show that our proposal builds useful *Object-Oriented Curricula*. Firstly we present our experimental setup, then the results along with discussion.

[7]If transition and reward functions are unknown, the environment will implicitly provide samples of those functions to the agent.

Algorithm 6 *createTasks*

Require: target task $\mathcal{T}_t$, repetition parameter n_{rep}.

1: $\boldsymbol{\mathcal{T}} \leftarrow \emptyset$
2: $\boldsymbol{F}_{simple} \leftarrow simplify(\boldsymbol{O}^{\mathcal{T}_t}_{C_{Env}})$
3: $C_{min} \leftarrow \arg\min_{C_i \in \boldsymbol{C} - C_{Env}} |\boldsymbol{O}^{\mathcal{T}_t}_{C_i}|$
4: $o_{min} \leftarrow |\boldsymbol{O}^{\mathcal{T}_t}_{C_{min}}|$
5: **for** n_{rep} times **do**
6: **for** $\forall i \in \{0, \ldots, o_{min}\}$ **do**
7: Draw F from $\boldsymbol{F}_{simple}$
8: $\boldsymbol{O} \leftarrow$ Draw i objects from $\boldsymbol{O}^{\mathcal{T}_t}_{C_{min}}$
9: **for** $\forall C_i \in \boldsymbol{C} - C_{Env}$ **do**
10: Draw q from $\{0, \ldots, |\boldsymbol{O}^{\mathcal{T}_t}_{C_i}|\}$
11: $\boldsymbol{O} \leftarrow \boldsymbol{O} \cup$ Draw q objects from $\boldsymbol{O}^{\mathcal{T}_t}_{C_i}$
12: $\boldsymbol{O} \leftarrow \boldsymbol{O} \cup F$
13: $S_0 \leftarrow initState(\boldsymbol{O}, \mathcal{T}_t)$
14: $A \leftarrow actionSpace(\boldsymbol{O}, \mathcal{T}_t, A^{\mathcal{T}_t})$
15: $T \leftarrow transitionFunction(T^{\mathcal{T}_t})$
16: $R \leftarrow rewardFunction(R^{\mathcal{T}_t})$
17: $\boldsymbol{\mathcal{T}} \leftarrow \boldsymbol{\mathcal{T}} \cup \langle \boldsymbol{C}, \boldsymbol{O}, S_0, A, T, R \rangle$
18: **return** $\boldsymbol{\mathcal{T}}$

5.1 Experimental Setup

We have chosen two domains for our experimental evaluation. The *Gridworld* domain, as proposed by Sevtlik *et al.* [30], and the *Half Field Offense* (HFO) [11] environment. The former shows the performance of the proposed method in a domain easy to implement and to gather results, while the latter shows the robustness of the method in a very challenging multiagent task with continuous observations[8]. As pointed out in Section 7, Curriculum Generation especially tailored to MAS is still an open question for further work. We evaluate our proposal both with manually given (*OO-Given*) and generated (*OO-Generated*) source tasks, comparing it with Svetlik's [30] *Curriculum* generation procedure (hereafter named *Svetlik*) and the regular learning without a *Curriculum* (Q-Learning for *Gridworld* and SARSA for HFO - hereafter called as *No Curriculum*). The TL procedure is carried out as defined in Section 3 for our proposal and through transfer of value functions [35] for *Svetlik*.

5.1.1 Gridworld. Figure 1 illustrates our *Gridworld*. Each cell in the grid may have one of the following objects or be empty: *fire*, *pit*, or *treasure*. The agent can move in four cardinal directions $A = \{North, South, East, West\}$, and the task is solved when the agent collects the treasure. The *Object-Oriented* description of the task has the classes $C = \{Pit, Fire, Treasure, Env\}$, where the environment attributes are the sizes of the grid $Att(Env) = \{sizeX, sizeY\}$ and the remaining classes have x and y attributes. The position attributes are observed in regard to the distance between the agent and the object, rather than the absolute position. The rewards are defined as in [30]. At each of every executed step, the agent observes one of the following rewards: (i) **+200** for collecting the

[8]Notice that, in the current work, we assume that the sequence of tasks is defined by a single agent. In case those are multiagent tasks, additional agents might join the learning agent in the tasks selected by it when required, but they cannot interfere with the *Curriculum* generation and use.

treasure; (ii) **-250** for getting next to a fire (4-neighborhood); (iii) **-500** for getting into a fire; (iv) **-2500** for falling into a pit; and (v) **-1** if nothing else happened.

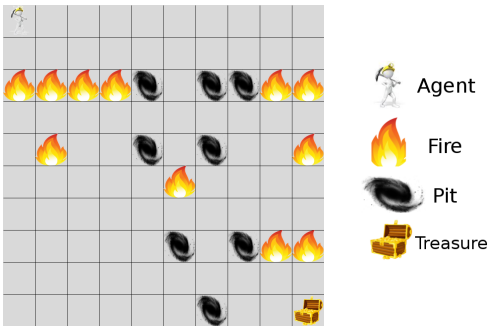

Figure 1: An illustration of the target task in Gridworld.

The target task is illustrated in Figure 1, and contains 8 pits, 11 fires, and 1 treasure. As in [30], we generate a set of source tasks by reducing the number of objects and/or reducing the size of the grid, defining $|\mathcal{T}| = 18$ source tasks. During learning, all source tasks given by the agent *Curriculum* were executed until the agent achieved the same cumulative reward for 2 consecutive episodes, or 30 learning episodes were carried out. Episodes start in the task initial state (e.g., the configuration shown in Figure 1) and end when the agent captures the treasure or falls into a pit.

The comparison metric here is the cumulative reward when trying to solve the target task. The threshold parameter for *Curriculum* generation was set $\epsilon = 1$, $n_{traj} = 3$, and $n_{steps} = 3$ for our proposal and $\epsilon = 15$ for *Svetlik*[9]. The task generation parameters for our proposal are $\epsilon = 0.5$, $n_{rep} = 2$, $n_{traj} = 2$, and $n_{steps} = 2$.

5.1.2 HFO. This domain is a simplification of the full RoboCup [13] simulated Soccer task. In our setting, a learning agent has to score a goal against a team of two highly specialized defensive agents (the initial state of the target task is illustrated in Figure 2).

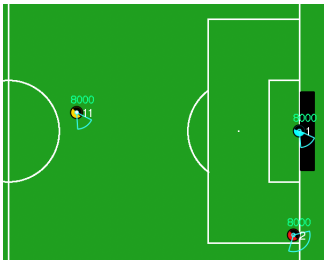

Figure 2: An illustration of the target task in HFO.

The defensive team follows the Helios policy [1] (the 2012 RoboCup champion team). The performance in this domain is defined by the percentage of scored goals in a predefined number of attempts. A learning episode starts with all agents initiated in a random position (defensive agents always near the goal), and the ball possession with the offensive agent. One opponent plays the role of goalkeeper, while the other is a defender. The episode ends when the agent scores a goal, a defensive agents captures the ball, the ball leaves the field, or after 200 game frames. We define the reward function as in [11]. A reward of +1 is awarded if a goal was scored, a reward of −1 is given in case a defensive agent captures the ball or the ball leaves the field, and a reward of 0 is given otherwise. When carrying the ball, the available actions are:

(1) **Shoot** – takes the best available shot;
(2) **Dribble** – advances the agent and ball towards the goal.

Without the ball possession the only available action is **Move**, which is a macroaction that tries to move the agent towards the best possible position in the field. In spite of only having two available actions when carrying the ball, this task is still very hard to solve, as a reward is received only when the episode ends and the agent scores a goal only when shooting at a propitious moment. The Helios strategy (when also playing as the offensive agent) scores in roughly 30% of the attempts, while a random agent has a score of roughly 3% of the attempts. We make use of the following observations of the current state[10], normalized in the range $[-1, 1]$:

- **Goal Proximity** – the distance from agent to goal center;
- **Goal Angle** – the angle from agent to goal center;
- **Goal Opening** – the largest angle from agent to goal with no blocking agent;
- **Opponent Proximity** – the distance between the agent and the nearest opponent.

The observations are discretized by Tile Coding [22] configured with 5 tiles of size 48 and equally spread over the range. The *Object-Oriented* description of the task has the classes $C = \{Agent, Opponent, Env\}$, in which the environment attribute is the average initial distance between the offensive agent and the goal $Att(Env) = \{dist\}$, *Agent* has attributes corresponding to the aforementioned observations, and *Opponent* has a single attribute specifying the agent strategy (*Helios* or *Base*). We generated $|\mathcal{T}| = 8$ tasks, in which we vary the initial distance between the teams and the number and strategy of opponents. The source tasks are executed until one of the following conditions is true: (i) the agent scores 80% of the attempts in the last 20 episodes; (ii) no goal is scored in the last 50 episodes; or (iii) after 500 learning episodes.

The threshold parameter for *Curriculum* generation was set $\epsilon = 0.5$, $n_{traj} = 3$, and $n_{steps} = 3$ for our proposal and $\epsilon = 0.75$ for *Svetlik*. The task generation parameters for our proposal are $\epsilon = 0.5$, $n_{rep} = 20$, $n_{traj} = 3$, and $n_{steps} = 3$.

5.2 Results

We here present the experimental results from both domains.

5.2.1 Gridworld. Figure 3a shows the performance observed when solving the target task in the *Gridworld* domain. Although *Svetlik* presents very good results in [30], we found out that their proposal is quite sensitive to parameters and to the provided source tasks. The performance shown in our experiments is the best result observed after trying several parameters, but it is still a little

[9]The parameters were defined in preliminary experiments. As Equations (1) and (3) result in values of different magnitudes, the threshold follows a different scale for each algorithm.

[10]For a list of all usable observations, refer to [11].

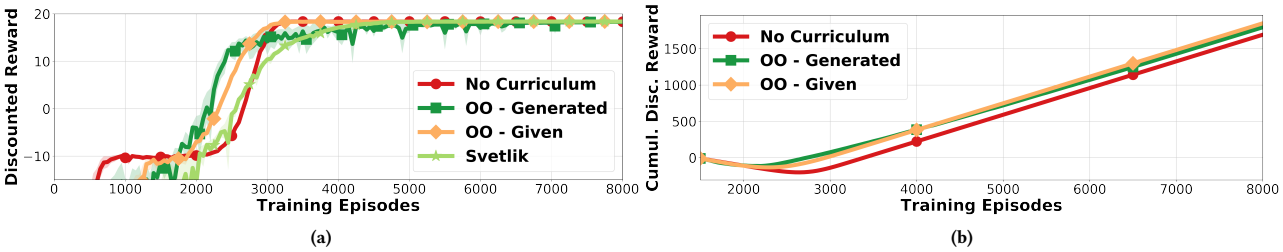

Figure 3: The average discounted rewards observed in 2,000 repetitions of the experiment in the *Gridworld* domain. (a) refers to the average performance during learning; and (b) to the cumulative rewards starting from step 1,500. Steps used to learn source tasks are also considered in the graph. The shaded area is the 95% confidence interval.

worse than *No Curriculum* if the steps in source tasks are also taken into account. In turn, both *OO-Generated* and *OO-Given* achieved positive results when compared to *No Curriculum*. Both of them achieved a slightly worse performance than *No Curriculum* until 1,500 learning steps. Then, our proposal learns how to avoid the negative rewards faster, achieving better results than *No Curriculum* between 1,700 and 3,000 learning steps. Then, *OO-Given* achieves the optimal policy few steps before *No Curriculum* while *OO-Generated* takes a little longer, but *OO-Generated* has already settled in a very good performance (near the optimal) by then. Figure 3b shows the accumulated reward in the target task starting from 1,500 steps for both configurations of our proposal and *No Curriculum*. All the three algorithms have similar negative cumulative reward up to roughly 2,300 learning steps, after which the improvement in the learning process when using our proposal becomes clear. This shows that even though fewer steps are needed in the target task, the performance achieved in training is better.

Although Figure 3a assumes that learning steps have the same difficulty for the agent in both the source and target tasks, for some applications learning in source tasks may be easier (for example, if source tasks are learned in a virtual simulator and the target task is in the real world [10]). Therefore, Figure 4 shows the results if the learning steps in source tasks are not considered. The difference is not dramatic for our approach (but still slightly better than in Figure 3a), as the source tasks are learned very fast. However, for *Svetlik* this represents a huge "speed-up". Now *Svetlik* learns faster than *No Curriculum* how to avoid negative rewards and it is even a little better than our proposal, despite taking longer than *OO-Given* to converge to the optimal policy. The results here show that both Svetlik's and ours proposals might be useful if steps in the source task are less costly than in the target task. Notice that the transfer potential metric is easier to compute with our proposal if an object-oriented description is available, though.

5.2.2 HFO. Figure 5a shows the goal percentage when solving the task for all algorithms. As the agents have a challenging task to solve, it is very hard to improve performance. At the end of training, all algorithms score goals around 20% of the attempts. This is a good performance, as the *Helios* team scores roughly in 30% of the attempts and it uses several hand-coded specialized strategies.

None of the *Curriculum Learning* approaches can achieve a better performance than *No Curriculum* at the end of learning, but all *Curriculum* approaches learn in the target task for less time. *OO-Given* starts learning in the target task around episode 6,000, and quickly reaches *No Curriculum* performance. *OO-Generated* takes longer to start learning in the target task, but the performance achieved is already similar to *No Curriculum* as soon as it starts learning in the target task. *Svetlik* also takes very long to start learning in the target task, which happens at roughly episode 52,000. After starting learning in the target task, *Svetlik* takes longer than our proposal to achieve *No Curriculum* performance, getting the same performance than the other algorithms at roughly episode 70,000.

Figure 5b more clearly shows that our proposal achieved a good performance faster than *No Curriculum* when taking into account only steps in the target task. *Svetlik* was also slightly better than *No Curriculum* but worse than both settings of our proposal. The number of cumulative goals scored by each approach (not shown graphically here) indicates that *Svetlik* achieves the same cumulative performance as *No Curriculum*, while both *OO-Given* and *OO-Generated* achieve a slightly better performance, thus making our proposal a better choice for this domain.

5.2.3 Summary of Experiments. In both experiments *OO-Given* and *OO-Generated* presented a better performance than *Svetlik* and *No Curriculum* in general. In the *Gridworld* domain our proposal achieved a good performance clearly faster than not using *Curriculum*, presenting advantages in both performance and number of steps executed in the target task. In the *HFO* domain, our proposal achieved a similar performance than not using *Curriculum*,

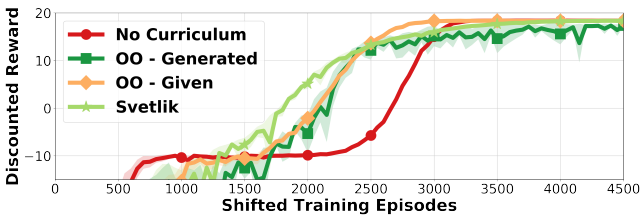

Figure 4: The average discounted rewards disregarding steps carried out in source tasks in the *Gridworld* domain. The shaded area is the 95% confidence interval.

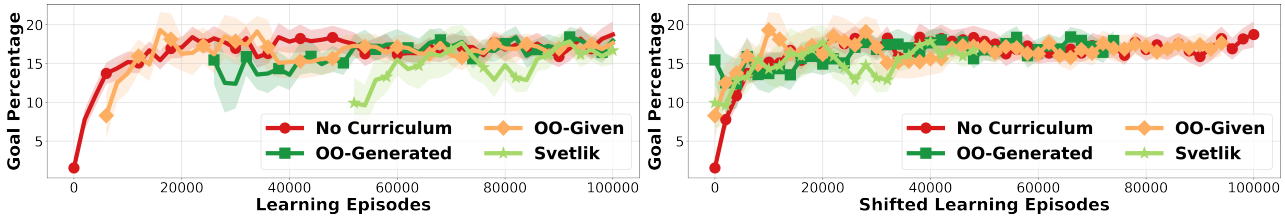

Figure 5: The average percentage of goals observed in 50 repetitions of the experiment in the *HFO* domain when the steps used to learn source tasks are: (a) considered; and (b) not considered. The shaded area is the 95% confidence interval.

but still required fewer steps training in the target task. Our experiments also show that the performance of our proposal when generating tasks (*OO-Generated*) is consistent and satisfactory, as in both domains the use of generated tasks achieved a competitive performance when compared to the manually generated set of tasks, which requires more manual intervention and domain-knowledge.

6 RELATED WORKS

Curriculum Learning for RL is a relatively new area of research. Narvekar *et al.* [18] were the first to propose the use of *Curricula* for RL. They showed that a *Curriculum* can indeed be used to accelerate learning in the complex *Half Field Offense* [11] and *Pacman* domains. However, their proposal requires a human-provided set of source tasks and a manually specified sequence of tasks to be executed by the agent. Later works showed that building a good *Curriculum* is not easy, especially if *Curricula* are built by non-experts human operators [20], which agrees with our view that, whenever possible, automatically generating *Curricula* could be very beneficial. Svetlik *et al.* [30] then proposed a method to estimate a *Curriculum* graph, which we used as base for our proposal. Narvekar *et al.* [19] later proposed an alternative method to generate *Curricula* by building a *Curriculum* MDP (CMDP), that is, modeling the autonomous construction of the task sequence as a sequential decision-making problem. However, their proposal is focused on adapting the *Curriculum* for the agent's individual capabilities, and the authors state that the complexity of learning in the CMDP makes it slower than learning from scratch if a previous *Curriculum* is not reused. Compared to our approach, none of those works focused on how to reuse knowledge from the already solved source tasks, or on how to automatically generate the set of source tasks. Sukhbaatar *et al.* [27] divided an agent into two components; one component creates *Curricula* to the other by manipulating when the control is shifted between them, hence artificially creating "initial" and "goal" states. However, this procedure is rather a manipulation of the learning algorithm than a source task creation, as only those states change between the generated tasks.

The area of *Transfer Learning (TL)* is closely related to the use of *Curricula*, as we could devise many ways to reuse knowledge from one task to another through TL. Previous works have successfully transferred samples of low-level interactions with the environment [16, 31, 33], policies [8, 36], value or Q functions [9, 32, 35], action suggestions [26, 38], and heuristics or biases for a more effective exploration [3, 4], each of them presenting benefits over learning from scratch, and they could all be potentially combined with *Curriculum Learning*. Other works also made use of relational representations to transfer knowledge [14, 15, 37], using either OO-MDPs or similar models, such as *Relational MDPs* [5, 7]. Multi-task Learning [8] and Lifelong Learning [12] are also related to our work, but under these paradigms the sequence of source tasks and the switch between them cannot be controlled as we do here.

7 CONCLUSION AND FURTHER WORK

Accelerating the learning process of Reinforcement Learning (RL) tasks is one of the main current concerns of the Machine Learning community. The use of Curriculum Learning in RL is an emerging and promising technique, but the previous works require carefully extracted domain knowledge to work, in the form of *Curriculum* construction and manual source task base generation. We here propose a procedure to generate the set of source tasks for a *Curriculum*, requiring less domain-specific knowledge than in the previous works. We also propose procedures to *Object-Oriented* Curriculum generation, which builds a *Curriculum* graph by using an intuitively-given *Object-Oriented* task description, and to use the generated *Curriculum* through a biased *Random Walk* approach. We have shown in an empirical evaluation that our proposal presents advantages over previous works in two challenging domains.

This work opens several lines of possible developments. The first one is the development of additional principled procedures for pruning *Curricula*, preferably adapting it according to the agent's unique particularities. We also intend to work on the development of *Curriculum* generation procedures especially tailored for Multiagent RL Systems [4], for which an *object-oriented* model already exists [25]. The *Transfer of Curriculum*, that is, autonomously transferring a *Curriculum* and adapting it to a new agent or for a new target task, could also be an exciting challenge for further developments in the area. Future works can also evaluate if the *Object-Oriented* representation can help humans to better understand the agent learning process [21] and to build better *Curricula*.

ACKNOWLEDGMENTS

We acknowledge financial support from CNPq, grants 311608/2014-0 and 425860/2016-7, and São Paulo Research Foundation (FAPESP), grants 2015/16310-4, 2016/21047-3, and 2018/00344-5.

REFERENCES

[1] Hidehisa Akiyama. 2012. Helios team base code. https://osdn.jp/projects/rctools/. (2012).

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*. 41–48. https://doi.org/10.1145/1553374.1553380

[3] Reinaldo A. C. Bianchi, Luiz A. Celiberto Jr., Paulo E. Santos, Jackson P. Matsuura, and Ramon Lopez de Mantaras. 2015. Transferring Knowledge as Heuristics in Reinforcement Learning: A Case-Based Approach. *Artificial Intelligence* 226 (2015), 102 – 121. https://doi.org/10.1016/j.artint.2015.05.008

[4] Georgios Boutsioukis, Ioannis Partalas, and Ioannis Vlahavas. 2011. Transfer Learning in Multi-agent Reinforcement Learning Domains. In *Proceedings of the 9th European Workshop on Reinforcement Learning*. http://ewrl.files.wordpress.com/2011/08/ewrl2011_submission_19.pdf

[5] Tom Croonenborghs, Karl Tuyls, Jan Ramon, and Maurice Bruynooghe. 2005. Multi-agent Relational Reinforcement Learning. In *Learning and Adaption in Multi-Agent Systems*. 192–206. https://doi.org/10.1007/11691839_12

[6] Carlos Diuk, Andre Cohen, and Michael L. Littman. 2008. An Object-Oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*. 240–247. https://doi.org/10.1145/1390156.1390187

[7] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. 2001. Relational Reinforcement Learning. *Machine Learning* 43, 1-2 (2001), 7–52. https://doi.org/10.1023/A:1007694015589

[8] Fernando Fernández and Manuela Veloso. 2006. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 720–727. https://doi.org/10.1145/1160633.1160762

[9] Ruben Glatt, Felipe Leno Da Silva, and Anna Helena Reali Costa. 2016. Towards Knowledge Transfer in Deep Reinforcement Learning. In *Proceedings of the 5th Brazilian Conference on Intelligent Systems (BRACIS)*. 91–96.

[10] Josiah Hanna and Peter Stone. 2017. Grounded Action Transformation for Robot Learning in Simulation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. 3834–3840.

[11] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. 2016. Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*. http://www.cs.utexas.edu/~AustinVilla/sim/halffieldoffense/

[12] David Isele, Mohammad Rostami, and Eric Eaton. 2016. Using Task Features for Zero-Shot Knowledge Transfer in Lifelong Learning. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. 1620–1626.

[13] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. 1997. RoboCup: A challenge problem for AI. *AI magazine* 18, 1 (1997), 73–85.

[14] Marcelo Li Koga, Valdinei Freire da Silva, and Anna Helena Reali Costa. 2015. Stochastic Abstract Policies: Generalizing Knowledge to Improve Reinforcement Learning. *IEEE Transactions on Cybernetics* 45, 1 (2015), 77–88. https://doi.org/10.1109/TCYB.2014.2319733

[15] Marcelo Li Koga, Valdinei Freire da Silva, Fabio Gagliardi Cozman, and Anna Helena Reali Costa. 2013. Speeding-up Reinforcement Learning Through Abstraction and Transfer Learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 119–126.

[16] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. 2008. Transfer of Samples in Batch Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*. 544–551.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. 2015. Human-level Control through Deep Reinforcement Learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

[18] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. 2016. Source Task Creation for Curriculum Learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 566–574.

[19] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. 2017. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 2536–2542.

[20] Bei Peng, James MacGlashan, Robert Loftin, Michael L. Littman, David L. Roberts, and Matthew E. Taylor. 2016. An Empirical Study of Non-expert Curriculum Design for Machine Learners. In *Proceedings of the IJCAI Interactive Machine Learning Workshop*. http://irll.eecs.wsu.edu/wp-content/papercite-data/pdf/2016iml-peng.pdf

[21] Ramya Ramakrishnan, Karthik Narasimhan, and Julie Shah. 2016. Interpretable Transfer for Reinforcement Learning based on Object Similarities. In *Proceedings of the IJCAI Interactive Machine Learning Workshop*.

[22] Alexander A. Sherstov and Peter Stone. 2005. Function Approximation via Tile Coding: Automating Parameter Choice. In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation (SARA)*. 194–205.

[23] Felipe Leno Da Silva and Anna Helena Reali Costa. 2017. Automatic Object-Oriented Curriculum Generation for Reinforcement Learning. In *Proceedings of the 1st Workshop on Scaling-Up Reinforcement Learning (SURL)*. http://surl.tirl.info/proceedings/SURL-2017_paper_1.pdf

[24] Felipe Leno Da Silva and Anna Helena Reali Costa. 2017. Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description. In *Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL)*. http://www.tirl.info/proceedings/2017/SilvaCosta-Towards.pdf

[25] Felipe Leno Da Silva, Ruben Glatt, and Anna H. R. Costa. 2017. MOO-MDP: An Object-Oriented Representation for Cooperative Multiagent Reinforcement Learning. *IEEE Transactions on Cybernetics* PP, 99 (2017), 1–13. https://doi.org/10.1109/TCYB.2017.2781130

[26] Felipe Leno Da Silva, Ruben Glatt, and Anna Helena Reali Costa. 2017. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 1100–1108.

[27] Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. 2018. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

[28] Richard S. Sutton. 1996. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Advances in Neural Information Processing Systems* (1996), 1038–1044.

[29] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction* (1st ed.). MIT Press, Cambridge, MA, USA.

[30] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. 2017. Automatic Curriculum Graph Generation for Reinforcement Learning Agents. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. San Francisco, CA, 2590–2596.

[31] Ming Tan. 1993. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the 10th International Conference on Machine Learning (ICML)*. 330–337.

[32] Adam Taylor, Ivana Dusparic, Edgar Galvan-Lopez, Siobhan Clarke, and Vinny Cahill. 2014. Accelerating Learning in Multi-Objective Systems through Transfer Learning. In *International Joint Conference on Neural Networks (IJCNN)*. 2298–2305. https://doi.org/10.1109/IJCNN.2014.6889438

[33] Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. 2008. Transferring Instances for Model-Based Reinforcement Learning. In *Machine Learning and Knowledge Discovery in Databases (Lecture Notes in Artificial Intelligence)*, Vol. 5212. 488–505.

[34] Matthew E. Taylor and Peter Stone. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10 (2009), 1633–1685. https://doi.org/10.1145/1577069.1755839

[35] Matthew E. Taylor, Peter Stone, and Yaxin Liu. 2007. Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. *Journal of Machine Learning Research* 8, 1 (2007), 2125–2167.

[36] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. 2007. Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 156–163.

[37] Nicholay Topin, Nicholas Haltmeyer, Shawn Squire, John Winder, Marie desJardins, and James MacGlashan. 2015. Portable Option Discovery for Automated Learning Transfer in Object-Oriented Markov Decision Processes. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. 3856–3864.

[38] Lisa Torrey and Matthew E. Taylor. 2013. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. In *Proceedings of 12th the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 1053–1060.

[39] Christopher J. Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.