# Automatic Object-Oriented Curriculum Generation for Reinforcement Learning

Felipe Leno da Silva and Anna Helena Reali Costa

Escola Politécnica of the University of São Paulo, Brazil
`{f.leno, anna.reali}@usp.br`

**Abstract.** Even though Reinforcement Learning (RL) is one of the most used techniques to train autonomous agents, learning complex tasks takes a very long time. One way to learn faster is dividing a complex task into several simple subtasks, building a *Curriculum* that guides Transfer Learning (TL) methods to reuse knowledge in a convenient sequence. However, the state-of-the-art in *Curriculum* Learning for RL does not take into account the used TL technique to build specialized *Curricula*, leaving the burden of a careful subtask selection to a human. We here rely on the Object-Oriented task description to guide both the *Curriculum* generation and knowledge reuse procedures, autonomously building object-based *Curricula*. Our initial experiments show that our proposal achieves a better performance while requiring less computation time when compared to the state-of-the-art technique.

**Keywords:** Reinforcement Learning, Transfer Learning, Curriculum Learning

## 1 Introduction

Although Reinforcement Learning (RL)[14] has been used to train agents to solve tasks autonomously, learning how to deliver a good performance takes very long time, especially in the challenging tasks for which autonomous agents are starting to be employed [8]. Since the classical RL algorithms are not scalable enough to be directly applied to such difficult tasks, a growing body of literature studies how to reuse past knowledge, like humans do, to accelerate the learning process [16].

More recently, inspired by the *Curriculum Learning* approach initially applied for Supervised Learning methods [1], Narvekar *et al.* [9] proposed to build *Curricula* 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 the hard task.

Narvekar has shown that a *Curriculum* can indeed be used to accelerate learning in the complex *Half Field Offense* [5] 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 [10]. Moreover, relying on such domain-specific and human-crafted knowledge is not always desirable for an autonomous agent.

Svetlik *et al.* [15] then proposed a method to autonomously estimate the *Curriculum*, requiring less domain-knowledge than in the original proposal. The *Curriculum* now is built as a graph, and some tasks can be solved in parallel by the agent or a set of agents. However, none of those works focused on how to reuse knowledge from the already solved source tasks, or on how to autonomously generate the set of source tasks.

In this work we intend to autonomously generate a *Curriculum* taking into account the TL method to better reuse the gathered knowledge. We use the Object-Oriented representation [3] to both generate the *Curriculum* and reuse knowledge, profiting from the generalization provided by it. We also make use of the representation to autonomously generate a set of source tasks. Our initial experiments indicate that our proposal requires only an intuitively-given task description to generate useful *Curricula*, and that our proposal is faster and simpler than the one proposed by Svetlik *et al.* Our source task generation procedure was also successful to autonomously build a set of source tasks that lead to an useful *Curriculum*. The remainder of this paper is organized as follows: In Section 2 we present the background knowledge required for understanding our work; in Section 3 we present our proposal adapting the *Curriculum* generation procedure to make better use of the Object-Oriented description; we then present our source task generation procedure in Section 4; in Section 5 we present our experimental setup and evaluation; and finally Section 6 concludes the paper and points towards future works.

## 2 Background

We here firstly present the related concepts in the RL area and then discuss the main *Curriculum Learning* works for RL.

### 2.1 Reinforcement Learning

A *Markov Decision Process* (MDP) is the most 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 $r = R(s, a, T(s))$. Each executed step results in a tuple of $\langle s, a, s', r \rangle$, that is the only feedback the agent has to learn how to solve the task (i.e., maximize 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 applying the Q-Learning [18] algorithm, which iteratively updates an estimate of action qualities for each state $Q : S \times A \rightarrow \mathbb{R}$. This estimate eventually converges to the optimal Q-function: $Q^*(s,a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, which 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.

The use of relational task descriptions can help to accelerate the learning process, as the state-action space is abstracted. The Object-Oriented representation [3, 12] can be used to describe the task in an intuitive manner and enable generalization opportunities. In *Object-Oriented MDPs* (OO-MDP) the state space is abstracted through the description of a set of classes $C = \{C_1, \dots, C_c\}$, where each class $C_i$ is composed of a set of *attributes* denoted as $Att(C_i) = \{C_i.b_1, \dots, C_i.b_b\}$. Each attribute $b_j$ has a *domain* $Dom(C_i.b_j)$, specifying the set of values this attribute can assume. $O = \{o_1, \dots, o_o\}$ 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 environment 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 relational task descriptions is especially helpful to *Transfer Learning* (TL) approaches [6], where the objective is to reuse previous knowledge to solve a task faster [16]. In the next Section we describe *Curriculum Learning*, an emerging method to accelerate learning by the use of TL techniques.

## 2.2 Curriculum Learning for RL

The main idea of *Curriculum Learning* is to, given a hard task to be solved (target task) $\mathcal{T}_t$, decompose $\mathcal{T}_t$ into several easier tasks $\mathcal{T}_{\mathbb{C}}$ that can be solved much faster and through TL learn all tasks in $\mathcal{T}_{\mathbb{C}}$ plus $\mathcal{T}_t$ faster than learning $\mathcal{T}_t$ from scratch.

In Narvekar *et al.*'s description [9], the *Curriculum* is a sequence of tasks within a single domain $\mathcal{D}$. $\mathcal{D}$ has a set of *degrees of freedom* $\mathcal{F}$, which are the task parameters. Any possible *MDP* that belongs to $\mathcal{D}$ can be generated 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 the target task $\mathcal{T}_t$. The steps to build and use a *Curriculum* are summarized in Algorithm 1. Given the target task $\mathcal{T}_t$, a set of candidates source tasks $\mathcal{T}$ is defined (manually

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

defined in all works so far). Then, the *Curriculum* $\mathbb{C}$ is built or given by a human, specifying a sequence of tasks $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t), \mathcal{T}_i \in \mathcal{T}_\mathbb{C}, \mathcal{T} \subseteq \mathcal{T}_\mathbb{C}$. All tasks are then solved in order and previous knowledge might be reused for each task.

**Algorithm 1** *Curriculum* generation and use

```
Require: target task $\mathcal{T}_t$.
1: $\mathcal{T} \leftarrow createTasks(\mathcal{T}_t)$
2: $\mathbb{C} \leftarrow buildCurriculum(\mathcal{T}, \mathcal{T}_t)$
3: Learn $\mathcal{T}_t$ using Curriculum $\mathbb{C}$.
```

Svetlik *et al.* [15] proposes to build the *Curriculum* as a graph, rather than a sequence of tasks. They also propose a method for automatic *Curriculum* generation and use Reward Shaping to transfer knowledge from one task to another. The main idea of their proposal is to build a graph of tasks according to a *transfer potential* metric, that estimates how much a source task would benefit the learning process of another. They calculate transfer potential as:

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

Where $\nu(\mathcal{T}_i, \mathcal{T}_j)$ defines the transfer potential value of two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, $|Q_{\mathcal{T}_i} \cap Q_{\mathcal{T}_j}|$ is the number of Q-values that those two tasks have in common, and $|S_{\mathcal{T}_j}|$ is the size of the state space of $\mathcal{T}_j$. A *Curriculum* graph is then generated by including all source tasks that have a transfer potential to $\mathcal{T}_t$ higher than a threshold parameter $\epsilon$.

While both works have shown that building *Curricula* may be beneficial to learning agents, none of them evaluate the advantage of generalization when transferring knowledge between *Curriculum* tasks. Moreover, the set of source tasks is manually given in all works, and an automated source task generation procedure was listed as one of the open problems for *Curriculum* approaches [15]. In the next Sections we describe our proposal to autonomously 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 the generalization provided by OO-MDPs to facilitate the contruction and use of a *Curriculum*. We use the Object-Oriented description to autonomously generate *Curricula*, and show that our proposal uses the extra domain knowledge received in the task description to build a better *Curriculum* using less computational power when compared to Svetlik *et al.*'s proposal [15].

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:

$$\mathcal{T}_i = \langle \mathcal{C}, \mathcal{O}^{\mathcal{T}_i}, S_0^{\mathcal{T}_i}, \mathcal{F}^{\mathcal{T}_i}, A^{\mathcal{T}_i}, T^{\mathcal{T}_i}, R^{\mathcal{T}_i} \rangle \tag{2}$$

where $\mathcal{C}$ is the set of classes, $\mathcal{O}_{\mathcal{T}_i}$ is the set of objects in $\mathcal{T}_i$, $S_0^{\mathcal{T}_i}$ is the probability distribution of initial states for $\mathcal{T}_i$, $\mathcal{F}^{\mathcal{T}_i}$ is the set of degrees of freedom not related to objects (e.g., size of the grid in a *Gridworld* domain), $A^{\mathcal{T}_i}$ 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. As in [15], we consider a *Curriculum* as a graph of tasks, and the *Curriculum* is generated by following Algorithm 2. The *Curriculum* is composed of the set of vertexes $\mathcal{V}$ and the set of edges $\mathcal{E}$. The first step is to split the set of source tasks $\mathcal{T}$ in groups according to their parameters (line 2). The grouping procedure is fully described in Algorithm 3 and the main idea is to build a set of candidate tasks $\mathcal{T}_{\mathbb{C}}$ according to their transfer potential, that we calculate based on the Object-Oriented description as:

**Algorithm 2** Automatic Curriculum Generation

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

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

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

**Algorithm 3** groupTasks

```
Require: set of source tasks $\mathcal{T}$, target task $\mathcal{T}_t$, threshold for task inclusion $\epsilon$.
 1: $\mathcal{T}_{\mathbb{C}} \leftarrow \emptyset$
 2: $G \leftarrow \emptyset$
 3: for $\forall \mathcal{T}_i \in \mathcal{T}$ do
 4:     if $\nu(\mathcal{T}_i, \mathcal{T}_t, \mathcal{T}_{\mathbb{C}}) > \epsilon$ then                              ▷ Eq. (3)
 5:         if $\nexists g : g \in G$ and $g.features = \mathcal{T}_i.features$ then
 6:             $G \leftarrow G \cup group(\mathcal{T}_i.features)$               ▷ Creates a new group
 7:         $g = \{g \in G | g.features = \mathcal{T}_i.features\}$
 8:         $g \leftarrow g \cup \mathcal{T}_i$
 9:         $\mathcal{T}_{\mathbb{C}} \leftarrow \mathcal{T}_{\mathbb{C}} \cup \mathcal{T}_i$
10: Sort tasks within groups by $\nu$
11: return $(G, \mathcal{T}_{\mathbb{C}})$,
```

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 train first in smaller tasks. Here, the similarity between tasks is calculated as:

$$sim_{\mathbb{C}}(\mathcal{T}_i, \mathcal{T}_j) = \sum_{C_i \in C} \frac{|\mathcal{O}_{C_i}^{\mathcal{T}_i} \cap \mathcal{O}_{C_i}^{\mathcal{T}_j}|}{\max(|\mathcal{O}_{C_i}^{\mathcal{T}_i}|, |\mathcal{O}_{C_i}^{\mathcal{T}_j}|)} + \frac{|S_s \cap S_t|}{|S_t|} \tag{4}$$

where $\mathcal{T}_i$ and $\mathcal{T}_j$ are tasks from the same domain, $C_i$ is one class from the set $C$, and $\mathcal{O}_{C_i}^{\mathcal{T}_i} \cap \mathcal{O}_{C_i}^{\mathcal{T}_j}$ is the set of objects that belong to class $C_i$ and have the same attribute values in both $\mathcal{T}_i$ and $\mathcal{T}_j$. Based on these metrics, the return of Algorithm 3 is a group for each possible task parameters with transfer potential higher than the threshold $\epsilon$. (Alg 3 line 4).

After the set of task groups is built, we define the edges of the *Curriculum* as proposed by Svetlik *et al.* [15]. Firstly we search for tasks that have a high transfer potential between themselves within the same task group (Alg. 2 lines 4-8), then we search for tasks that belong to different groups, as long as the features of the source task are contained in the features of $\mathcal{T}_t$ (lines 9-14). Finally, we create an edge from the tasks with outdegree *zero* to the target task (line 16).

In order to use this *Curriculum* for learning, an unsolved task with indegree *zero* must be selected and used for training until a stopping criterion. Then, all the edges from the selected task are removed from the graph and this process is repeated until the target task is selected (that will be the last one).

After a task is selected for training, a procedure for reusing knowledge from the previously solved tasks must be executed. The TL literature has studied many ways of reusing knowledge from one task to another, such as transferring low-level interactions with the environment [7], policies [4], value or $Q$ functions [17], abstract or partial policies [6], action suggestions [13], and heuristics or biases for a more effective exploration [2], each of them presenting benefits over learning from scratch. We use a method based on value function reuse as performed by Narvekar *et al.* [9], but taking advantage of the Object-Oriented representation. We first build Probabilistic Inter-TAsk Mappings (PITAM) as proposed in [11] to define a mapping between the Object-Oriented states in the source and target tasks. A PITAM is a mapping between states where: $\mathscr{P}_{\mathcal{T}_s,\mathcal{T}_t} : S_{\mathcal{T}_t} \times S_{\mathcal{T}_s} \rightarrow [0,1]$. This mapping is calculated according to a similarity metric, which we define as: $sim_{PITAM}(s_t, s_s) = |\mathcal{O}_{s_t} \cap \mathcal{O}_{s_s}|$. That is, states that have no object in common have a mapping with zero probability, and as higher the number of common objects, as higher will be the mapping probability. This similarity value is calculated for all the state space in the source task and the PITAM probability is normalized as $\sum_{s \in S_{\mathcal{T}_s}} \mathcal{P}_{\mathcal{T}_s,\mathcal{T}_t}(s_t, s) = 1$. Although calculating exact PITAM probabilities is computationaly 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 as:

$$Q_{\mathcal{T}_t}(s_t, a_t) \leftarrow \frac{\sum_{\mathcal{T}_i \in \mathbb{C}_{\mathcal{T}_t}} \sum_{s_i \in S_{\mathcal{T}_i}} \mathscr{P}_{\mathcal{T}_i,\mathcal{T}_t}(s_t, s_i) Q_{\mathcal{T}_i}(s_i, a_t)}{|\mathbb{C}_{\mathcal{T}_t}|}, \tag{5}$$

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-centred 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 the distances between the agent and objects are used (agent-centred representation), it is much easier to find state correspondences.

Using the *Curriculum* and this transfer procedure, the agent is then expected to learn faster than when learning from scratch. In the next Section we describe our proposal to autonomously generate source tasks.

## 4 Object-Oriented Autonomous Source Task Generation

As *Curriculum Learning* aims at solving portions of the target task first to accelerate learning, the success of a *Curriculum* depends on: (i) a proper set of source tasks; (ii) a proper ordering of those tasks; (iii) the efficacy of the chosen TL algorithm; (iv) a good stopping criterion to identify when to switch tasks. Even though Svetlik *et al.* [15] had proposed a method to autonomously define the sequence of tasks, autonomously defining the set of source tasks was still an open problem [15, 9, 10].

We here propose a method to autonomously generate source tasks by using the object-oriented representation, requiring much less human effort than as in previous works in which the set of source tasks must be manually given.

Algorithm 4 fully describes our source task generation procedure (which fits as a *createTask* function in Alg. 1 line 1). At first we define the set $F_{simple}$, which is a set of possible values of the *degrees of freedom* in the target task. As the object-oriented representation has no information about $\mathcal{F}$, we rely on a human's knowledge to specify the $simplify$ function. Then, we define the set of objects belonging to the class with fewer objects $C_{min}$ (lines 2–3). We then create $o_{min}$ tasks, each of them containing from 0 to $o_{min}$ objects from $C_{min}$ (line 5), one possible set of values from $F_{simple}$ (line 6), and a random number of objects from the other classes (line 9). 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 initial state in the target task. The action space, transition function, and reward functions for the new task are then defined through the *actionSpace*, *transitionFunction*, and *rewardFunction* functions, and the task is finally added to the set of source tasks $\mathcal{T}$. This process is repeated multiple times according to a parameters $n_{rep}$.

**Algorithm 4** $createTasks(\mathcal{T}_t, \mathcal{F}^{\mathcal{T}_t}, n_{rep})$

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

This procedure can be used to generate a set of source tasks when a human is unavailable to provide it. However, this procedure is not valid for all possible domains, because we change the number of objects without knowledge about the transition function. When using this procedure, the designer must ensure that solvable tasks are generated (it might be necessary to set a minimum number of objects of a given class or to create the initial state in a domain-dependent way).

In the next Section we present our experimental evaluation.

## 5 Experimental Evaluation

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

## 5.1 Experimental Setup

As experimental domain, we have chosen the *Gridworld* domain, as proposed by Sevtlik *et al.* [15]. 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 $C = \{Pit, Fire, Treasure\}$, each of them with $x$ 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 degrees of freedom for this domain are the size of the grid $\mathcal{F} = \{sizeX, sizeY\}$. At each of the executed steps, the agent observes one of the following rewards: (i)**+200** for collecting the treasure; (ii) **-250** for getting next to a fire; (iii) **-500** for getting into a fire; (iv) **-2500** for falling into a pit; and (v) **-1** if nothing else happened.

The final target task is illustrated in Figure 1, and contains 8 pits, 11 fires, and 1 treasure. As in [15], we generate a set of source tasks by reducing the number of objects and/or reducing the size of the grid. In total, we had $|\mathcal{T}| = 27$ source tasks. During training, all source tasks given by the agent *Curriculum* were executed until the agent presents the same cumulative reward for 2 consecutive episodes, or 10 learning episodes were carried out.

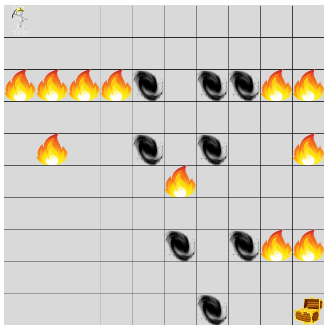

**Fig. 1.** An illustration of the target task in the Gridworld domain.

We compare our proposal with Svetlik's [15] *Curriculum* generation procedure and the regular Q-Learning. Svetlik's proposal is implemented with Reward Shaping as in the original paper. The comparison metric is the cumulative reward when trying to solve the target task. The threshold parameter for *Curriculum* generation was set $\epsilon = 4$ for our proposal and $\epsilon = 15$ for Svetlik's. We also evaluate our proposal with autonomously generated source tasks, using $\epsilon = 1$ and $n_{rep} = 10$. Note that the transfer potential in Svetlik's proposal is calculated based on numbers of Q-table entries, that are orders of magnitude bigger than numbers of objects (used in our proposal). Hence, the $\epsilon$ threshold should be scaled accordingly.

The time taken by each algorithm to generate a *Curriculum* was stored. Given the same target task, we variate the number of source tasks (randomly generated) and evaluate the computation time to each algorithm.

## 5.2 Results

Hereafter we refer to results achieved by each algorithm as: *Regular Q-Learning*: **No Curriculum**; *Svetlik's Proposal*: **Svetlik**; *Our proposal with a manually given set of source tasks*: **OO - Given**; *Our proposal with an autonomously generated set of source tasks* (Section 4): **OO - Generate**.

Figure 2 shows the experimental results in the *Gridworld* domain. Although very good results were shown in [15] using Svetlik's proposal, we found out that their proposal is quite sensitive to parameters and to the provided source tasks. Even though we had a similar number of source tasks and parameters as in [15] (27 source tasks and 6-task *Curriculum* against 30 source tasks and 5-task *Curriculum*), we could not achieve a better performance than *No Curriculum* using their proposal. In its turn, *OO-Given* presents a better performance than *No Curriculum* under the same situation, showing the advantages of our proposal to accelerate learning without the need of extreme care when engineering the source task set and setting parameters. *OO-Generated* presented a slightly worst performance than *No Curriculum* between 1850 and 2500 learning steps. However, *OO-Generated* then achieves the optimal policy very fast, even before than *OO-Given*. Table 1 shows the number of learning steps taken by each algorithm to achieve the optimal policy. Notice that our approach achieves it faster than both *No Curriculum* and *Svetlik*. Those results show that our proposal is promising for building useful *Curricula*, even if a manually given set of source tasks is not available.

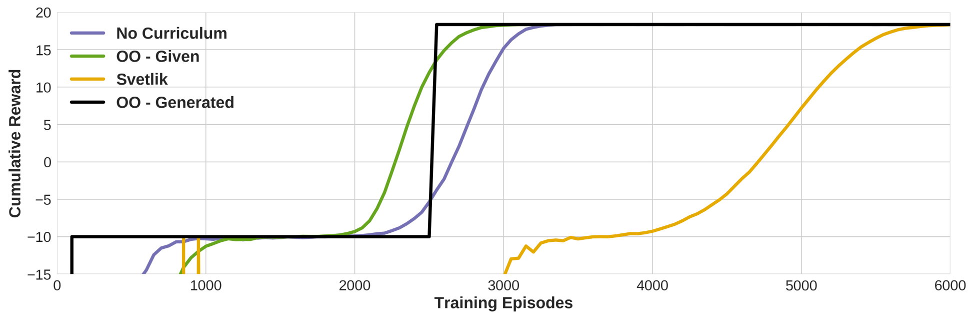

**Fig. 2.** The average discounted cumulative rewards observed in 10, 000 repetitions of the experiment. Steps used to learn source tasks are also considered in the graph.

Figures 3, 4, and 5 show the resulting *Curriculum* when using *OO-Generated*, *OO-Given*, and *Svetlik* respectively. Notice that the same source tasks were given to *OO-Given* and *Svetlik*, while *OO-Generated* generates its own source tasks.

**Table 1.** Average number of learning steps taken to achieve the optimal policy (average of 10000 executions).

| **Alg.** | Steps |
|---|---|
| **No Curriculum** | 3250 |
| **OO- Given** | 2900 |
| **OO- Generated** | 2550 |
| **Svetlik** | 6100 |

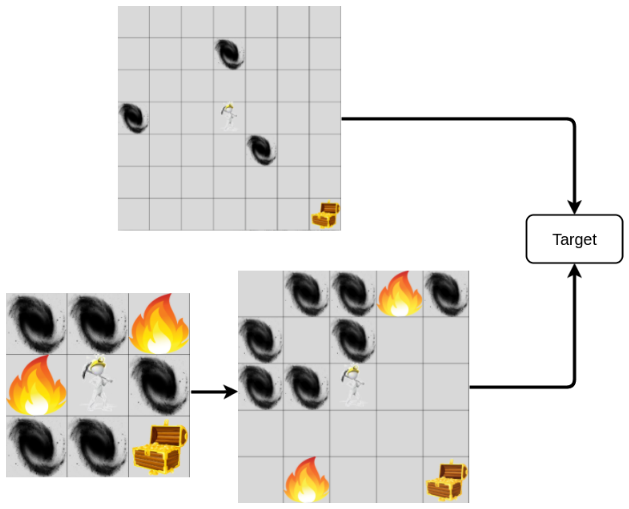

**Fig. 3.** The resulting *Curriculum* graph when using *OO-Generated*.

*OO-Generated* creates tasks that seem unintuitive at first sight, but are still useful for learning. While in the two lower tasks the agent learns to avoid pits and fires, the upper task helps the agent to learn how to reach the treasure.

Since *OO-Given* uses manually-given tasks, the outcome is more intuitive. The source tasks are mostly smaller versions of the target task, in which a small number of objects exist to help the agent learn how to interact with them.

*Svetlik* however prefers tasks without objects. Most of the chosen source tasks have no objects, which doubtlessly results in tasks that can be learned faster. But as the target task consists in avoiding fires and pits, those source tasks are not very helpful and result in the worst performance among the evaluated algorithms.

Table 2 shows the required time to generate a *Curriculum* for each algorithm. Our proposal is much faster than Svetlik's for all the evaluated sizes of source task sets. The difference in computation time is mostly due to the calculation of the transfer potential between tasks (Equations (1) and (3)). While computing $|Q_{\mathcal{T}_i} \cup Q_{\mathcal{T}_j}|$ is very computationally intensive, we estimate the transfer potential through the Object-Oriented description in a simpler manner. Notice also that

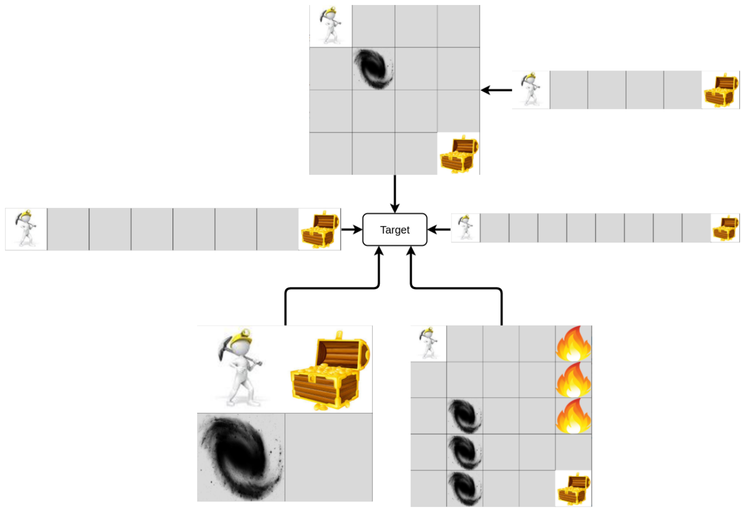

**Fig. 4.** The resulting *Curriculum* graph when using *OO-Given*.

the time taken to generate the set of source tasks is negligible when summed with the *Curriculum* generation time.

**Table 2.** Computation time to generate a *Curriculum*.

| | $\|\mathcal{T}\| = 10$ | $\|\mathcal{T}\| = 50$ | $\|\mathcal{T}\| = 100$ | $\|\mathcal{T}\| = 200$ | $\|\mathcal{T}\| = 500$ |
|---|---|---|---|---|---|
| **OO- Given** | 1ms | 13ms | 50ms | 205ms | 1.2s |
| **OO- Generated** | 1.6ms | 14ms | 50ms | 205ms | 1.2s |
| **Svetlik** | 54ms | 440ms | 1.4s | 4.4s | 26.7s |

Our experimental evaluation has shown that using the Object-Oriented description to generate *Curricula* presents benefits in both performance and computation time. Moreover, it is possible to autonomously generate the set of source task and still build an useful *Curriculum*, without degradation of the computation time.

## 6 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 proposals so far require carefully extracted domain knowledge to work, in the form of parameter selections and construction of a source task base. We here propose an Object-Oriented Curriculum generation procedure that builds a *Curriculum* graph by using an intuitively-given Object-Oriented task description. We also

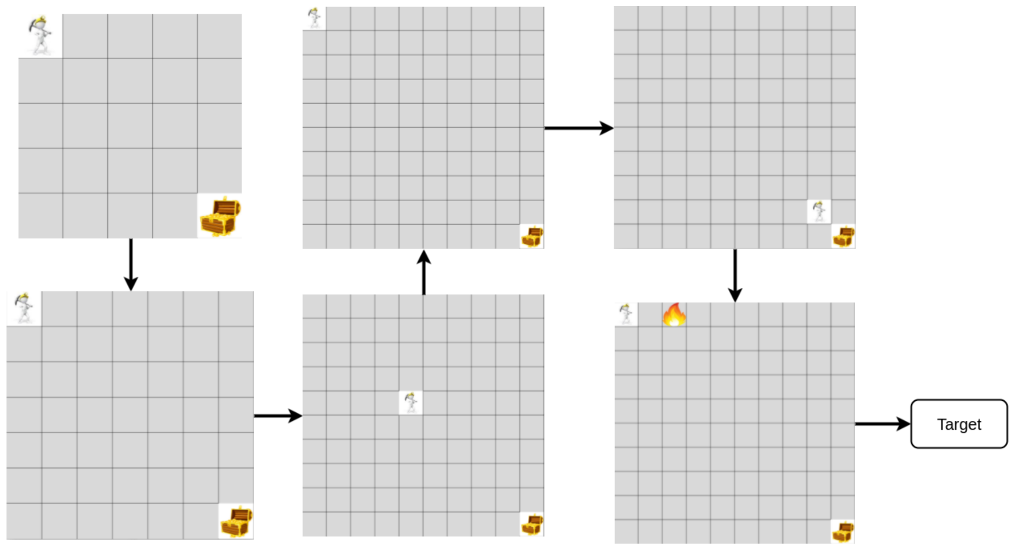

**Fig. 5.** The resulting *Curriculum* graph when using *Svetlik*.

propose a procedure to autonomously generate the set of source tasks to a *Curriculum*, requiring less domain-specific knowledge than in the previous works. We have shown in our experiments that our proposal outperforms previous works in both performance and *Curriculum* Generation speed. Future works can also evaluate if Object-Oriented Curricula can help humans to better understand the agent learning process and build better Curricula.

## Acknowledgements

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

## References

1. Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48, 2009.
2. R. A. Bianchi, L. A. C. Jr., P. E. Santos, J. P. Matsuura, and R. L. de Mantaras. Transferring Knowledge as Heuristics in Reinforcement Learning: A Case-Based Approach. *Artificial Intelligence*, 226:102 – 121, 2015.
3. C. Diuk, A. Cohen, and M. L. Littman. An Object-oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 240–247, 2008.
4. F. Fernández and M. Veloso. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 720–727, 2006.
5. M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone. Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, May 2016.
6. M. L. Koga, V. F. da Silva, and A. H. R. Costa. Stochastic Abstract Policies: Generalizing Knowledge to Improve Reinforcement Learning. *IEEE Transactions on Cybernetics*, 45(1):77–88, 2015.
7. A. Lazaric, M. Restelli, and A. Bonarini. Transfer of Samples in Batch Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 544–551, 2008.
8. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.
9. S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone. Source Task Creation for Curriculum Learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 566–574, 2016.
10. B. Peng, J. MacGlashan, R. Loftin, M. L. Littman, D. L. Roberts, and M. E. Taylor. An Empirical Study of Non-expert Curriculum Design for Machine Learners. In *Proceedings of the IJCAI Interactive Machine Learning Workshop*, 2016.
11. F. L. d. Silva and A. H. R. Costa. Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description. In *Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL)*, 2017.
12. F. L. d. Silva, R. Glatt, and A. H. R. Costa. Object-Oriented Reinforcement Learning in Cooperative Multiagent Domains. In *Proceedings of the 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 19–24, 2016.
13. F. L. d. Silva, R. Glatt, and A. H. R. Costa. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1100–1108, 2017.
14. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.
15. M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone. Automatic Curriculum Graph Generation for Reinforcement Learning Agents. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, pages 2590–2596, San Francisco, CA, February 2017.
16. M. E. Taylor and P. Stone. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

17. M. E. Taylor, P. Stone, and Y. Liu. Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.
18. C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.