LEARNING OF SOPHISTICATED CURRICULUMS BY VIEWING THEM AS GRAPHS OVER TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Curriculum learning consists in learning a difficult task by first training on an easy version of it, then on more and more difficult versions and finally on the difficult task. To make this learning efficient, given a curriculum and the current learning state of an agent, we need to find what are the good next tasks to train the agent on.

Teacher-Student algorithms assume that the good next tasks are the ones on which the agent is making the fastest progress or digress. We first simplify and improve them. However, two problematic situations where the agent is mainly trained on tasks it can't learn yet or it already learnt may occur.

Therefore, we introduce a new algorithm using min max ordered curriculums that assumes that the good next tasks are the ones that are learnable but not learnt yet. It outperforms Teacher-Student algorithms on small curriculums and significantly outperforms them on sophisticated ones with numerous tasks.

1 INTRODUCTION

Curriculum learning. An agent with no prior knowledge can learn a lot of tasks by reinforcement, i.e. by reinforcing (taking more often) actions that lead to higher reward. But, for some very hard tasks, it is impossible. Let's consider the following task:

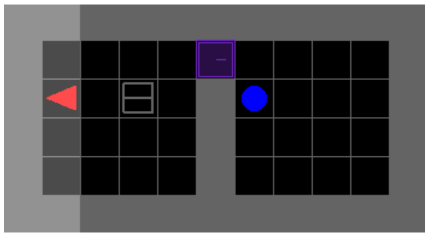


Figure 1: The agent (in red) receives a reward of 1 when it picks up the blue ball in the adjacent room. To do so, it has to first open the gray box, take the key inside and then open the locked door.

This is an easy task for humans because we have prior knowledge: we know that a key can be picked up, that we can open a locked door with a key, etc... However, most of the time, the agent starts with no prior knowledge, i.e. it starts by acting randomly. Therefore, it has a probability near 0 of achieving the task in a decent number of time-steps, so it has a probability near 0 of getting reward, so it can't learn the task by reinforcement.

One solution to still learn this task is to do *curriculum learning* (Bengio et al. (2009)), i.e. to first train the agent on an easy version of the task, where it can get reward and learn, then train on more and more difficult versions using the previously learnt policy and finally, train on the difficult task.

Learning by curriculum may be decomposed into two parts:

1. Defining the *curriculum*, i.e. the set of tasks the agent may be trained on.
2. Defining the *program*, i.e. the sequence of curriculum's tasks it will be trained on.

These two parts can be done online, during training.

Curriculum learning algorithms. Defining a curriculum and a program can be done manually, e.g. by defining a hand-chosen performance threshold for advancement to the next task (Zaremba & Sutskever (2014); Wu & Tian (2017)).

However, if an efficient algorithm is found, it may save us a huge amount of time in the future. Besides, efficient (and more efficient than humans) algorithms are likely to exist because they can easily mix in different tasks (what is hard for humans) and then:

- avoid catastrophic forgetting by continuously retraining on easier tasks;
- quickly detect learnable but not learnt yet tasks.

Hence, it motivates the research of curriculum learning algorithms.

Curriculum learning algorithms can be grouped into two categories:

1. *curriculum algorithms*: algorithms that define the curriculum;
2. *program algorithms*: algorithms that define the program, i.e. that decide, given a curriculum and the learning state of the agent, what are the good next tasks to train the agent on.

In this paper, we will focus on program algorithms, in the reinforcement learning context. Recently, several such algorithms emerged, focused on the notion of *learning progress* (Matiisen et al. (2017); Graves et al. (2017); Fournier et al. (2018)). Matiisen et al. (2017) proposed four algorithms (called Teacher-Student) based on the assumption that the good next tasks are the ones on which the agent is making the fastest progress or digress.

We first simplify and improve Teacher-Student algorithms (section 4). However, even improved, two problematic situations where the agent is mainly trained on tasks it can't learn or it already learnt may occur. Therefore, we introduce a new algorithm (section 5), focused on the notion of *mastering rate*, based on the assumption that the good next tasks are the ones that are learnable but not learnt yet.

We show that this algorithm outperforms Teacher-Student algorithms on small curriculums and significantly outperforms them on sophisticated ones with numerous tasks.

2 BACKGROUND

2.1 CURRICULUM LEARNING

First, let's recall some general curriculum learning notions defined in Graves et al. (2017). A *curriculum* $\mathcal{C}$ is a set of tasks $\{c_1, ..., c_n\}$. A *sample* x is a drawn from one of the tasks of $\mathcal{C}$. A *distribution* d *over* $\mathcal{C}$ is a family of non-negative summing to one numbers indexed by $\mathcal{C}$, i.e. $d = (d_c)_{c \in \mathcal{C}}$ with $d_c \geq 0$ and $\sum_{c \in \mathcal{C}} d_c = 1$. Let $\mathcal{D}^{\mathcal{C}}$ be the set of distributions over $\mathcal{C}$. A *program* [1] $d : \mathbb{N} \to \mathcal{D}^{\mathcal{C}}$ is a time-varying sequence of distributions over $\mathcal{C}$.

Without loss of generality, we propose to perceive a distribution d over tasks $\mathcal{C}$ as coming from an attention a over tasks, i.e. $d := \Delta(a)$. An *attention* a *over* $\mathcal{C}$ is a family of non-negative numbers indexed by $\mathcal{C}$, i.e. $a = (a_c)_{c \in \mathcal{C}}$ with $a_c \geq 0$. Intuitively, a_c represents the interest given to task c. Let $\mathcal{A}^{\mathcal{C}}$ be the set of attentions over $\mathcal{C}$.

$\Delta : \mathcal{A}^{\mathcal{C}} \to \mathcal{D}^{\mathcal{C}}$ is called a *distribution converter*. In Matiisen et al. (2017); Graves et al. (2017); Fournier et al. (2018), several distribution converters are used (without using this terminology):

- the *argmax* distribution converter: $\Delta^{Amax}(a)_c := \begin{cases} 1 & \text{if } c = \text{argmax}_{c'} a_{c'} \\ 0 & \text{otherwise} \end{cases}$.

  A greedy version of it is used in Matiisen et al. (2017), i.e. $\Delta^{GAmax}(a) := \varepsilon \cdot u + (1 - \varepsilon) \cdot \Delta^{Amax}(a)$ with $\varepsilon \in [0, 1]$ and u the uniform distribution over $\mathcal{C}$.

[1] What we call "program" is called "syllabus" in Graves et al. (2017). We prefer the primer word because it is more explicit and can't be confused with "curriculum".

- the *exponential* distribution converter: $\Delta^{Exp}(a)_c := \frac{\exp(a_c)}{\sum_{c'} \exp(a_{c'})}$ (used in Graves et al. (2017)).
- the *Boltzmann* distribution converter: $\Delta^{Boltz}(a)_c := \frac{\exp(a_c/\tau)}{\sum_{c'} \exp(a_{c'}/\tau)}$ (used in Matiisen et al. (2017)).
- The *powered* distribution converter: $\Delta^{Pow}(a)_c := \frac{(a_c)^p}{\sum_{c'} (a_{c'})^p}$ (used in Fournier et al. (2018)).

An *attention function* $a : \mathbb{N} \to \mathcal{A}^{\mathcal{C}}$ is a time-varying sequence of attentions over $\mathcal{C}$. A program d can be rewritten using this notion: $d(t) := \Delta(a(t))$ for a given attention converter Δ.

Finally, a *program algorithm* can be defined as follows:

Algorithm 1: A program algorithm

```
input: A curriculum $\mathcal{C}$ ;
       An agent $\mathbf{A}$;
for $t \leftarrow 1$ to $T$ do
    Compute $a(t)$ ;
    Deduce $d(t) := \Delta(a(t))$ ;
    Draw a task $c$ from $d(t)$ and then a sample $x$ from $c$ ;
    Train $\mathbf{A}$ on $x$ and observe return $r_t$;
```

A program algorithm needs to internally define an attention function $\mathcal{A}$ and an attention converter Δ. All the program algorithms in Matiisen et al. (2017); Graves et al. (2017); Fournier et al. (2018) fit this formalism.

2.2 Teacher-Student program algorithms

The Teacher-Student paper (Matiisen et al. (2017)) presents four attention functions called *Online*, *Naive*, *Window* and *Sampling*. They are all based on the idea that the attention must be given by the absolute value of an estimate of the learning progress over tasks, i.e. $\mathcal{A}(t) := |\beta(t)|$ where $\beta_c(t)$ is an estimate of the learning progress of the agent $\mathbf{A}$ on task c.

For the Window attention function, they first estimate the "instantaneous" learning progress of the agent $\mathbf{A}$ on task c by the slope $\beta_c^{Linreg}(t)$ of the linear regression of the points $(t_1, r_{t_1}), ..., (t_K, r_{t_K})$ where $t_1, ..., t_K$ are the K last time-steps when the agent was trained on a sample of c and where r_{t_i} is the return got by the agent at these time-steps. From this instantaneous learning progress, they define β_c as the weighted moving average of β_c^{Linreg}, i.e. $\beta_c(t+1) := \alpha\beta_c^{Linreg}(t) + (1-\alpha)\beta_c(t)$.

For all the algorithms, a Boltzmann or greedy argmax distribution converter is used. For example, here is the GAmax Window program algorithm proposed in the paper:

Algorithm 2: GAmax Window algorithm

```
input: A curriculum $\mathcal{C}$ ;
       An agent $\mathbf{A}$;
$\beta := 0$ ;
for $t \leftarrow 1$ to $T$ do
    $a := |\beta|$ ;
    $d := \Delta^{GAmax}(a)$ ;
    Draw a task $c$ from $d$ and then a sample $x$ from $c$ ;
    Train $\mathbf{A}$ on $x$ and observe return $r_t$ ;
    $\beta_c^{Linreg}$ := slope of lin. reg. of $(t_1, r_{t_1}), ..., (t_K, r_{t_K})$ ;
    $\beta_c := \alpha\beta_c^{Linreg} + (1-\alpha)\beta_c$ ;
```

3 Experiment settings

Before presenting simplifications and improvements of the Teacher-Student algorithms (section 4) and then introducing a new algorithm (section 5), we present the experiment settings.

Three curriculums were used to evaluate the algorithms, called *BlockedUnlockPickup*, *KeyCorridor* and *ObstructedMaze* (see appendix A for screenshots of all tasks). They are all composed of Gym MiniGrid environments (Chevalier-Boisvert & Willems (2018)).

These environments are partially observable and at each time-step, the agent receives a $7 \times 7 \times 3$ image (figure 1) along with a textual instruction. Some environments require language understanding and memory to be efficiently solved, but the ones chosen in the three curriculums don't.

The agent gets a reward of $1 - \frac{n}{n_{max}}$ when the instruction is executed in n steps with $n \leq n_{max}$. Otherwise, it gets a reward of 0.

4 Simplification & improvement of Teacher-Student algorithms

4.1 Recommendations

Before simplifying and improving Teacher-Student algorithms, here are some suggestions about which distribution converters and attention functions of the Teacher-Student paper to use and not to use.

First, in this paper, two distribution converters are proposed: the greedy argmax and the Boltzmann ones. We don't recommend to use the Boltzmann distribution converter because τ is very hard to tune in order to get a distribution that is neither deterministic nor uniform.

Second, four attention functions are proposed: the Online, Naive, Window and Sampling ones. We don't recommend to use:

- the Naive attention function because it is a naive version of the Window one and performs worst (see figure 5 in Matiisen et al. (2017));
- the Sampling attention function because it performs worst than the Window one (see figure 5 in Matiisen et al. (2017)). Moreover, the reason it was introduced was to avoid hyperparameters but it still require to tune a ε to avoid deterministic distributions (see algorithm 8 in Matiisen et al. (2017))...

It remains the Online and Window attention functions. But, the Online one is similar to the Window one when $K = 1$.

Finally, among all what is proposed in this paper, we only recommend to use the Window attention function with the greedy argmax distribution converter, i.e. to use the GAmax Window algorithm (algorithm 2). This is the only one we will consider in the rest of this section.

4.2 Simplifications & Improvements

Now, let's see how we can simplify and improve the GAmax Window algorithm.

Firstly, we simplify the Window attention function by removing the weighted moving average (i.e. by taking $\beta_c := \beta_c^{Linreg}$) without impacting performances (figures 2 and 3). We call *Linreg* this new attention function.

Secondly, we introduce the *proportional* distribution converter: $\Delta^{Prop}(a)_c := \frac{a_c}{\sum_{c'} a_{c'}}$. This corresponds to the powered distribution converter when $p = 1$. We then replace the greedy argmax distribution converter by a greedy proportional one, and improve performances (figures 2 and 3).

This gives this new program algorithm:

Algorithm 3: GProp Linreg algorithm

```
input: A curriculum $\mathcal{C}$ ;
       An agent $\mathbf{A}$;
$\beta^{Linreg} := 0$ ;
for $t \leftarrow 1$ to $T$ do
    $a := |\beta^{Linreg}|$ ;
    $d := \Delta^{GProp}(a)$ ;
    Draw a task $c$ from $d$ and then a sample $x$ from $c$ ;
    Train $\mathbf{A}$ on $x$ and observe return $r_t$ ;
    $\beta_c^{Linreg}$ := slope of lin. reg. of $(t_1, r_{t_1}), ..., (t_K, r_{t_K})$ ;
```

In the rest of this article, this algorithm will be referred as our "baseline".

4.3 Results

The two following figures show that:

- the GAmax Linreg algorithm performs similarly to the GAmax Window algorithm, as asserted before. It even seems a bit more stable because the gap between the first and last quartile is smaller.
- the GProp Linreg algorithm performs better than the GAmax Linreg and GAmax Window algorithm, as asserted before.

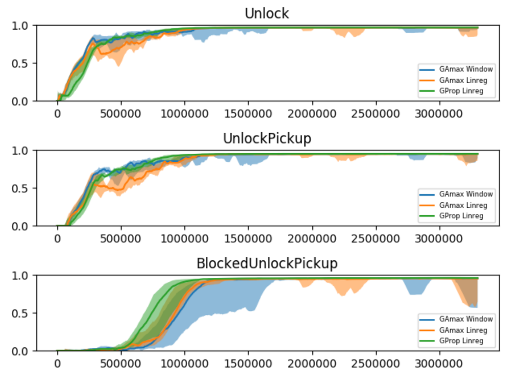


Figure 2: GAmax Window, GAmax Linreg and GProp Linreg algorithms were each tested with 10 different agents (seeds) on the BlockedUnlockPickup curriculum. The median return during training, between the first and last quartile, are plotted.

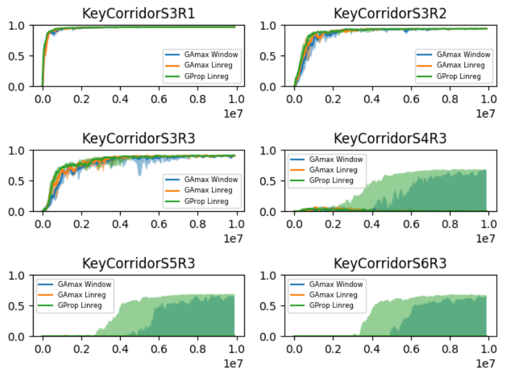


Figure 3: GAmax Window, GAmax Linreg and GProp Linreg algorithms were each tested with 10 different agents (seeds) on the KeyCorridor curriculum. The median return during training, between the first and last quartile, are plotted.

5 A Mastering Rate based algorithm (MR algorithm)

Algorithms introduced in Matiisen et al. (2017); Graves et al. (2017); Fournier et al. (2018), and in particular Teacher-Student algorithms and the baseline algorithm, are focused on the notion of *learning progress*, based on the assumption that the good next tasks are the ones on which the agent is making the fastest progress or digress. However, two problematic situations may occur:

1. The agent may be mainly trained on tasks it **already learnt**. The frame B of the figure 4 shows that, around time-step 500k, the agent already learnt Unlock and UnlockPickup but is still trained 90% of the time on them, i.e. on tasks it already learnt.
2. It may be mainly trained on tasks it **can't learn yet**. The more the curriculum has tasks, the more it occurs:

- The frame A of the figure 4 shows that, initially, the agent doesn't learn Unlock but is trained 66% of the time on UnlockPickup and BlockedUnlockPickup, i.e. on tasks it can't learn yet.
- The figure 5 shows that agents spend most of the time training on the hardest task of the ObstructedMaze curriculum whereas they have not learnt yet the easy tasks.

.

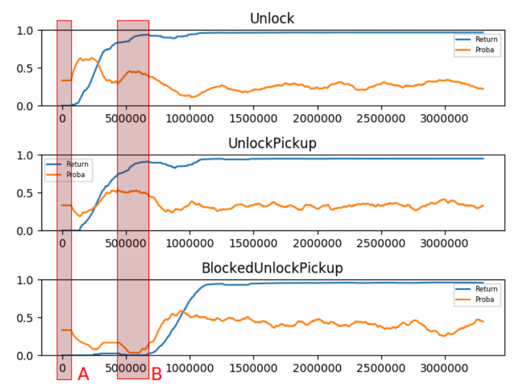


Figure 4: The agent with seed 6 was trained on the BlockedUnlockPickup curriculum using the baseline algorithm. The return and probability during training are plotted. Two particular moments of the training are framed.

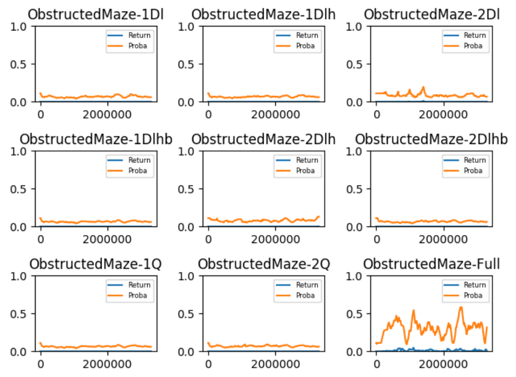


Figure 5: 10 agents (seeds) were trained on the ObstructedMaze curriculum using the baseline algorithm. The mean return and mean probability during training are plotted.

To overcome these issues, we introduce a new algorithm, focused on the notion of *mastering rate*, based on the assumption that the good next tasks to train on are the ones that are learnable but not learnt yet. Why this assumption? Because it can't be otherwise. Mainly training on learnt tasks or not learnable ones is a lost of time. This time must be spent training on respectively harder or easier tasks.

In subsection 5.1, we first define what are learnt and learnable tasks and then, in subsection 5.2, we present this new algorithm.

5.1 LEARNT & LEARNABLE TASKS

Learnt tasks. A *min-max curriculum* is a curriculum $\mathcal{C} = \{c_1, ..., c_n\}$ along with:

- a family $(m_c^1)_{c \in \mathcal{C}}$ where m_c^1 is an estimation of the minimum mean return the agent would get on task c. It should be higher than the true minimum mean return.
- and a family $(M_c^1)_{c \in \mathcal{C}}$ where M_c^1 is an estimation of the maximum mean return the agent would get on task c. It should be lower than the true maximum mean return.

On such a curriculum, we can define, for a task c:

- the *live mean return* $\bar{r}_c(t)$ by $\bar{r}_c(1) := m_c^1$ and $\bar{r}_c(t) = (r_{t_1} + ... + r_{t_K})/K$ where $t_1, ..., t_K$ are the K last time-steps when the agent was trained on a sample of c and where r_{t_i} is the return got by the agent at these time-steps;
- the *live minimum mean return* by $m_c(1) := m_c^1$ and $m_c(t) := \min(\bar{r}_c(t), m_c(t-1))$;
- the *live maximum mean return* by $M_c(1) := M_c^1$ and $M_c(t) := \max(\bar{r}_c(t), M_c(t-1))$.

From this, we can define, for a task c, the *mastering rate* $\mathcal{M}_c(t) := \frac{\bar{r}_c(t) - m_c(t)}{M_c(t) - m_c(t)}$. Intuitively, a task c would be said "learnt" if $\mathcal{M}_c(t)$ is near 1 and "not learnt" if $\mathcal{M}_c(t)$ is near 0.

Learnable tasks. An *ordered curriculum* $\mathcal{O}^{\mathcal{C}}$ is an acyclic oriented graph over tasks $\mathcal{C}$. Intuitively, an edge goes from task A to task B if A must be learnt before B.

A *min-max ordered curriculum* is an ordered curriculum along with a minimum and maximum mean return estimation for each task.

On such a curriculum, we can define, for a task c, the *learnability rate*:

$$\mathcal{M}_{\text{Anc } c}(t) := \min_{c' | c' \rightsquigarrow c} \mathcal{M}_{c'}(t)$$

where $c' \rightsquigarrow c$ means that c' is an ancestor of c in the graph $\mathcal{O}^{\mathcal{C}}$. If c has no ancestor, $\mathcal{M}_{\text{Anc } c}(t) := 1$. Intuitively, a task c would be said "learnable" if $\mathcal{L}_c(t)$ is near 1 and "not learnable" if it is near 0.

5.2 A MASTERING RATE BASED ALGORITHM (MR ALGORITHM)

Let's introduce our mastering rate based algorithm (MR algorithm) that assumes that the good next tasks to train on are the ones that are learnable but not learnt yet.

Unlike Teacher-Student algorithms that require a curriculum as input, this algorithm requires a min-max ordered curriculum: this is more data to provide but this is not much harder to do. For all the experiments in Matiisen et al. (2017); Graves et al. (2017); Fournier et al. (2018), min-max ordered curriculums could have been provided easily.

If $a_c(t)$ represents the attention given to task c, then the MR algorithm is based on the following formula:

$$a_c(t) = [(\mathcal{M}_{\text{Anc } c}(t))^p] \left[\delta(1 - \mathcal{M}_c(t)) + (1-\delta)\hat{\beta}_c^{Linreg}(t)\right] [1 - \mathcal{M}_{\text{Succ } c}(t)] \tag{1}$$

where:

- $\hat{\beta}_c^{Linreg}(t) := \frac{\beta_c^{Linreg}(t)}{\max_{c'} \beta_{c'}^{Linreg}(t)}$ if possible. Otherwise, $\hat{\beta}_c^{Linreg}(t) := 0$;
- $\mathcal{M}_{\text{Succ } c}(t) := \min_{c' | c \to c'} \mathcal{M}_{c'}(t)$. If c has no successor, $\mathcal{M}_{\text{Succ } c}(t) := 0$.

The attention $a_c(t)$ given to a task c is the product of three terms (written inside square brackets):

- The first term only gives attention to tasks that are learnable. The power p controls how much a task should be learnable in order to get attention.
- The second term, when $\delta = 0$, only gives attention to tasks that are not learnt yet. When $\delta \neq 0$, it gives also attention to tasks that are progressing or regressing the more. This is useful because if the maximum mean return estimation is 0.5 whereas as the true maximum mean return is 0.9, then agents will stop training on tasks when they reach 0.5 mean return whereas they could have reached 0.9 by continuing...
- The last term only gives attention to tasks whose successors are not learnt yet. Without it, tasks learnt and whose successors are learnt might still have attention because of $\hat{\beta}_c^{Linreg}(t)$ in the second term.

Now that the attention function is defined, all we have to do is to choose a distribution converter. Because we don't want to sample learnt or not learnable tasks, we can't use the greedy proportional or the greedy argmax ones. We rather prefer the proportional one. However, with this distribution converter, nothing ensures that easier tasks will be sampled regularly.

To overcome this, each task first gives a part γ_{pred} of its attention to its predecessors:

$$a'_c(t) := (1 - \gamma_{pred}) a_c(t) + \sum_{c' | c \to c'} \frac{\gamma_{pred}}{n_{c'}} a'_{c'}(t) \tag{2}$$

where n_c is the number of predecessors. Then, each task gives a part γ_{succ} of its attention to its successors:

$$a_c''(t) := (1 - \gamma_{succ}) a'_c(t) + \sum_{c' | c' \to c} \frac{\gamma_{succ}}{n_{c'}} a'_{c'}(t) \tag{3}$$

where n_c is the number of successors.

Hence, we get the MR algorithm:

Algorithm 4: MR algorithm

```
input: A min-max ordered curriculum $\mathcal{C}$ ;
       An agent $\mathbf{A}$;
$\beta^{Linreg} := 0$ ;
for $t \leftarrow 1$ to $T$ do
    Compute attention $a$ using equation 1 ;
    Compute redistributed attention $a'$ following equation 2 and 3;
    $d := \Delta^{Prop}(a')$ ;
    Draw a task $c$ from $d$ and then a sample $x$ from $c$ ;
    Train $\mathbf{A}$ on $x$ and observe return $r_t$ ;
    $\beta_c^{Linreg}$ := slope of lin. reg. of $(t_1, r_{t_1}), ..., (t_K, r_{t_K})$ ;
```

Finally, we can remark that the MR algorithm is just a more general version of Teacher-Student algorithms and the baseline algorithm. If we consider min-max ordered curriculums without edges (i.e. just curriculums), if $\delta = 0$, and if we use the GProp dist converter instead of the Prop one, then the MR algorithm is exactly the GProp Linreg algorithm.

5.3 Results

The MR algorithm with $\delta = 0.6$ (see appendix B for the min-max ordered curriculums given to this algorithm) outperforms the baseline algorithm on all the curriculums, especially on:

- KeyCorridor where the median return of the baseline is near 0 after 10M time-steps on S4R3, S5R3 and S6R3 while the first quartile of the MR algorithm is higher than 0.8 after 6M time-steps (see figure 6).
- ObstructedMaze where the last quartile of the baseline is near 0 after 10M time-steps on all the tasks while the last quartile of the MR algorithm is higher than 0.7 after 5M time-steps on 1Dl, 1Dlh, 1Dlhb, 2Dl, 2Dlhb (see figure 7)..

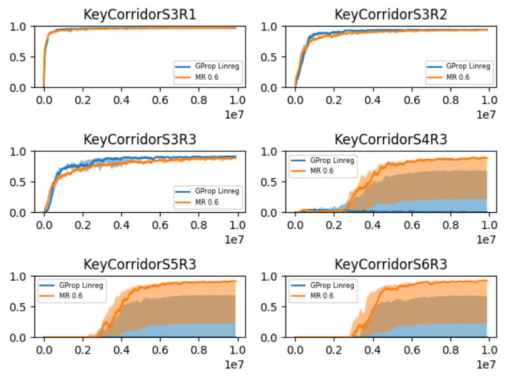


Figure 6: GProp Linreg and MR with $\delta = 0.6$ were each tested with 10 different agents (seeds) on the KeyCorridor curriculum. The median return during training, between the first and last quartile, are plotted.

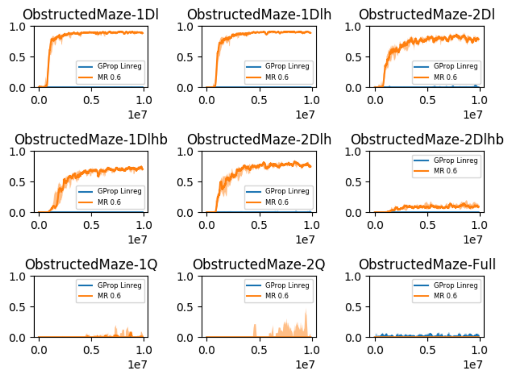


Figure 7: GProp Linreg and MR with $\delta = 0.6$ were each tested with 10 different agents (seeds) on the ObstructedMaze curriculum. The median return during training, between the first and last quartile, are plotted.

Acknowledgments

We acknowledge Jacob Leygonie, Claire Lasserre and Salem Lahlou for fruitful conversations regarding these ideas, and Compute Canada for computing support.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL `http://doi.acm.org/10.1145/1553374.1553380`.

Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer. Accuracy-based Curriculum Learning in Deep Reinforcement Learning. *ArXiv e-prints*, June 2018.

Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. *CoRR*, abs/1704.03003, 2017. URL `http://arxiv.org/abs/1704.03003`.

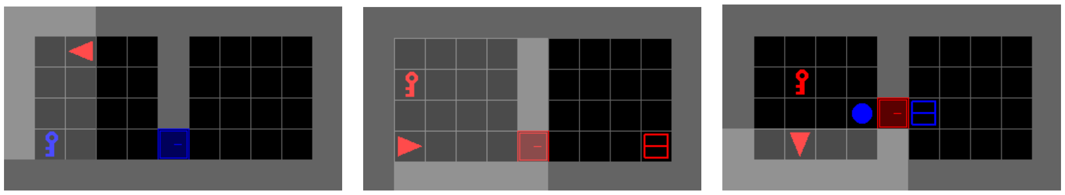
Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *CoRR*, abs/1707.00183, 2017. URL `http://arxiv.org/abs/1707.00183`.

Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. 2017.

Wojciech Zaremba and Ilya Sutskever. Learning to execute. *CoRR*, abs/1410.4615, 2014. URL `http://arxiv.org/abs/1410.4615`.

A CURRICULUMS

Three curriculums were used to evaluate the algorithms: BlockedUnlockPickup (3 tasks), KeyCorridor (6 tasks) and ObstructedMaze (9 tasks).

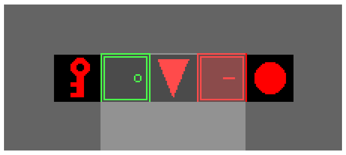


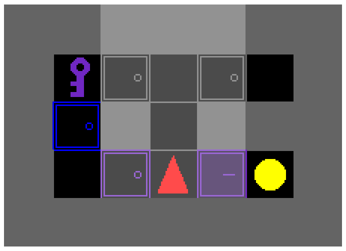
(a) *Unlock*. $n_{max} = 288$

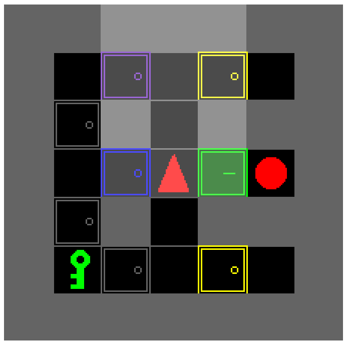
(b) *UnlockPickup*. $n_{max} = 288$

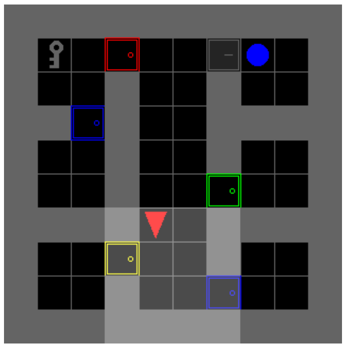
(c) *BlockedUnlockPickup*. $n_{max} = 576$

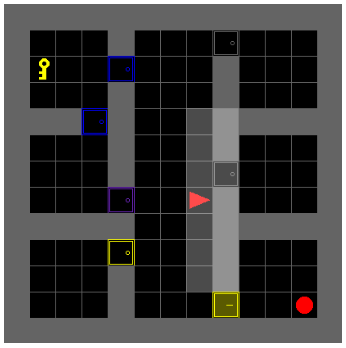
Figure 8: *BlockedUnlockPickup* curriculum. In Unlock, the agent has to open the locked door. In the others, it has to pick up the box. In UnlockPickup, the door is locked and, in BlockedUnlockPickup, it is locked and blocked by a ball. The position and color of the door and the box are random.


(a) *S3R1*.
$n_{max} = 270$


(b) *S3R2*.
$n_{max} = 270$


(c) *S3R3*.
$n_{max} = 270$


(d) *S4R3*.
$n_{max} = 480$


(e) *S5R3*.
$n_{max} = 750$

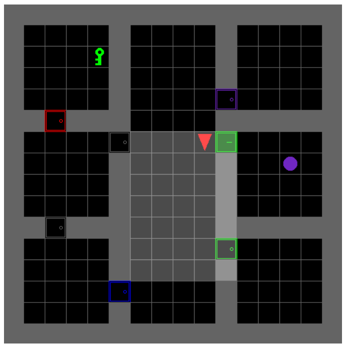
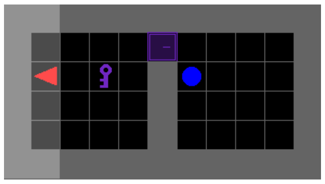
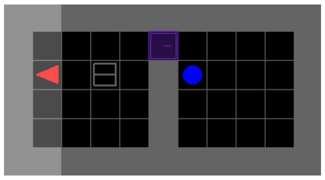
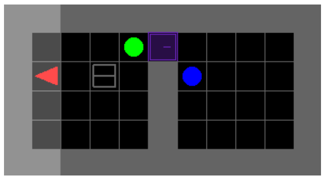
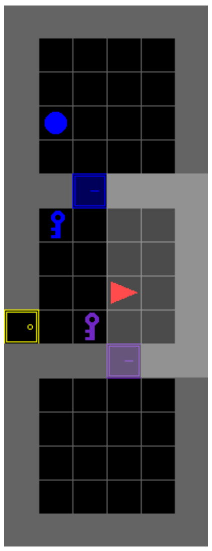

(f) *S6R3*.
$n_{max} = 1080$

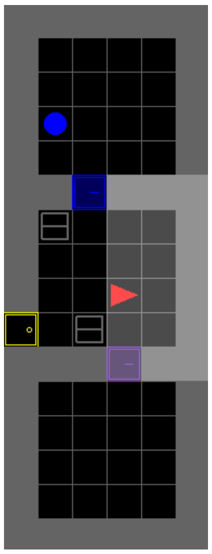
Figure 9: *KeyCorridor* curriculum. The agent has to pick up the ball. However, the ball is in a locked room and the key in another room. The number of rooms and their size gradually increase. The position and color of the key and the doors are random.

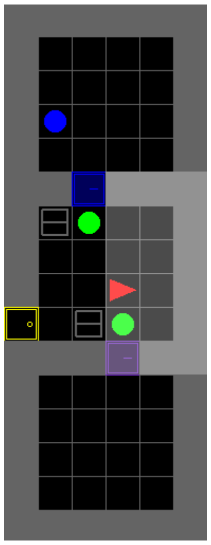

(a) *1Dl*.
$n_{max} = 288$

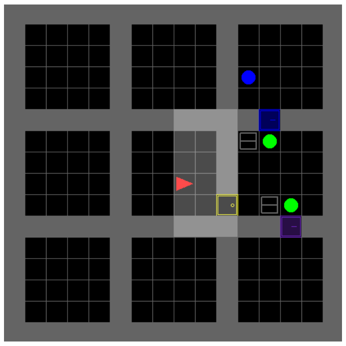

(b) *1Dlh*.
$n_{max} = 288$

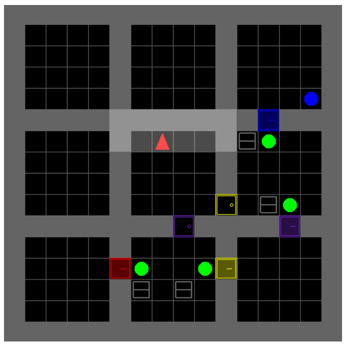

(c) *1Dlhb*.
$n_{max} = 288$


(d) *2Dl*.
$n_{max} = 576$


(e) *2Dlh*.
$n_{max} = 576$


(f) *2Dlhb*.
$n_{max} = 576$


(g) *1Q*.
$n_{max} = 720$


(h) *2Q*.
$n_{max} = 1584$

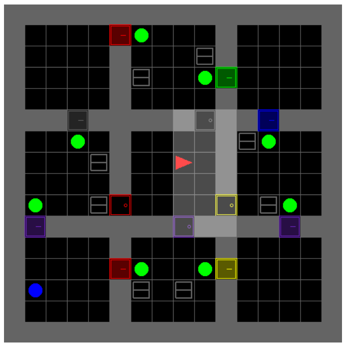

(i) *Full*.
$n_{max} = 3600$

Figure 10: *ObstructedMaze* curriculum. The agent has to pick up the blue ball. The boxes hide a key. A key can only open a door of its color. The number of rooms and the difficulty opening doors increase.

B MIN-MAX ORDERED CURRICULUMS

Here are the min-max ordered curriculums given to the MR algorithm in subsection 5.3.

For every task c, we set m_c^1 to 0 and M_c^1 to 0.5. The real maximum mean return is around 0.9 but we preferred to take a much lower maximum estimation to show we don't need an accurate one to get the algorithm working.

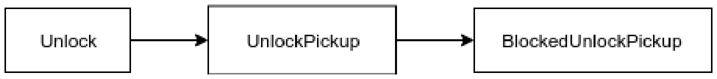


Figure 11: Oriented curriculum for BlockedUnlockPickup.

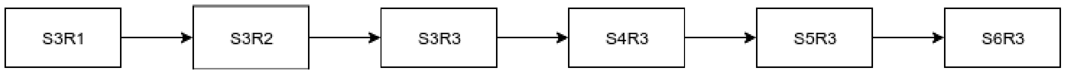


Figure 12: Oriented curriculum for KeyCorridor.

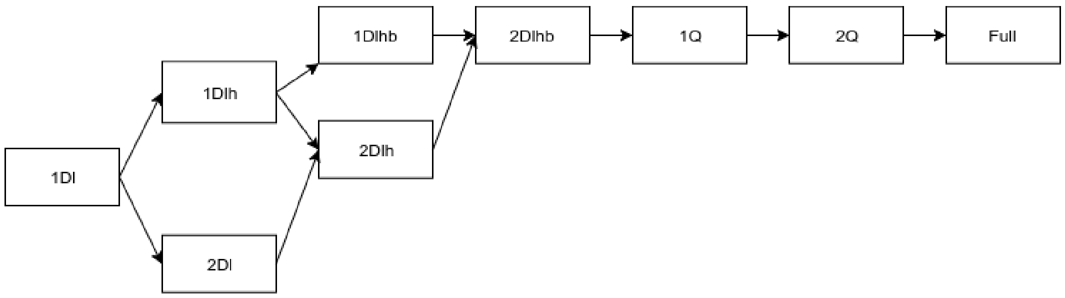


Figure 13: Oriented curriculum for ObstructedMaze.

C Hyperparameters

α	0.1
ε	0.1
K	10

Table 1: Hyperparameters used for Teacher-Student and baseline algorithms. They are the same than those used in the Teacher-Student paper.

δ	0.6
γ_{pred}	0.2
γ_{succ}	0.05
K	10
p	6

Table 2: Hyperparameters used for the MR algorithm.

D PERFORMANCE OF THE MR ALGORITHM ON BLOCKEDUNLOCKPICKUP

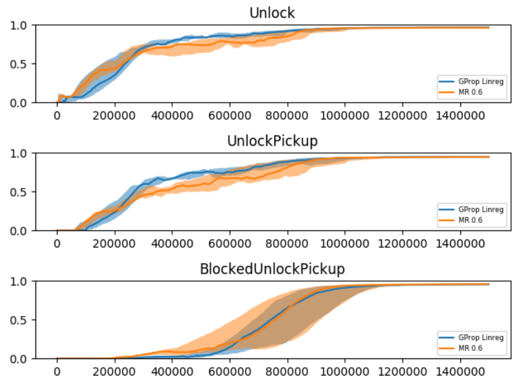


Figure 14: GProp Linreg and MR with $\delta = 0.6$ were each tested with 10 different agents (seeds) on the BlockedUnlockPickup curriculum. The median return during training, between the first and last quartile, are plotted.