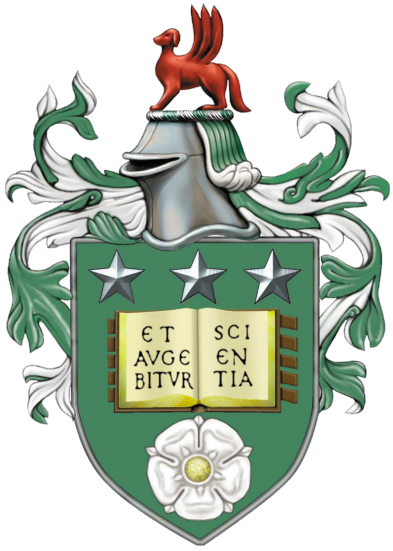
Curriculum Learning for Online Reinforcement Learning

Francesco Foglino

Submitted in accordance with the requirements for the degree of
Doctor of Philosophy



The University of Leeds
School of Computing
December 2020

To my family, who could show me a path beyond what life offered them

Declaration

The candidate confirms that the work submitted is his/her own, except where work which has formed part of a jointly authored publication has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Some parts of the work presented in this thesis have been published in the following articles. The publications are primarily the work of the candidate.

Foglino, F., Coletto Christakou, C., Luna Gutierrez, R., and Leonetti, M. (2019b). Curriculum learning for cumulative return maximization. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI

Foglino, F., Christakou, C. C., and Leonetti, M. (2019a). An optimization framework for task sequencing in curriculum learning. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 207–214. IEEE

Acknowledgements

First I want to express my gratitude to my supervisors Doctor Matteo Leonetti and Professor Anthony Cohn for their priceless guidance throughout these years.

I would like to thank all the colleagues of the Robotics Lab in the School of Computing, who shared with me this experience and supported me during the countless challenges I encountered in my research path. A special thought goes to Wissam who proved to be an incredible workmate and an even better friend.

I would also like to thank all of my friends all over the world. Jack and Aleks who lived Leeds at the forefront with me and whom I could always count on, the many ones who shared different chapters of these years in the United Kingdom with me and all the old ones who showed once again to be brave enough to strive for our friendship.

Lastly I want to thank my family for their precious support which was always close to me despite the miles in between us and most importantly my sister Caterina whose strength and love inspire me every day.

Abstract

Curriculum learning in reinforcement learning is a rapidly growing research field used to shape exploration by presenting the agent with increasingly complex tasks. The idea of curriculum learning has been largely applied in both animal training and pedagogy. In reinforcement learning, most of the previous task sequencing methods have shaped exploration with the objective of reducing the time to reach a given performance level.

In this work, we start by proposing novel uses of curriculum learning, which arise from choosing different objective functions. We define a general optimization framework for task sequencing based on combinatorial optimization. The framework is composed of several performance metrics for the evaluation of a curriculum and three different task scenarios. Furthermore we study the shape of the curricula search space in order to understand what are the salient features characterizing it.

We adapt popular metaheuristic search methods to the task sequencing problem in curriculum learning to find curricula optimizing any of the given performance metrics. Critical tasks, in which suboptimal exploratory actions must be minimized, can benefit from curriculum learning, and its ability to shape exploration through transfer. We propose a task sequencing algorithm maximizing the cumulative return, that is, the return obtained by the agent across all the learning episodes. By maximizing the cumulative return, the agent not only aims at achieving high rewards as fast as possible, but also at doing so while limiting suboptimal actions.

Finally we evaluate the performance of the metaheuristic search methods on several tasks. We show that curriculum learning can be successfully used to: improve the initial performance, take fewer suboptimal actions during exploration, and discover better policies. We also experimentally compare them to our task sequencing algorithm, and show that it achieves significantly better performance on the problem of cumulative return maximization. Furthermore, we validate our algorithm on a critical task, optimizing a home controller for a micro energy grid.

Table of contents

List of figures

List of tables

Nomenclature

Greek Symbols

α	Learning rate
ε	Exploration parameter
γ	Discount factor of an MDP
ϕ	Features of a function approximator
π	Policy
θ	Parameters of a function approximator

Other Symbols

$\mathscr{C}$	Set of curricula
$\mathscr{F}$	Set of final tasks
$\mathscr{M}$	A set of MDPs
$\mathscr{P}$	Performance function
$\mathscr{T}$	Set of source tasks
A	Set of actions in an MDP
c	A curriculum
C_0	Empty curriculum
G_0	Expected discounted return
L	Maximum curriculum length
m	A reinforcement learning task

n Source set dimension

p Transition function of an MDP

q Q-function

r Reward function of an MDP

S Set of states in an MDP

Acronyms / Abbreviations

ACO Ant Colony Optimization

ANN Artificial Neural Network

CL Curriculum Learning

CO Combinatorial Optimization

CR Cumulative return performance

DRL Deep Reinforcement Learning

GA Genetic Algorithm

GAN Generative Adversarial Network

JS Jumpstart performance

MDP Markov Decision Process

MR Max-return performance

PPO Proximal Policy Optimization

RL Reinforcement Learning

TL Transfer Learning

TS Tabu Search

TTT Time-to-threshold performance

Chapter 1

Introduction

> "As the result of careful scheduling, pigeons, rats, and monkeys have done things during the past five years which members of their species have never done before. It is not that their forebears were incapable of such behavior; nature had simply never arranged effective sequences of schedules"
>
> *- B. F. Skinner 1959*

1.1 Motivation

Recent advances in reinforcement learning allowed us to develop algorithms able to solve problems that, just a few years ago, were unimaginable to approach. The reason behind the great success of these state-of-the-art techniques can be largely attributed to the advent of Deep Reinforcement Learning (DRL) methods [50]. Although it is not the first time that a reinforcement learning agent surpasses human performance on artificial domains [89], DRL agents are the first to directly learn from high-dimensional sensory inputs using end-to-end reinforcement learning. This achievement was rapidly overtaken by even more outstanding results in incredibly complex domains such as in the game of Go where AlphaGo, the artificial agent developed by Google DeepMind, won against the grandmaster player Lee Sedol [74]. The number of domains where DRL agents can outperform professional human players keeps growing, challenging always more complex environments as in the case of popular online strategy games such as Starcraft [92] and Dota 2 [9] or even real-world robotics training [97].

The results achieved in these domains show the benefits of deep reinforcement learning algorithms although they still present a big weakness in terms of sample efficiency. Because of this reason, all these solutions require an enormous number of training iterations before being able to perform well over the designated task, and therefore only new powerful generations of hardware and parallel computing could really allow DRL to overtake human

performances in these tasks. For instance, OpenAI developed a Dota 2 AI named OpenAI Five which accomplished expert-level performance by playing over 10,000 years worth of games. One of the reasons justifying this fact is that most of these agents are trained without leveraging any prior knowledge, hence, differently from any biological learner, they learn how to solve any task from scratch.

As we compare the final performance of these astonishing artificial agents to those of human experts in the task domain, we believe it still remains of crucial importance to also notice the differences between the two learners at a training level in order to better understand how to develop the next generation of artificial agents.

In this Ph.D. thesis we identify the previously mentioned knowledge transferability issue as an opportunity to improve the performance of state-of-the-art reinforcement learning algorithms. We argue how a promising solution taking advantage of these techniques while improving generalization capabilities is represented by curriculum learning. This field is inspired by the way biological learners approach problems during the course of their lives and develop methods solving particularly complex tasks by learning through a set of increasingly more difficult challenges, exploiting previously accumulated experience.

1.2 Curriculum Learning in Reinforcement Learning

Let us consider a reinforcement learning problem with a very large number of states. Even in the case of simple domains, any RL agent would take a long time before converging to an optimal solution if no prior knowledge about the problem is available to the learner. Nevertheless, in many cases it can be easy to identify the most interesting areas of the state space if the agent already performed training over different problems in the same domain. Following this principle, given an RL task, it is possible to specifically design new problems to train the agent on in such a way that it learns increasingly more difficult domain features useful to solve the final reinforcement leaning task.

In machine learning, curriculum learning is the field that studies algorithms for sorting the experience over which the artificial agent is trained, in order to improve its learning performance. These types of techniques were first studied in the context of supervised learning where the focus was mainly on ordering the training set [8]. Curriculum learning was adapted to the reinforcement learning setting only more recently and, as we will later analyze in Chapter 3, it rapidly branched out into a number of approaches substantially different from each other.

This work studies algorithms sequencing different reinforcement learning tasks for transferring the knowledge coming from each of them through the sequence (curriculum) in order to provide a robust initialization for agents approaching a particularly complex final

task. The methods researched in this thesis strongly rely on transfer learning algorithms efficiently passing experience from one task to another. We are particularly interested in studying the category of reinforcement learning tasks where curriculum learning applications can have a great impact. This is the case of problems where it is possible to spend a long time offline searching for an optimal curriculum in order to minimize the number of expensive actions taken by the agent during online learning.

1.3 Curriculum Learning in Humans

Research in the field of machine learning aims at developing artificial agents able to learn to solve problems in a given environment with performance equal or even superior to those observable in nature. Because of this reason, the relationship between this area and others such as neuroscience, psychology and biology is historically very strong. The first steps towards a theory of reinforcement learning date back to the *puzzle box* experiments conducted by the american psychologist Edward Thorndike, where cats successfully learned how to escape a cage *learning by trial and error* [90]. Animal experiments continuously played a crucial role in the development of what we nowadays consider the foundation of reinforcement learning. For instance, the term *conditioning* was first adopted to describe animal reactions to natural stimuli and how they could be trained to respond with a similar behaviour to different artificial stimuli [59], and behaviourism dates back to Skinner's animal behaviour experiments, where the psychologist engineered the reward signal provided to animals so that they could learn extremely complex sequences of actions [75].

Curriculum learning is also directly inspired by the previously mentioned fields with particular focus on human training. A great example is represented by the training strategies used by teachers in school. It has never been the case, indeed, that students begin learning a new subject starting from its most complicated and particular features, but rather the teacher carefully designs classes in order to initially show only the easiest concepts increasingly teaching more complicated ones over the course of the years. Education is not the only area where we can find examples of curriculum learning as this technique is used to train over the most of the tasks we learn how to solve in our lives, from speaking, writing or reading to playing a sport, driving a car or cooking. The strategy in these cases is always to divide a complex and important task into subproblems that are more practical to learn for the student but also easier to explain for the teacher. Often times we can also observe how different skills are required in order to solve a task and therefore each of them is trained separately so to improve the performance on the whole task.

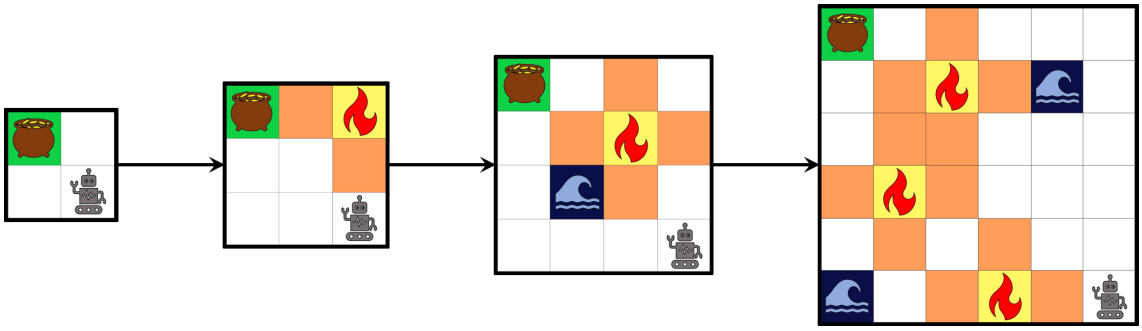


Fig. 1.1 An example of curriculum in the artificial domain Gridworld

As we will discuss in more detail in Chapter 3, these are some of the major training strategies that curriculum learning exploits in order to develop more efficient learning strategies for artificial agents.

1.4 Challenges and Contributions

In spite of the inspiration that we can draw from nature in order to shape curriculum learning algorithms, artificial learners differ from biological ones in many different aspects. This dissimilarity, in our opinion, represents an important challenge for this field. For example, in Figure 1.1 we show a possible curriculum for learning a complex task (the rightmost in figure) in a Gridworld domain. This curriculum challenges the agent with iteratively more complex tasks, introducing for each new intermediate problem a different element present in the final task that we desire the agent to learn. Intuitively, this is a good learning strategy but, without any further specification of the actual reinforcement learning algorithm used to learn each single task, or details regarding the transfer learning technique for passing knowledge from one task to the next one, it is actually impossible to determine whether or not the curriculum will be beneficial to our agent. We humans are naturally able to acknowledge the importance and potential impact of a curriculum for a learner but we also tend to apply human intuition into the creation of the curriculum itself which, as we will later see, in some cases can be desirable but could also bring unexpected consequences.

In this work we investigate algorithms able to find optimal curricula for a given final task searching among the many sequences of intermediate tasks that can be generated starting from a finite set of source tasks. More specifically:

- we propose a framework for curriculum learning in reinforcement learning with several performance metrics and task scenarios

- we collect a dataset of curriculum learning experiments and perform a preliminary analysis over the salient features characterizing the task sequencing problem
- we adapt popular metaheuristic algorithms for combinatorial optimization to search optimal curricula for any given performance metric
- we propose our heuristic search method to perform task sequencing for cumulative return maximization
- we evaluate all the proposed algorithms in multiple experiments over different artificial domains
- we discuss a methodology for applying our heuristic search to real-world problems and experimentally evaluate its effectiveness

1.5 Thesis Overview

In this chapter we introduced the motivations behind this Ph.D. thesis and briefly discussed the strong connections between curriculum learning and early behavioural studies for animal training. Furthermore, we also elaborated on the challenges of the field and how our study attempts to consider and exploit them to attain a better comprehension of the problem we are trying to solve.

In Chapter 2 we briefly introduce basic terminology and concepts regarding reinforcement learning, transfer learning and combinatorial optimization in order to allow for a better comprehension of our study.

In Chapter 3 we discuss all the relevant literature in the field of curriculum learning with particular focus on those works that are the closest to our research. We elaborate on the exploration-exploitation problem in reinforcement learning and show how curriculum learning is a proposition to solve this issue and we also investigate the origins of this area of research from its early applications in cognitive science.

In Chapter 4 we introduce a curriculum learning framework where we identify three different task scenarios and four performance metrics as possible optimization objectives. We formulate the task sequencing problem in curriculum learning as a combinatorial optimization search and analyze the main features characterizing it.

In Chapter 5 we discuss different algorithms for performing a search in the space of all the possible curricula generated by a set of intermediate source tasks. We also introduce our own heuristic method able to sequence tasks at the purpose of maximizing the cumulative return performance.

In Chapter 6 we perform an experimental evaluation of all the proposed algorithms and show the performance of our heuristic method on a real-world application to demonstrate the practical implication of curriculum learning.

We finally conclude with Chapter 7 where we illustrate the contributions brought by this thesis but also the limitations and possible future expansions for this research work.

Chapter 2

Background

In this Chapter we provide a general introduction of the main subject areas and techniques discussed and utilized in this work. We present basic terminology and concepts of reinforcement learning, transfer learning and combinatorial optimization in order to offer the reader the appropriate background knowledge to understand the research in this thesis.

2.1 Reinforcement Learning

We model tasks as episodic Markov Decision Processes. An MDP is a tuple $\langle S, A, p, r, \gamma \rangle$, where S is the set of states, A is the set of actions, $p : S \times A \times S \to [0,1]$ is the transition function, $r : S \times A \to \mathbb{R}$ is the reward function and $\gamma \in [0,1]$ is the discount factor. If a state is represented as a vector $s = \langle v_1, \ldots, v_d \rangle$ of d variables the representation of the state space is said to be *factored*. Episodic tasks have *absorbing* states, that are states that can never be left, and from which the agent only receives a reward of 0.

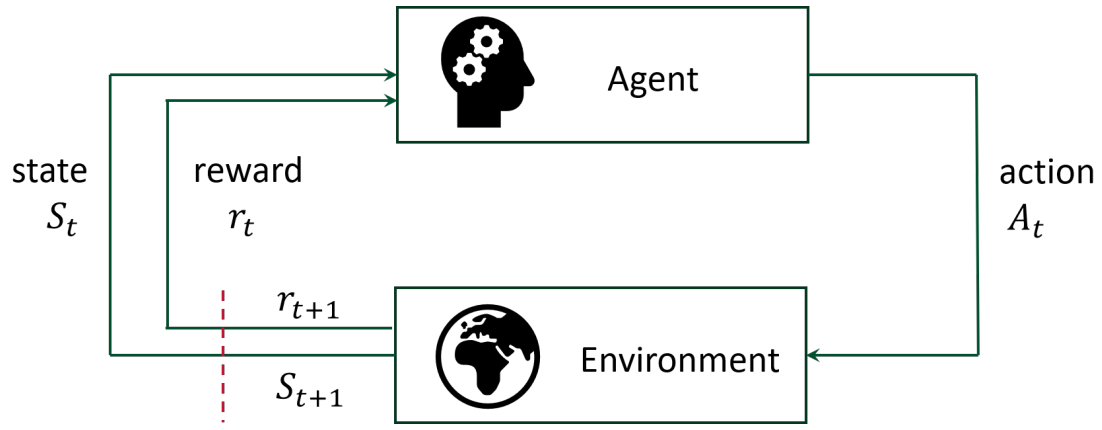


Fig. 2.1 The classic reinforcement learning diagram representing the interaction between agent and environment in an MDP [81]

For each time step t, the agent receives an observation of the state and takes an action according to a policy $\pi : S \times A \rightarrow [0,1]$. The aim of the agent is to find the *optimal* policy π^* that maximizes the expected discounted return $G_0 = \sum_{t=0}^{t_M} \gamma^t r(S_t, A_t)$, where t_M is the maximum length of the episode.

In this work we perform two sets of experiments with different learning algorithms and function approximators in order to test our solutions under different settings. To this purpose we select two of the most popular reinforcement learning algorithms that differ one another on many different aspects.

Sarsa(λ) is a learning algorithm that takes advantage of an estimate of the *value* function $q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a]$. We represent the value function with a linear function approximator, so that the learning algorithm computes an estimate $\hat{q}(s,a) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s,a)$ of $q_\pi(s,a)$ as a liner combination of *features* $\boldsymbol{\phi}$.

Proximal Policy Optimization (PPO) [71] is a policy gradient algorithm that optimizes the loss function defined as $L^{CLIP}(\theta) = \mathrm{E}_t[\min(p_t(\theta)\hat{A}_t, \mathrm{clip}(p_t(\theta), 1-e, 1+e)\hat{A})]$, where $p_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ and $\hat{A}$ is the advantage function. In this case, θ are the parameters of a neural network used as function approximator for this learning algorithm.

2.2 Transfer Learning

Curriculum learning leverages transfer learning to transfer knowledge through the curriculum, in order to benefit a final task. In transfer learning, an agent performs training over a task, at the end of which the resultant knowledge is moved to another task in order to improve the learning capabilities of the agent. The transfer takes place between pairs of tasks, referred to as the *source* and the *target* of the transfer.

In this work we use two different TL methodologies, one per type of function approximator. For Tile Coding we use a transfer learning method based on value function transfer [88], which uses the learned source q-values, representing the knowledge acquired in the source task, to initialize the value function of the target task. A detailed explanation of this method can be found in Chapter 4. In the case of artificial neural networks we used a similar transfer learning technique specifically designed for this type of architectures: *progressive networks* [66]. This method instantiates a new neural network for each new task the agent encounters during learning and links its layers to those coming from the previously trained networks by building *lateral connections*. These two techniques share the idea of transferring the whole source function approximator to the target task. This fact is desirable in curriculum learning as any part of the knowledge gathered through the interaction with a source task can be useful for learning any of the following tasks in the curriculum.

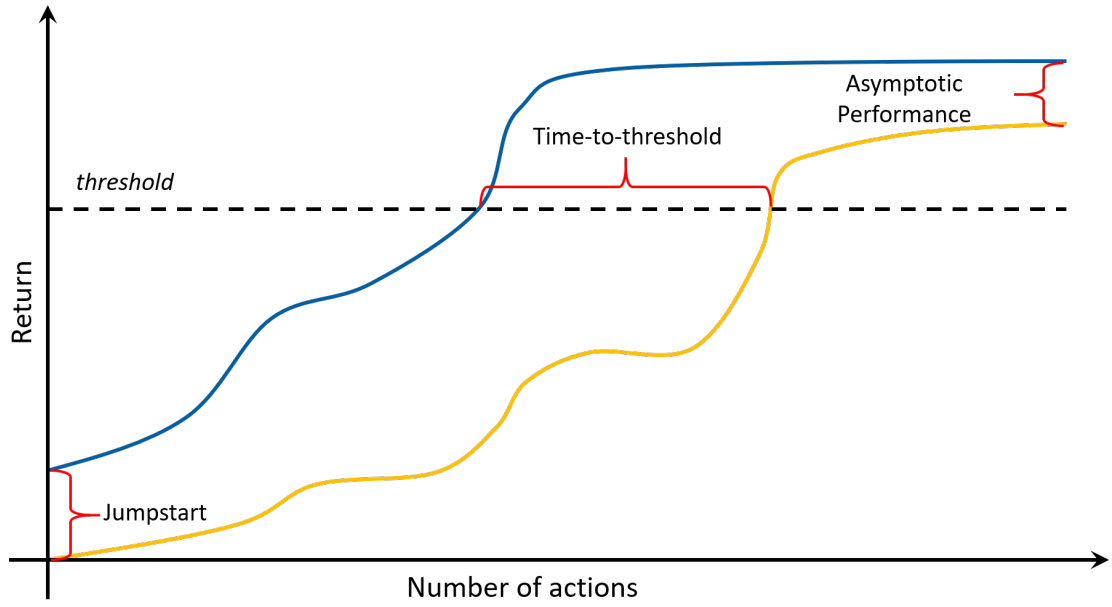


Fig. 2.2 Graphical representation of learning improvements on transfer learning metrics [88]

Several metrics have been designed to evaluate the performance improvement introduced by the adoption of a transfer learning algorithm for training over a target task [88]:

- *Jumpstart* measures the performance of the agent during the initial learning steps in the target task;
- *Time-to-threshold* is the time needed by the agent for reaching a pre-specified performance value;
- *Asymptotic Performance* corresponds to the value of the policy learned by the agent at the end of the training;
- *Total Reward* is the total amount of reward accumulated by the agent during the training in the target task (in other words, the area under the learning curve)
- *Transfer Ratio* measures the ration between the total reward of the agent learning the target task with and without transfer

We study and discuss the first four explaining how to adapt them to curriculum learning. In Figure 2.2 we give a graphical representation of the improvement that positive transfer can bring to an agent learning from scratch. Three of the five previously listed metrics are directly indicated in the figure but it is also possible to appreciate the performance difference in terms of total reward and transfer ratio.

2.3 Combinatorial Optimization

Combinatorial Optimization (CO) problems are characterized by the goal of finding the optimal configuration of a set of discrete variables. The most popular approaches in this field, called *metaheuristics*, are *approximate* algorithms, that do not attempt to search the solution space completely, but give up global optimality in favor of finding a good solution more quickly. Metaheuristics are applicable to a large class of optimization problems, and are the most appropriate methods for black-box combinatorial optimization, when a particular structure of the objective function (for instance, convexity) cannot be exploited. Task sequencing is one such black-box problem and, since the time required for learning through a sequence of RL tasks is usually large, an exhaustive testing of all possible curricula is generally unfeasible. Therefore we selected four of the most popular metaheuristics algorithms for comparison with our search method: Beam Search [45, 58], Tabu Search [29], Genetic Algorithms [31], and Ant Colony Optimization [17].

Beam Search and Tabu Search are *trajectory-based* algorithms, which starting from a single instance, search through the neighborhood of the current solution for an improvement. More specifically, Beam Search is a greedy algorithm that, at each iteration, generates new candidate solutions by expanding the previous w best ones. In order to be implemented, this metaheuristic only needs the definition of the expansion action and the value of the *beam width* w. Tabu Search, instead, performs a local search of the currently best solution in order to iteratively move to a new one with improved performance. Because of this reason, for this algorithm it is necessary to define the neighborhood of a solution so to determine the next set of candidate solutions to evaluate. Furthermore Tabu Search requires the definition of a fitness function $\mathscr{F}_t$ for performing the evaluation step, and the parameter T representing the size of the tabu list where previously visited solutions are stored.

Genetic Algorithms and Ant Colony Optimization are *population-based* algorithms, that start from a set of candidate solutions, and improve them iteratively towards successive areas of interest. A Genetic Algorithm is an evolutionary algorithm that begins from an initial population of candidate solutions and repeatedly evolves them towards better ones. This is possible by implementing nature-inspired processes such as the crossover and mutation operations. Similarly to Tabu Search, this metaheuristic needs the definition of its fitness function $\mathscr{F}_g$ and the value U of the size of the new population. Ant Colony Optimization is inspired by the behaviour of real foraging ants leaving pheromone on successful trails so that other ants can follow it. It leverages multiple agents, that at each iteration attempt to move closer to the goal of the problem by modifying their trails (solutions). The possible modifications and the relative probability of being applied on a trail need to be specified in order to implement this algorithm. Furthermore, Ant Colony

Optimization also requires the definition of the pheromone update rule so that promising solutions are visited and modified by different ants several times.

Chapter 3

Related Work

Curriculum learning (CL) in reinforcement learning (RL) is a recent field of research which aims at improving the exploratory behaviours of RL agents by leveraging knowledge coming from previous experience of the agent on a number of different tasks. Curriculum learning exploits this knowledge for improving the learner performance over one or multiple particular RL tasks called the *final* tasks. Although the optimization objectives can vary substantially for different settings, curriculum learning always acts directly on the exploration strategy employed by the learning agent, adapting it accordingly to the final task scenario. CL methods can also be seen as a natural expansion of transfer learning (TL) algorithms which mainly focus on moving knowledge from one only intermediate task to the final one.

In this chapter we begin by surveying the most relevant exploration techniques in reinforcement learning and discuss the challenge of applying them to modern RL algorithms. We then move our focus to the origins of curriculum learning as a theory in language studies experimentally applied to simple supervised learning problems. Later we perform an in-depth analysis of state-of-the-art curriculum learning research with a particular focus on task sequencing algorithms which represent the core of this Ph.D. thesis. Finally we discuss literature relative to closely related fields, the source task creation problem and transfer learning algorithms specifically designed for curriculum learning.

3.1 Exploration in Reinforcement Learning

In reinforcement learning the agent learns how to solve problems in a given environment throughout an iterative interaction within it. In classic RL problems, the learner starts without any prior knowledge about the best policy to be applied for solving a specific task. Therefore, during the beginning of the training phase, the agent, in each state, takes decisions that are strongly biased towards its initial estimate of the quality of each action.

This can be a problem of major concern as, in this way, the agent in each learning episode could select and update always the same combination of actions, thus quickly converging to a sub-optimal solution. Because of this reason, at each time step, a good learner must also take actions that look less promising, in other words it must *explore*. Finding the right balance between, visiting new actions and states not explored yet, and following again the ones already visited in past iterations for learning more about them, is famously known as the *exploration-exploitation* problem [81, 84]. The design of effective exploratory strategies is of crucial importance in reinforcement learning in order to guarantee the discovery of optimal solutions. In this section we survey the most popular such strategies and discuss their limitations.

3.1.1 Classic Exploratory Strategies

The most popular exploration policy in reinforcement learning is ε*-greedy* due to its simplicity. This technique defines a probability $\varepsilon > 0$ for which the learning agent, instead of taking the action currently estimated as best, randomly selects another one, hence explores. By definition, ε-greedy uniformly samples all the sub-optimal actions assigning them the same probability. Alternatively it is possible to compute different probability values for each sub-optimal action based on their current estimate. Such a technique is called *Boltzmann exploration*. These two strategies are widely used in many different reinforcement learning applications because of the simplicity of their implementation. Nonetheless it is possible to design more advanced exploration techniques depending on the particular RL setting.

In the context of bandit problems, where the MDP has one state only, more sophisticated exploratory policies were proposed. In some cases, it is indeed possible to calculate the upper confidence bound (UCB) for each action. An algorithm leveraging such information is *UCB1* [4] which, at each time step, selects the action with the best upper confidence bound. This technique can be further improved by including the variance of the reward associated with each action in order to improve the UCB calculation [3]. Moreover, whenever it is also possible to compute the lower confidence bound, we can use even more drastic biases for selecting the best action at each time step [20, 48, 51].

All of the previously listed works are applied to bandit problems where we only have one state, therefore the exploration problem is limited to efficiently sample all the possible actions. This Ph.D. thesis focuses on reinforcement learning agents training in MDPs with multiple states and more specific exploration policies were previously studied for solving this type of problem. Ideally we would want our agent to visit all the possible state-action combinations in the given MDP in order to fully explore it. It is possible to achieve such a result for deterministic MDPs [57] although this is intractable for very large

tasks. In case of stochastic MDPs, instead, there exist no exploratory behaviours with the same guarantees, even for small problems [84]. It is indeed impossible to define a finite amount of time to systematically visit all state-action combinations in this particular case as any exploration strategy is still subject to the stochastic nature of state transitions for this category of MDPs. As a result, the most effective exploration strategies for general MDPs, E^3 [16, 40] and its improvement R-max [10], aim at exploring the given MDP by dynamically changing the focus of exploration towards states and actions whose estimate is the most uncertain.

The exploration problem can also be translated into two different problem definitions. The first one is regret minimization, where the agent needs to reduce the number of expensive actions it takes while still training in the environment. The algorithm *UCRL2* [36], which uses upper confidence bounds to choose a new exploration policy at the beginning of each new learning episode, is a possible approach to this problem, although it cannot be applied to all the classes of MDPs such as those with transient states, i.e. states that are not accessible from some other states. Alternatively, another objective can be that of minimizing the time required to reach a pre-defined performance value. Particularly interesting for this case are PAC-MDP algorithms (Probably Approximately Correct in Markov Decision Processes), where this time is bounded with high probability as a polynomial function in the MDP parameters. Such algorithms are the previously mentioned R-Max [10, 37] and many others as well [78, 79, 85, 86]. The main issue with these approaches is that they are best applied to small MDPs and fail to scale efficiently.

The exploration techniques we discussed so far are all single-task type of exploration, as they are directly applied on the only task the agent has to learn, thus they are not designed to use any prior knowledge. Next we are going to discuss algorithms that instead are based on training the agent on other tasks before approaching the target one.

3.1.2 Transfer Learning

The majority of modern reinforcement learning problems are characterized by a very large number of state features. This generates MDPs that are intractable by tabular RL methods and the previously discussed exploratory strategies are not efficient enough to be applied in these cases. A classic approach to this issue is using function approximators and, nowadays, non-linear ones, especially Artificial Neural Networks (ANNs) are widely applied in reinforcement learning due to their effectiveness which comes at the cost of losing the guarantee of global optimality. Because of this reason the most popular exploration policies today are simple algorithms like ε-greedy, or in policy gradient methods exploration can be directly included in the definition of the learning algorithm.

An alternative approach to the exploration-exploitation problem can be found in transfer learning (TL) algorithms [43, 88]. These methods are based on the idea of exploiting previous knowledge gathered on an auxiliary task, *source task*, in order to improve the performance of the learner over the *target task*. In this way it is possible to avoid exploring less promising areas of the target task state space following the knowledge gathered over the source task. Despite the fact that the explicit objective of this field is not studying innovative exploration strategies, this is still an implicit result of the research in this area. Nonetheless it is possible to specifically design TL algorithms to systematically explore target tasks [47].

Transfer learning methods can be divided following the type of knowledge that the agent can exploit for learning the target task [88]. This can be in the form of low-level knowledge such as value functions [87], policies [23] or task model [21], or high-level knowledge like options [76] or reward shaping [56]. The former case gives a complete initialization for the initial policy of the agent, while the latter provides additional information to the learner which will be used to guide its learning process [88]. Particularly interesting is the reward shaping case where the learner is provided with additional rewards, other than those supplied by the underlying MDP, in order to guide its exploration towards the most interesting areas of the state space.

Curriculum learning expands the core idea of transfer learning into a setting where the previous knowledge is coming from a number of different source tasks instead of just one. As we will later discuss in this thesis, transfer learning is directly employed in curriculum learning for implementing the passage of knowledge for any pair of tasks and it also plays a crucial role in the definition of CL performance metrics.

3.2 Curriculum Learning

In machine learning, curriculum learning is the area that studies training strategies for iteratively introducing a number of gradually more complicated problems to an agent learning how to solve a complex task. The idea is that, by doing so, the agent is able to improve its performance over the given final task, which can result in a faster learning process or, more generally, in achieving more robust knowledge.

3.2.1 Origins of Curriculum Learning

Questioning the different importance of samples composing a training dataset and studying these concepts for machine learning agents was initially introduced in the context of cognitive science while studying learning capabilities in children approaching the problem of learning a language. Many studies, indeed, were arguing how the number of given

training samples was not the only important feature to be taken into consideration for accomplishing effective language learning, but the presence of positive and negative examples [11, 61] or their relative complexity [18] were also important topics.

The importance of starting small for efficiently learning particularly difficult tasks was firstly applied to artificial agents in language studies [18]. This theory argues that, learning through increasingly more complex examples can improve the performance of an agent training over a complex problem. In this study, a simple recurrent connectionist network was used to learn an artificial grammar. The system was tested both when learning from the whole training set without a particular ordering and when instead, incrementally more difficult training examples were introduced to it. The results showed how the selected task was virtually impossible to learn without sorting the samples.

Later works applied similar concepts in different domains such as robotics [68]. In this context *Trajectory Extension Learning* was proposed, a method involving gradual parameter modifications to allow quicker convergence for learning control in robot manipulation tasks. The underlying idea is that knowledge is bootstrapped from regions of easy solvability into regions that require more difficult dynamic behaviours.

Later robotics applications followed up on this strategy to learn control strategies for a 2-link robotic manipulator with four single-joint muscles and two double-joint ones [39]. This was possible by gradually adjusting parameters such as the speed of the arm movements and the viscoelastic property of the arm during the learning process.

Interestingly, task decomposition for tackling complex robotic domains requiring online learning was introduced in the same period [14]. Curriculum learning in reinforcement learning was used for the first time in those years, still in the robotics domain [2]. In this case the problem consisted of a robotic agent learning how to shoot a ball into a goal, based on visual inputs. The learning took place in simulation in order to be later transferred into the real world so to test the learned policy. This was not sufficient to learn useful policies in a sensible time because of the extreme reward sparsity characterizing the problem. As a solution the authors proposed *Learning from Easy Missions*, an algorithm for controlling the robot and ball position in the environment to create tasks of increasing difficulty in order to help the agent progress faster in learning the desired policy. The concept of starting small was also strongly criticized [65] although showing the importance to learn starting with simpler models (agent architectures) or to initialize the system with pre-trained weights.

In machine learning, the increasing popularity of non-linear function approximators such as neural networks for solving tasks in always more complicated environments, introduced new challenges in the field as these architectures started growing deeper and more complex. Different studies investigated techniques for using training data in the most efficient way. Starting small became a less popular idea at this purpose, but it is

interesting noticing how, new specifically designed techniques were based on the same key ideas emerging from the previously described studies such as incrementally training different parts of a neural network [33] or pre-initializing it for better learning from the given samples [19].

3.2.2 Experience Replay Applications

Despite its origins from applications in the field of cognitive science, curriculum learning gained popularity and became an important technique for optimizing the training of supervised learning agents [8] by ordering the experience samples from the easiest to the hardest before using them to train the learner. The result of such a process is extremely beneficial to the agent because, in this way, it can learn faster and improve its generalization capabilities.

It is easy to adapt this technique of sample ordering also to a reinforcement learning set-up with experience replay [44]. In this type of setting, the learner stores state-action-reward tuples in a *replay buffer* in order to reuse them multiple times, and improve so its learning performance. This is a popular choice for some of the most advanced RL algorithms such as Deep-Q-Networks [50]; nevertheless, in this case, the tuples are randomly sampled from the replay buffer.

The first technique to introduce an ordering for the samples in the replay buffer is Prioritized Experience Replay (PER) [69], where more importance is given to those tuples whose expected learning progress is high. This can be measured through the temporal difference (TD) error. In this way the most useful transition samples are used more often, and therefore the agent learns how to solve a specific task faster. Other types of ordering are also possible by taking into account richer information about the transition samples, for instance the number of times each of them is selected, or the degree of difficulty in using a particular tuple in the current learning step [63]. The main issue with these approaches is that they need to be tuned and designed specifically for each environment, and therefore they do not generalize well over different domains. Furthermore, in both these methods, the importance of the training samples is calculated off-line, after the replay buffer contains enough of them. However, this relevance value can also be calculated with an artificial neural network specifically designed to solve this prediction problem [42].

The previous methods might result ineffective for problems characterized by sparse reward. In this setting a reinforcement learning agent can take an extended period of time before collecting tuples containing reward, especially during the initial part of the learning process. Hindsight Experience Replay (HER) [1], instead of sorting the training sample following a pre-defined metric, focuses on restructuring the tuples in order to be able to exploit as much as possible the experience coming from unsuccessful episodes.

This can be done by replaying the experience fixing as a goal the state that was actually achieved during the episode rather than the one that the agent was aiming to achieve. This trajectory is then used by an off-policy RL algorithm allowing the agent to exploit an implicit curriculum arising from the fact that, in early episodes an agent is likely to reach easy states more often than difficult ones. In spite of this, better curricula can be created over such re-elaborated trajectories by giving replay priority to goals that have not been used yet (curiosity) and goals that are similar to the actual goal the agent aims at achieving (proximity) [22]. These two criteria are combined so as to order the training samples in an adaptive way to the progress of the learner training over the given task.

The clear advantage of curriculum learning techniques for experience replay is the exploitation of the most useful transitions encountered during the learning phase of the agent. Thanks to off-policy RL algorithms, this process can be repeated multiple times during the training, sensibly improving the learner performance. At the same time, these methods alone introduce a hard constraint to the general formulation of the curriculum learning problem. The transition samples, indeed, must be directly collected over the target task to be optimized. We will later analyze techniques that relax this restriction in various ways in order to generate curricula of entirely different tasks. This is the type of curriculum studied in this Ph.D. thesis. Nevertheless, all curriculum learning algorithms in any setting can still be traced back to ordering experience samples, being these from the same task or many different ones. Thus, CL in experience replay represents an important way to deduce broad research directions for curriculum learning in any type of setting.

3.2.3 Changes on Initial and Terminal state distribution

In order to create strong curricula, a curriculum learning algorithm needs to be able to directly influence the experience that the learner accumulates while training over a given task. Although the methods in this section are still limited to knowledge gathered during episodes on the target task, they represent an expansion to curriculum learning techniques for experience replay. These algorithms, indeed, induce a partial ordering on the states to be visited during each episode progressively sampling the state space.

The first work adopting this strategy, *learning from easy missions*, was applied to a robotic agent learning to shoot a ball into a goal [2]. This is possible by progressively moving the agent away from the goal inducing in this way a curriculum. The learning agent receives a visual input which is both used for learning how to solve the given problem and defining the heuristic method determining the next task the agent needs to train on. Each of these tasks is created manually and the adopted heuristic is domain specific and based on the assumption that it is possible to identify a dimension over which the difficulty of the task varies.

A more general method expanding on this idea proposes to generate, at each algorithm iteration, a distribution of possible starting states [26]. During the first training iterations, these are selected by taking a random walk around the goal state only, while later also previous starting states are included. In this way the agent effectively manages to learn over a curriculum of progressively more complex tasks that can easily exploit previously gathered knowledge, as part of the state space is always shared among these tasks. The starting states are selected for an expansion depending on the value of their expected return. In this way it is possible to create tasks that are neither too difficult nor too simple for the agent to progress on.

A similar method performs expansion steps not from the goal state but from the given starting one [25]. This allows for the discovery and exploration of new goal states at each episode. In this context, the curriculum generation is managed by an Generative Adversarial Network (GAN) [32]. The GAN *generator* proposes a new goal region (task) to the learning agent, while the GAN *discriminator* evaluates if the given goal region represents a good task for the agent to train next. Even in this case, this can be determined by checking the return value of a starting state and as a result, at each iteration the algorithm generates goal states always farther from the starting state.

An alternative solution for iterative goal generation is SAGG-RIAC (Self-Adaptive Goal Generation — Robust Intelligent Adaptive Curiosity) [6]. This algorithm automatically generates and samples new goal states defining in this way tasks of increasing difficulty for learning inverse models in high-dimensional redundant robots. The method is based on the definition of *competence*, a measure of the distance between the final state reached by the agent and the actual goal state for the given task, and *interest*, the change of competence for a set of goals during the course of training. These two parameters are then used to divide the state space into different areas from which goals are automatically selected naturally generating a curriculum.

All the methods described in this section create different curricula by only modifying the initial or terminal state distribution in the target task MDP. These techniques are extremely important in the curriculum learning literature because of their relative simplicity which makes them easy to apply to different environments. Nevertheless, these algorithms still imply some strong limitations in the definition of the intermediate tasks.

3.2.4 Changes on the Reward function

Another category of curriculum learning methods defines new source tasks by only acting on the reward function of the final task MDP similarly to reward shaping [56]. This allows for the design of algorithms able to suggest to the learner where to focus its exploration.

Sparse reward reinforcement learning problems are particularly sensitive to the exploration problem. Curriculum learning can be applied in this kind of setting by defining new intermediate tasks within the final task MDP by only changing the reward function to train a real world robotic agent to solve manipulation tasks [64]. An external scheduler selects which sub-tasks (intentions) need to be executed in order to improve the performance of the agent over the final task. The agent learns simultaneously how to solve these intermediate tasks via off-policy reinforcement learning in order to efficiently explore the environment.

In the context of first-person shooter games, curriculum learning has been applied by creating a more informative reward function in order to lead exploration in a reward shaping manner [96]. Here, for the purpose of obtaining efficacious curricula, the whole final task MDP is modified to create a new intermediate task. Several curricula are thereafter defined by using these two different MDPs and varying domain parameters controlling the advantage of the learner against the enemies it training to fight (for instance changing its health points and movement speed). Finally an adaptive curriculum learning algorithm shifts the learning agent towards more complex tasks in order to progress faster for completing the full final task.

The just described techniques are able to approach and solve extremely difficult reinforcement learning problems generating new source tasks by acting on multiple features of the final task. Nevertheless the creation of these intermediate tasks is still restricted to few possible modifications of the final task MDP.

3.2.5 Changes on the whole MDP

So far we have been describing curriculum learning methods that define new tasks by varying some environment features such as the initial/final state or the reward function, while keeping constant the rest of the final task MDP. On the one hand, these algorithms have the advantage of allowing the user to rapidly design several intermediate tasks, on the other hand, they usually solve a fairly specific problem or require some domain expert knowledge, failing so to scale and generalize well. In this section we introduce those algorithms that are the closest to the work discussed in this Ph.D. thesis. These methods relax the bottleneck given by the freedom of modifying only some parts of the final task MDP, letting the user create totally different MDPs and problems over which the agent can train before facing the final task. This clearly opens up to the creation of very diverse curricula within the same domain although loosing the ease of automatically creating new intermediate tasks.

Curriculum as a Graph

The most general way to represent a curriculum in this context is by using directed acyclic graphs. Differently from the previously introduced techniques, we are here able to draw experience from different source tasks in parallel which can be desirable for some particular reinforcement learning problems.

Each node in the graph can be set to represent one of the available intermediate tasks, and the graph edges can be automatically built in order to define a source - target relationship for transferring knowledge [82]. The graph organization strongly relies on the definition of a feature vector, which is a binary approximated description available for each task in the set of sources. This allows the intermediate tasks to be divided into sub-groups which are then reorganized into sub-graphs following a new heuristic, the *transfer potential*. This metric balances the usefulness/relevance of a task against how complex it is to learn it (approximated with the dimension of the state space).

An expansion of the previous work represents the domain as an object-oriented MDP where states consist of a set of objects [15]. In this way a task can leverage a more powerful feature descriptor that is not limited to a binary representation but is characterized by the objects in the task as well as the states, actions, reward and transition functions. In this way the curriculum graph can be built by only considering the objects characterizing each single intermediate task and new sources can be created by directly modifying these objects.

Despite the extreme flexibility of these techniques in being able to represent the most diverse curricula, organizing all the intermediate tasks in a graph introduces some important computational challenges and strongly relies on the design of an heuristic method to group all the source tasks into sub-graphs. In particular, these algorithms not only investigate how useful a specific intermediate task is for learning the final one, but also express its usefulness in being a source task for another source task. Therefore this type of representation can be undesirable in some specific settings where instead the curriculum learning problem can be constrained to designing a sequence of tasks.

Curriculum as an MDP

The algorithms in this section develop solutions to the sequencing problem in curriculum learning by formalizing it as an MDP. Because of this reason, in these cases it is possible to define a new artificial agent solving the problem of finding powerful sequences of tasks.

A curriculum MDP can be defined for a set of states representing all the possible policies that the agent can assume while training its weights [54]. On such an MDP, an action is defined as the next task over which the underlying learning agent has to train next, and the reward corresponds to the time spent learning such task. Using this technique it is possible to find a curriculum optimizing the time that it takes for the agent to converge

over the final task. It is also possible to prove how the curriculum found by this algorithm is strongly bounded to the specific learning agent and its sensing skills.

Similarly, a POMDP can be defined for these same state and actions while considering the agent internal weights inaccessible [49]. Here the reward is implemented as the increase of return value given by solving the currently selected task compared to the last time the agent approached it. Because of this reason, the objective of the curriculum agent is to train the underlying learner to efficiently solve all the given tasks rather that just the final one.

In neither of these works does the curriculum agent actually perform learning, but rather it uses a heuristic in order to select the next task to be sequenced. Despite this fact, it is possible to perform learning for the curriculum agent when the curriculum MDP states are directly represented by the function approximator weight vector of the reinforcement learning agent [55]. Interestingly this approach selects the next task for the learning agent depending on its current learning capabilities without necessarily needing to learn each task in the curriculum to convergence.

Representing the task sequencing problem as an MDP is a compact and elegant way to find powerful curricula for reinforcement learning agents which ease the curriculum learning problem from the complex graph representation previously described. The main drawback with this kind of approach is the computational time required to train the curriculum agent and how fast this time grows if we require it to be able to optimize for different agents in different domains.

Continuous Curricula

All the curriculum learning approaches we have considered so far in this chapter focus on how to sequence intermediate tasks after modeling the problem in different ways. Only recently, the research in this field showed how particular attention should also be given to how to best exploit the knowledge coming from each single task in a curriculum. Indeed, while most of the sequencing algorithms learn each task until convergence, it was proven that this is not always desirable, and more efficient methods can be found [55].

An interesting curriculum learning approach that takes this important aspect into account introduces the concept of *decay function* [7]. The purpose of this function is to control the degree of difficulty of the current task that the agent is training on, in order to change it in a continuous manner, allowing in this way the agent to progress on the most appropriate task for the right amount of time. This technique relies on the definition of a feature vector similar to the ones introduced by the research of Section 3.2.5. The decay function needs to be mapped to the feature vector in order to accordingly change the

degree of difficulty. This implies that it is possible to define a feature vector dimension for varying a task complexity.

Interestingly this approach can be applied in the most of the curriculum learning settings discussed in this chapter. On the one hand, it can be used in order to find better sequences of tasks than the ones proposed by the automatic algorithms of Section 3.2.3, and on the other hand this method can be applied in order to optimize the time spent over each task in a sequence found by another curriculum learning algorithm.

3.3 Other Curriculum Learning Paradigms

In this section we review curriculum learning studies that take a considerably different approach to the problem compared to the ones surveyed in the previous sections. Nevertheless, results and conclusions in these areas are of great influence in the whole curriculum learning field.

3.3.1 Teacher-Student Curriculum Learning

In Section 3.2.5 we studied algorithms that model the task sequencing problem in curriculum learning as an MDP. In this way, it is possible to introduce a curriculum agent whose purpose is to find curricula by solving this newly designed MDP. This can be obtained by either following a heuristic or directly learning how to solve the task sequencing problem.

Instead of modeling the sequencing problem as an MDP, this new agent (teacher) can be directly implemented within the same environment of the underlying learning agent (student) [80]. The teacher is assigned a specific reward function which trains it for adaptively proposing new problems to the student, which learns how to solve generic tasks in the given environment. In this way, the teacher iteratively proposes increasingly more difficult tasks to the student thus inducing a curriculum for exploration.

3.3.2 Human-in-the-loop Curriculum Learning

Machine learning is strongly related to fields like cognitive science and psychology. In Section 3.2.1 we discussed how curriculum learning was firstly applied in research related to language studies, in order to prove the validity of popular learning theories. So far we have analyzed algorithms that create and search for curricula either automatically or with the help of an external artificial agent. Here we discuss curriculum learning research studying the importance of human intuition in creating curricula for artificial agents.

Human domain expertise can be used for developing curricula for applications in artificial domains. NeuroEvolving Robotic Operatives (NERO) is one such example [77],

where a group of artificial agents evolve and adapt real-time in a strategy game setting, by following curricula of tasks specifically created by expert users.

Another interesting example of leveraging domain expertise for the design of curricula is applied in solving problems for a simulated robot soccer game [46]. This domain is particularly complex and the learner can benefit from learning different skills independently.

Instead of directly employing domain experts for the development of efficient curricula, naive human strategies can be used for studying new techniques for the automatic generation of curricula. This was the case for teaching object graspability to an artificial agent [41] or even in more complex sequential decision-making tasks [60].

Finally, direct naive human experience can also be used to improve performance of learning agents training in complex sparse reward Atari games [34].

3.3.3 Self-play

Self-play is a popular methodology that aims at improving the learning process in complex environments throughout the interaction of multiple learning agents. This technique has been widely explored for competitive multi-agent problems like in the case of TD-Gammon [89], AlphaGo [74] and AlphaStar [92].

From this field, the works most related to curriculum learning involve the design of an agent playing against the primary one. For instance, self-play was used to improve the generalization capabilities of a robotic system learning in simulation [62]. Here the adversary optimizes the return of its own reward function that fosters it to apply different disturbances to the given model in order to identify the hardest conditions under which the real-world agent might have to work.

In a multi-agent setting, instead, self-play was used to train two opposing teams of learning agents [5]. In this case the two teams were competing against each other on a game of hide and seek. Because of their interaction each team developed always stronger strategies to counteract the one of the opponents.

In many multi-agent settings it is crucial to develop an agent able to access a library of policies in order to learn different types of strategy to counteract different opponents. This is also a popular solution for multi-task reinforcement learning where the agent needs to memorize a number of policies for solving a set of final tasks and avoid catastrophic forgetting. In curriculum learning, instead, we are interested in optimizing the performance of the agent over only one final task.

3.3.4 Closely Related Fields

Curriculum learning represents one of many possible techniques that nowadays are used for improving the performance of state-of-the-art reinforcement learning agents. In this section we briefly discuss research fields that share with curriculum learning the problem of transferring knowledge coming from multiple tasks.

In *meta learning* [24], the objective of the learning agent is to rapidly adapt to a given new task by exploiting training over a number of source tasks. The main difference with curriculum learning here is given by the fact that there exist multiple final tasks instead of only one, and the objective is to learn how to solve each of them as fast as possible.

Multi-task reinforcement learning [95] studies algorithms for optimizing the learner performance over multiple reinforcement learning tasks. Also here, the main difference with curriculum learning is the absence of a single final task. The algorithms in this field train the learning agent for improving its generalization capabilities.

Lifelong learning [67] is a machine learning paradigm where an agent is continuously presented with new tasks. Similarly to multi-task learning, in lifelong learning it is crucial that the agent masters all the tasks it encounters, and therefore no particular importance is given to any task. Furthermore, in this case, the learner has no control of either the ordering of the tasks it needs to learn or their design.

3.4 Decomposition of the Curriculum Learning Problem

This Ph.D. thesis focuses on the task sequencing problem in curriculum learning. In this area of research there exist other two important problems that we discuss in this section and play a crucial role in the design of effective curricula: source task creation and transfer learning for curriculum learning problems. Algorithms performing automatic task creation, selection and transfer learning together will be able to offer us extremely powerful curricula. Nevertheless, it is important to notice how task sequencing represents a critical first step towards the creation of complete curriculum learning algorithms. An efficient task selection method can indeed work on a large set of intermediate tasks where only few of them are actually beneficial for a curriculum.

3.4.1 Source Tasks Creation

In our work we model task sequencing as a combinatorial optimization problem. This shows how the space of all the possible curricula rapidly grows with the number of available source tasks. In the most of the environments it is possible to randomly generate a big set of sources, so to ensure the presence of useful intermediate tasks without focusing on

the design of each single one. Nevertheless, even with the best sequencing algorithm, we would have no guarantee to find strong curricula in a practical time. Because of this reason, the most of the works discussed in this chapter rely on human domain expertise to create the set of source tasks. It is of crucial importance providing the sequencing algorithm with a set of intermediate tasks where each of them can positively impact the performance of the learner over the final task, in order to quickly find powerful curricula.

Some of the sequencing algorithms in Section 3.2 present an automated method for the source tasks creation, which is directly implemented in the definition of the sequencing technique. For instance, task sequencing algorithms that create curricula by changing the initial or terminal state distribution in the final task, can generate the next possible source tasks dynamically by iteratively performing a random walk on the previously generated initial states [26]. In alternative, it is possible to employ a GAN (Generative Adversarial Network) to design the next intermediate tasks by setting new goal states at each algorithm iteration [25]. Similarly, the state space can be automatically divided in different areas characterized by new possible terminal states of different utility for the learning agent [6].

On the one hand, these algorithms are compact solutions solving the sequencing and task creation problem together, on the other hand, this same characteristic prevents decoupling the two processes which therefore cannot generalize well over different environments curriculum learning settings. Source task creation is directly discussed in only a few research works. The earliest example is in a multi-task learning scenario, thus the objective is to optimize the performance of the agent over a number of different tasks [70]. In curriculum learning, source task creation follows specifically designed heuristics. When the given domain can be parametrized with a set of features for describing differences among a group of tasks, then new intermediate tasks can be automatically generated by changing these features values following different techniques [53]. A similar approach can be applied if, instead, each task is described following an object-oriented representation. Also in this case, new tasks can be created by modifying the attributes contained in the class descriptor by following heuristic methods [15].

All the just introduced methods still strongly rely on human domain expertise for defining the dimensions over which a task can be modified and the heuristics for automating the source creation process. More work on this particular problem can sensibly improve the performance of state-of-the-art task sequencing algorithms which would then be able to find strong curricula faster.

3.4.2 Transfer Learning for Curriculum Learning

A fundamental building block of any curriculum learning algorithm is the underlying transfer learning technique. This is the method responsible for effectively transferring

knowledge gathered over a source task to the next one in the curriculum. It is important to notice how a reinforcement learning agent training through an entire curriculum, is challenged not only by the difficulty of learning each single task in the sequence. The learner, indeed, needs to transfer its incrementally more complex knowledge to the next task in order to improve it even further. In this section we aim at reviewing the most popular transfer learning strategies used in curriculum learning in order to later contextualize the one chosen for this study.

In some particular cases, no transfer methodology is strictly required for implementing a curriculum learning algorithm. This is usually true for the algorithms described in Section 3.2.2, where a curriculum is defined as a a particular ordering of the experience samples collected during training. No transfer can also be an option for all those methodologies where the source tasks are defined in the same environment of the final task, although in those cases some transfer learning technique is usually implemented.

A popular transfer learning methodology is to transfer the policy learned at the end of the training over a source task. This is applied across different domains in different curriculum learning settings, like algorithms modifying the initial and final states distribution [25, 26], the reward function [64] or the whole MDP [7, 49] but even in the absence of autonomous sequencing [13].

An alternative transfer method in the curriculum learning literature, that is only used in the case of sequencing of source tasks created by modifying the whole final task MDP, is potential-based reward shaping [55, 82]. In general, reward shaping is a transfer learning algorithm previously used for transferring policies from a source to a target task [12] and can be used between any pair of tasks in a curriculum. This technique was previously proved to be equivalent to value function transfer [94] which in the context of curriculum learning is widely used for curricula with no particular restrictions on the intermediate tasks design [15, 35, 54, 55]. This latter is the methodology we decided to adopt for our experiments as they all fall in this category of curriculum learning. In Chapter 4 we will introduce the transfer learning technique we specifically designed for our experiments when using linear function approximators, while for artificial neural networks we employed Progressive Neural Nets [66], a method following similar principles.

A particularly powerful idea is also transferring high level knowledge such as options or skills. Despite its promising implications there exist only few autonomous task sequencing algorithms that implement this method [6, 38, 91, 98]. Transferring high level knowledge can be very effective in particularly complex domains, but it also requires extra computation time and human expertise.

Finally, it is also possible to transfer entire task models [60, 72, 96] but this is the least explored transfer learning methodology in curriculum learning.

3.5 Summary

In this chapter we started by analyzing the exploration-exploitation problem in reinforcement learning. Firstly we introduced exploration algorithms to be directly applied on the task we need to learn, before then moving the focus to transfer learning techniques that are able to exploit experience coming from another task in order to shape exploration.

Curriculum learning is an expansion of this concept, allowing a reinforcement learning agent to leverage experience accumulated over multiple intermediate tasks. We saw how there exist different curriculum learning settings that allow these source tasks to be created following different constraints and model the problem in various ways. Among these methods, those that let the source tasks be created without any limitation are the closest to our work, in particular the ones modeling the sequencing problem as an MDP.

None of the methods surveyed in this chapter formalizes the curriculum learning problem in such a way that it is possible to identify the optimal curriculum given the set of source tasks. This is due to the fact that we are generally interested in developing practical search methods to find curricula improving the performance of the agent over the final task rather than employing potentially extremely expensive methods to find the best possible curriculum. It is therefore difficult to understand how well a specific curriculum really is performing or what the real potential of a set of source tasks is.

Because of this reason, this Ph.D. thesis models task sequencing in curriculum learning as a combinatorial optimization problem performing an analysis of the overall complexity of finding an optimal curriculum in the space of all the possible curricula.

An earlier solution to the curriculum learning problem as combinatorial optimization, introduced four different heuristic searches based on different intuitions [35]. Even in this case, the curricula found following the proposed heuristic searches, are not compared with the global optima but solely with the performance of the agent directly learning the final task.

In the next chapter we introduce our curriculum learning framework composed of different performance metrics and task scenarios. We also perform a preliminary analysis of the curricula space generated by all the possible sequences of intermediate tasks for four different experiments focusing on the salient features of such space and the performance attained by the best curriculum.

Chapter 4

A Curriculum Learning Framework

As previously discussed in Chapter 3, there exist two main problems within curriculum learning research which directly derive from the field of transfer learning: source (intermediate) tasks creation and task sequencing. This work focuses on the latter without making any assumption on how the given source tasks set is created. This makes the algorithms and results of this research easy to integrate with source task creation techniques, which will allow for finding even more powerful curricula in future research.

There exist multiple approaches to the problem of task sequencing, and in this regard we introduce one only constraint to its formulation, which is limiting the definition of a curriculum to a sequence of source tasks without repetitions. As we will discuss in this chapter, as a result of this restriction it is possible to better study and visualize the problem of finding the optimal curriculum of source tasks in the space of all the possible curricula to be generated from the source tasks set.

In this chapter we start by introducing a framework for curriculum learning based on combinatorial optimization. Later we propose novel uses of this technique, which arise from choosing different objective functions. We then discuss and motivate the choice of the transfer learning algorithms adopted in our experiments along with the artificial domains used in this study. Furthermore, we show that curriculum learning can be successfully used to: improve the initial performance, take fewer suboptimal actions during exploration, and discover better policies. Lastly we perform a preliminary analysis of the problem on four different experiments drawing some initial conclusions justifying the detailed study of the next chapters.

4.1 Problem Definition

Let $\mathscr{M}$ be a set of MDPs, composed of $\mathscr{T} \subset \mathscr{M}$, a finite set of candidate tasks for the curriculum, and $\mathscr{F} \subset \mathscr{M}$, a finite set of *final* tasks. Furthermore, $\mathscr{M}$ is solely composed

of elements from either $\mathscr{T}$ or $\mathscr{F}$ and no candidate task can be a final task and vice versa, therefore $\mathscr{T} \cup \mathscr{F} = \mathscr{M}$ and $\mathscr{T} \cap \mathscr{F} = \emptyset$. The final tasks are the tasks the designer wants the agent to learn more efficiently through the curriculum. We assume that the agent learns each task until convergence, and that each task serves the purpose of learning one additional skill, therefore we define a curriculum as a sequence of tasks in $\mathscr{T}$ without repetitions:

Definition. [Curriculum] *Given a set of tasks $\mathscr{T}$, a* curriculum *over $\mathscr{T}$ of length l is a sequence of tasks $c = \langle m_1, m_2, \ldots, m_l \rangle$ where each $m_i \in \mathscr{T}$, and $\forall i, j \in [1, l]$ $i \neq j \Rightarrow m_i \neq m_j$.*

Let $\mathscr{C}_l^{\mathscr{T}}$ be the set of all curricula over $\mathscr{T}$ of length l. In the rest of the chapter we will drop the superscript wherever the set of candidate tasks is implicit in the context.

We define $\mathscr{C}_{\leq L} := \bigcup_{l=0}^{L} \mathscr{C}_l$ as the set of all curricula of length at most L. We represent with $\mathscr{C}_0$ the set containing the *empty* curriculum of length 0, denoted with $\langle\rangle$. The empty curriculum corresponds to learning the final tasks directly.

Given a performance metric $\mathscr{P} : \mathscr{C}_{\leq L} \times \mathscr{F} \to \mathbb{R}$, which evaluates curricula for a specific set of final tasks, we consider the problem of finding an optimal curriculum c^*, such that:

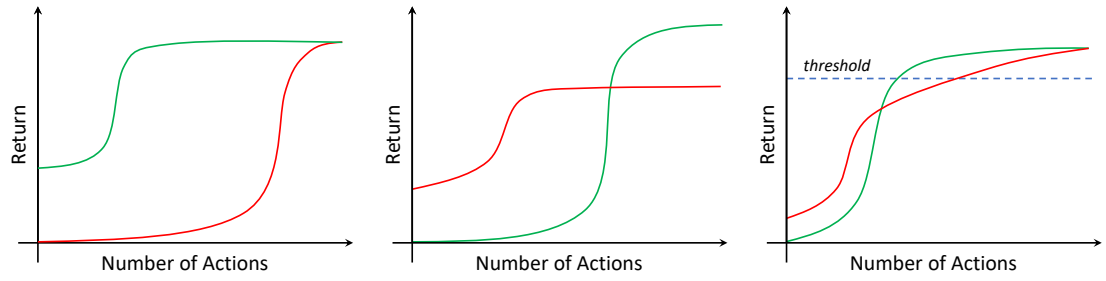
$$\mathscr{P}(c^*, \mathscr{F}) \geq \mathscr{P}(c, \mathscr{F}) \quad \forall c \in \mathscr{C}_{\leq L}.$$

4.2 Task Scenarios

In MDPs, if the value function can be represented exactly, and the exploration strategy guarantees that every action is executed in every state enough many times, the value function converges to the value of the optimal policy, regardless of its initialization. In tasks of practical interest, however, both the use of function approximators, and the need for a more limited exploration, determine convergence to a local optimum at best, which depends on the initial value. The role of the curriculum is to identify the optimal initial value, that is, to prepare the agent to the final tasks as best as possible. In the rest of this section we describe three scenarios with their requirements, and in the rest of the chapter we show how to apply the curriculum learning framework to them.

4.2.1 Critical Tasks

The first scenario is provided by *critical* tasks, where exploration is costly and suboptimal actions must be limited as much as possible while learning online. Examples of this case are abundant in robotics, where limiting exploration is one of the main concerns. We assume the existence of a simulator, also a common occurrence for such domains, since



(a) Critical Task (b) Complex Task (c) Time Sensitive Task

Fig. 4.1 A graphical representation of desirable (green) and undesirable (red) learning curves for each of the three proposed task scenarios.

learning cannot be performed in the real critical task directly. A set of final tasks, modeling the real task, can be set up in simulation, and the optimal curriculum computed without the need to act in the real task. We are primarily interested in optimizing the behaviour of the agent for the real final task, and we consider the time spent generating the curriculum in simulation as a sunk cost. This setting is common to many real-world applications, where simulators are available, and real-world exploration is costly. At the time of deployment, the knowledge achieved by the end of the last task of the curriculum is used to initialize the agent in the real task, since, by construction, this is the best possible initialization to the model of the real task used for training. In Figure 4.1a we show two learning curves achieving the same return in the same amount of learning time on an hypothetical real-world task. The use of curriculum learning in critical tasks aims at minimizing as much as possible expensive and dangerous exploratory behaviours (red line). This can be obtained by maximizing the area under the learning curve by improving the initial policy and/or rapidly moving towards more optimal areas of the search space (green line).

4.2.2 Complex Tasks

The second scenario corresponds to *complex* tasks. In tasks complex enough, the optimal policy is unknown, and the agent cannot be guaranteed to achieve it in any feasible amount of time. One example of such a case is the game StarCraft [73]. In this scenario, exploration is not a concern, as long as the agent achieves a policy of high value. Furthermore, training time is secondary, since the agent would take an enormous amount of time if not learning through the curriculum anyway. In this case, we are interested in the initial knowledge that can make the agent discover the best possible policy in the final task. In Figure 4.1b we graphically represent two learning curves of agents training to solve a complex task. The first one, in red, is quickly converging to a sub-optimal policy while adopting a safe exploratory strategy, while the second one, in blue, takes a longer time to converge and

behaves expensively for the most of the learning phase, but converges to a much higher quality policy. This is preferable in this task scenario as the performance of the agent will be computed by solely considering the behaviour obtained at the end of the training.

4.2.3 Time Sensitive Tasks

The third scenario is the one previously considered in the literature, and therefore we will only introduce it briefly. A large task is broken down into smaller subproblems, so that the agent can learn the optimal policy faster by learning the subproblems in sequence. For this type of task the only concern for the user is the time that it takes the agent to converge to the solution. In some cases we can also introduce a performance *threshold* to indicate the return value to be achieved by the agent in order to guarantee that the discovery of a satisfactory policy. In this setting, the performance threshold needs to be reached as soon as possible, regardless of the quality of the policy at convergence. We represent in Figure 4.1c two agents using different exploratory behaviours converging at to the same return value, though the one in green, does it faster than the red one.

4.3 Performance Metrics

In this section we describe the objective functions we propose to use in our framework, and assign them to the three newly introduced task scenarios for novel applications of curriculum learning.

The following metrics are adaptations of the most popular transfer learning methods for performance comparison to the problem of curriculum learning. We selected these in consequence of the strong correlation between the two fields although other specifically designed metrics could also be used, but this is not the focus of our study.

To the best of our knowledge, this work is the first considering all these metrics together, allowing for a more complete understanding of the field of curriculum learning itself and opening novel and important uses of it.

For simplicity, in this section we consider only one final task m_f, therefore each performance $\mathscr{P}(c, m_f)$ is calculated exclusively over it. The more general performance formulation can be computed for instance as $\mathscr{P}(c, \mathscr{F}) = \mathbb{E}[\mathscr{P}(c, m_f)]$ for each $m_f \in \mathscr{F}$.

Cumulative return (CR) is one of the metrics used to optimally balance exploration and exploitation in single-task learning, whereby the agent maximizing cumulative return attempts to converge to the optimal policy while acting suboptimally as little as possible. It is defined as:

$$\mathscr{P}_{cr}(c, m_f) \coloneqq \sum_{i=1}^{N} \mathbb{E}[G_f^i], \tag{4.1}$$

where G_f^i is the return obtained by the agent in the final task m_f at episode i, and N is the maximum number of episodes executed in the final task. Analogous objectives have been considered in the literature in the case of single-task exploration (regret [36]), and transfer learning (area under the curve [88], and area ratio [43]).

The following objective functions have been defined in the context of single-task transfer learning [88, 43], and can be imported into curriculum learning.

Jumpstart (JS) evaluates the average reward of the agent within the first D episodes:

$$\mathscr{P}_j(c, m_f) := \frac{1}{D} \sum_{i=1}^{D} \mathbb{E}[G_f^i],$$

where G_f^i is the return obtained during episode i in task m_f. Jumpstart can be used if it is crucial that the agent is deployed in the final task with the highest possible initial performance, and is a version of cumulative return that focuses only on a few initial episodes.

Jumpstart and cumulative return are objectives over the quality of the exploration (how it starts, and how it proceeds respectively) in the final task. Because of this direct correlation with the exploratory behaviour, they are evaluated on every training episode while still performing exploration and result perfectly suited for the critical task scenario.

In the following two metrics, we focus on the value of the learned policy rather than on exploration. The value of the actions taken during learning is not the objective of the optimization, and the agent aims at either getting to a certain performance level faster, or reaching a higher level. For this reason, every K learning episodes, we introduce an *evaluation* phase, in which the current policy is executed Q times with no exploratory actions, to estimate its expected return. During this phase, no updates are performed to the value function (or the policy). In the rest of this section, we denote with $\mathscr{E}$ the set of the evaluation steps.

The objective *max-return* (MR) focuses on the value of the policy learned within a given horizon:

$$\mathscr{P}_m(c, m_f) := \max_{I \in \mathscr{E}} \mathbb{E}[G_f^I],$$

where $G_f^I = \frac{1}{Q} \sum_{i=1}^{Q} G_f^i$ is the average return over the Q episodes in the evaluation step I on the final task. This is conceptually equivalent to *asymptotic performance* introduced for transfer learning [88], whereby the agent maximizes its performance by the end of learning. Max-return takes into account the non-monotonic nature of learning with function approximation, so that the best discovered behaviour may not be at the end of a trial, but anywhere during it. Max-return naturally fits the requirements and constraints imposed by a complex task thus being the optimal performance metric, among the ones proposed in this section, to optimize in this scenario.

Lastly, we consider the objective function that is currently the most used by the curriculum leaning algorithms described in Section 3.2.5, which are the ones closest to our work: *time-to-threshold* (TTT). It evaluates the number of actions executed throughout the curriculum in order to achieve a given threshold performance g during an evaluation step in the final task. Let $a(m_i)$ be the number of actions the agent executed in task m_i before moving on to the next task in the curriculum, and $a_g(m_f)$ be the number of actions the agent executed in the final task until the evaluation step in which the policy achieves an average return of g. The time-to-threshold metric is defined as follows:

$$\mathscr{P}_t(c, m_f) \coloneqq -(a_g(m_f) + \sum_{m_i \in c} a(m_i)),$$

where we intend to minimize the time to threshold, therefore the total time is multiplied by -1. Time-to-threshold is the only metric in which each task contributes to the total performance explicitly. In the other metrics the intermediate tasks affect the performance exclusively through transfer learning, and its effect on the behaviour in the final task. This situation is similar to having a reward in an MDP along the way, rather than only in the final state, and as such it is easier to build heuristics for it, which has been done in previous work [54, 15]. Nonetheless, it is in principle possible to define a weaker time-to-threshold as $\mathscr{P}_t(c, m_f) \coloneqq -a_g(m_f)$, where the training time during the curriculum is not taken into account. It is clear how, by adapting its description to the specific problem we need to solve, time to threshold is by definition best applied on time sensitive tasks.

4.4 Transfer of Knowledge

As we have previously discussed in Chapter 3, transfer learning is a powerful solution for shaping exploration in modern RL problems, by leveraging experience coming from another task. This same idea is used and extended in different ways in other research fields such as multi-task reinforcement learning, meta reinforcement learning or lifelong learning.

In curriculum learning an agent approaches a complex final task after training over a number of intermediate tasks. The learner needs to be provided with a transfer learning algorithm in order to effectively pass the experience coming from each source task through the whole curriculum. The particular knowledge to be transferred along the sequence varies depending on the specific implementation. The most popular choices at this regard are to either transfer policies or value functions, but the latter is often preferred for cases similar to the ones studied in this thesis [52]. Whenever the intermediate tasks are created by modifying any part of the final task MDP, indeed, the preferred knowledge to transfer is

the whole value or q-function attained at the end of the training over the task. Since this is exactly our setting we decided to use a transfer learning algorithm transferring q-functions.

For the experimental domains in this work, the q-function of each task is represented with a function approximator either linear or non-linear. In this chapter all the agents use Tile Coding which is a linear function approximator for which we have developed a specific transfer learning algorithm.

First of all, in order to favor transfer, we used an egocentric representation (using distances with respect to the agent) and local variables, as described separately for the two domains. We also normalized the variables in $[0,1]$, so that the input is invariant to the scale of the domain.

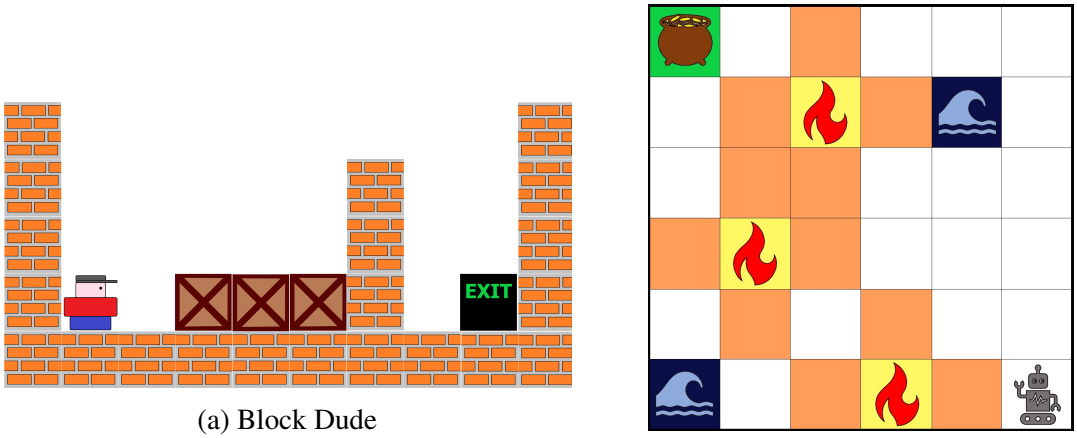
Furthermore, we used a particular value-function transfer [88] inspired by Concurrent Layered Learning [93]. In Concurrent Layered Learning, an agent learns a complex behaviour by incrementally learning sub-behaviours (layers). The more complex behaviours (higher layers) directly depend on the easier ones (lower layers), and during training all the layers are updated simultaneously.

This concept was implemented by carrying over the features of the source task into the target task, along with its parameters. Let V_i be the set of variables defined for the source task, and V_j the variables defined for the target task, in a source-target pair along the curriculum. Let $q_i(s,a) = \boldsymbol{\theta}_i^T \boldsymbol{\phi}_i(s,a)$ be the value function of the source task. The value function for the target task, q_j, is defined as:

$$q_j(s,a) := \begin{cases} \boldsymbol{\theta}_i^T \boldsymbol{\phi}_i(s,a) + \boldsymbol{\theta}_j^T \boldsymbol{\phi}_j(s,a), & \text{if } V_i \subseteq V_j. \\ \boldsymbol{\theta}_j^T \boldsymbol{\phi}_j(s,a), & \text{otherwise.} \end{cases}$$

where $\boldsymbol{\phi}_j$ is the feature vector of the target task. Therefore, if the variables of the target are compatible with the variables of the source, so that the features of the source are defined in the target, the features and their parameters are carried over. Otherwise the agent simply learns the new task without leveraging any transferred knowledge. The new parameters, introduced in the target, are set to 0 so that the imported features initially dominate the behaviour. In our experiments we do not remove features, and the number of features and parameters grows with every transfer. However, it is possible to perform feature selection, and remove the features that do not affect the value function significantly.

This transfer learning algorithm is designed to allow for effective knowledge transferability between pairs of similar tasks in order to ensure the creation of powerful curricula. Often, this resulted in extremely interesting exploratory behaviours where the agent could focus on its weaknesses rapidly moving towards the most interesting regions of the state space. In other words, in these cases the agent starts learning in the target task with a high



(a) Block Dude

(b) Gridworld

Fig. 4.2 Sample tasks for the two domains: Block Dude and Gridworld. In the first, the agent has to use movable boxes to navigate the map and get the exit block as fast as possible. Similarly, in Gridworld, the agent has to reach the treasure tile with the least number of actions while avoiding pits and fires.

quality policy that, during the exploration phase, only needs to be optimized for few states in order to converge to the optimal one.

4.5 Domains for Curriculum Learning

We performed a thorough experimental evaluation on two domains, with two sets of experiments each. We intended to show that curriculum learning is indeed a valid method to improve cumulative return, jumpstart, and max-return over learning from scratch, which makes it applicable to the new scenarios we consider.

The two domains, BlockDude and Gridworld, were implemented within the software library Burlap[1]. We used the implementation of Sarsa(λ) available in Burlap with $\lambda = 0.9$, learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.999$. We represented the action-value function through the Burlap Tile Coding function approximator, initializing the Q-values to $q(s,a) = 0$ for all state-action pairs. The agents explore with an ε-greedy policy, with $\varepsilon = 0.1$ and decreasing linearly from 0.1 to 0 in the last 25% of the episodes.

4.5.1 GridWorld

GridWorld, Figure 4.2b, is an implementation of an episodic grid-world domain used in the evaluation of existing curriculum learning methods [83, 15]. Each cell can be free, or occupied by a *fire*, *pit*, or *treasure*. The agent can move in the four cardinal directions, and

[1] http://burlap.cs.brown.edu

the actions are deterministic. The reward is -2500 for entering a pit, -500 for entering a fire, -250 for entering the cell next to a fire, and 200 for entering a cell with the treasure. The reward is -1 in all other cases. The episodes terminate under one of these three conditions: the agent falls into a pit, reaches the treasure, or executes a maximum number of actions (50). The variables fed to tile coding are the distance from the treasure (which is global and fulfills the Markov property), and distance from any pit or fire within a radius of 2 cells from the agent (which are local variables and alone do not fulfill the Markov property but allow the agent to learn how to deal with these objects when they are close, and transfer this knowledge).

4.5.2 BlockDude

BlockDude, Figure 4.2a is another domain available in Burlap, which has also been used for curriculum learning [83]. It is a puzzle game where the agent has to stack boxes in order to climb over walls and reach the exit. The available actions are moving left, right, up, pick up a box and put down a box. The agent receives a reward of -1 for each action taken. The variables used as input to tile coding are distance from the exit, distance from each box, distance from each edge of the map, direction of the agent (binary) and whether or not it is holding a box (also binary).

4.6 Preliminary Analysis

In this section we perform a preliminary analysis of the curriculum learning problem based on the dataset collected over the four experiments in the two just introduced artificial domains. We intend to deduce some initial conclusions and directions in order to follow up on these in later chapters. More specifically, here, we aim at identifying the most promising subproblems within the curriculum learning field. The idea is to develop a sequencing algorithm for the optimization of one or more performance metrics based on the results of these four experiments in BlockDude and Gridworld.

4.6.1 Experiments

We chose two relatively small domains so that we could perform a thorough evaluation, by computing and analyzing all curricula within the given maximum length.

In our experiments, we perform an evaluation phase each $K = 10$ episodes in order to estimate the quality of the learned policy. As the environments are deterministic, we can perform each evaluation step $Q = 1$ times. Each curriculum has been executed 10 times, and its value for each metrics estimated as the average over those trials. It is

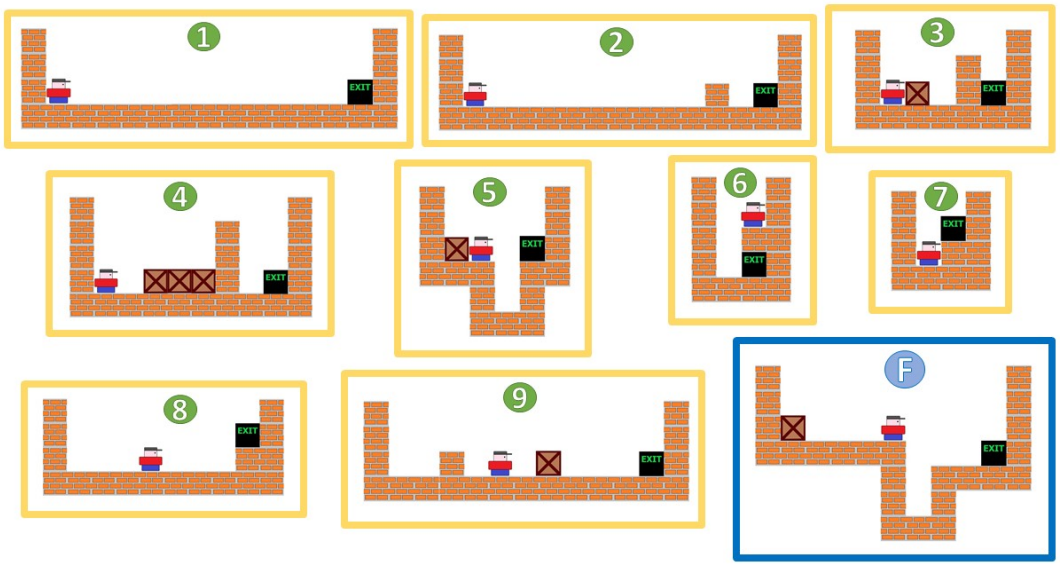


Fig. 4.3 Intermediate (in yellow) and final task (in blue) of the second experiment in the BlockDude domain. The global optimum is the curriculum 7-1-5.

necessary to execute each curriculum multiple times since the learner needs to perform exploration in each single task composing the curriculum. All these exploration phases implied in learning a curriculum can lead to considerably different policies for each task in the sequence. Moreover, the effect of these exploration phases has a greater impact on those tasks that are learned late in the curriculum as their execution will depend on multiple previous random explorations.

We ran two sets of experiments per domain, one in which the number of tasks is high and the maximum length is low, and one in which, on the contrary, the number of tasks is low, but the maximum length is high. For Gridworld, the first set of experiments has parameters $n := |\mathscr{T}| = 12$ and $L = 4$, while the second $n = 7$, and $L = 7$. For BlockDude, the first set of experiments has parameters $n = 18$ and $L = 3$, while the second $n = 9$ and $L = 5$. These parameters were chosen so that the total number of curricula does not exceed 20000. For both domains, the intermediate tasks have been generated manually, by varying the size of the environment, adding and removing elements (pits and fires in GridWorld, and columns and movable blocks in BlockDude). Figure 4.3 shows all the intermediate tasks, and the final task (marked with F) for one of the two BlockDude experiments. All experiments have a different final task, and set of intermediate tasks. All tasks are run for a number of episodes that ensures that the agent has converged to the optimal policy, and were determined at the time of task generation.

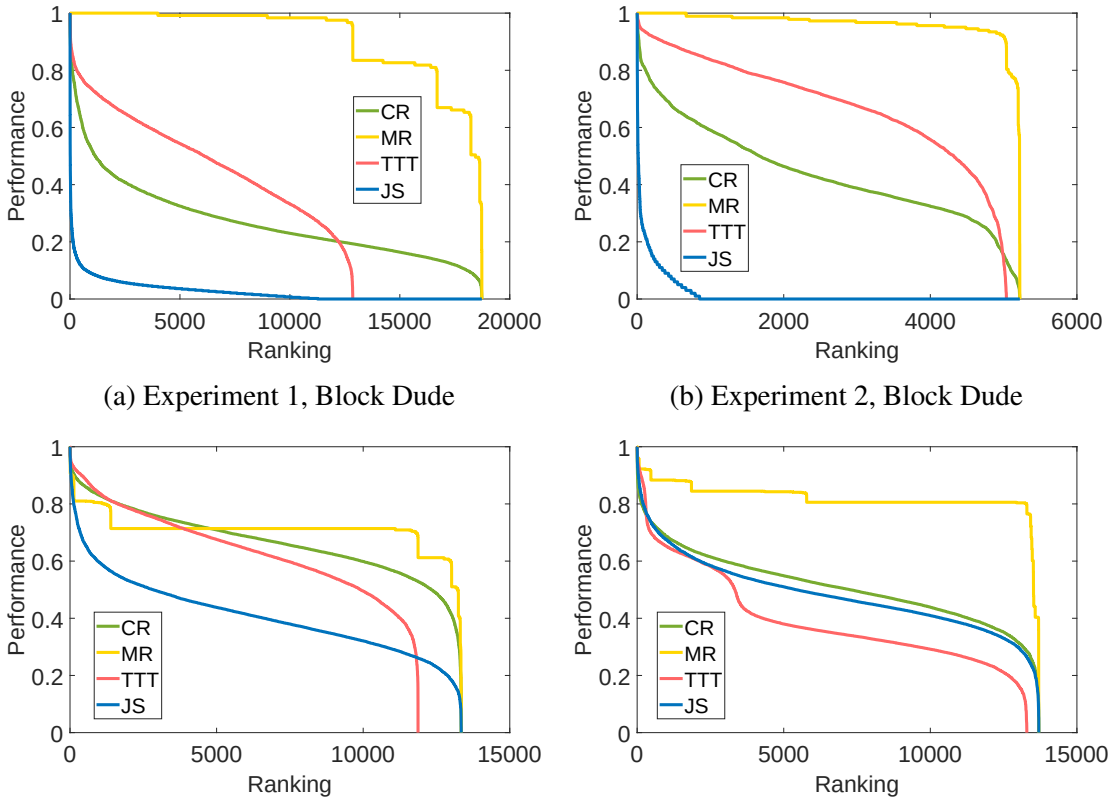
4.6.2 Optimization Complexity for Performance Metrics

We start our analysis by studying the difficulty of optimizing each of the performance metrics introduced in Section 4.3 with curriculum learning. Figure 4.4 shows four performance plots, one per experiment, where the trend of each performance is represented by a line of a different color.

In all the plots, the x-axis spans the curricula rank from the best, in position 0, to the worst, in the last position correspondent to the total number of curricula run in the given experiment. On the y-axis, instead, we have the performance value for each curriculum.

Since the aim of these plots is to allow for a comparison among the optimization problems associated to the reported metrics, it is important to notice how each performance is plotted independently from the others. This means that the value on the x-axis actually represents four different curricula on the same experiment, sharing the same rank on different performance. In other words, the i^{th} best curriculum in terms of cumulative return is different from the i^{th} best curriculum for the time-to-threshold performance in the same experiment. In order to ease the visual comparison of these performances, all the metrics are also normalized in $[0, 1]$, with 0 representing the lowest score, and 1 the highest. Because of this reason, in each plot the four functions are semi-monotonic non-increasing and span the same values both on the x and y-axis.

In this context, a curve with an initial steep slope represents a difficult performance which is interesting for optimization. This is because, in such cases, only few curricula are able to attain nearly optimal values for that specific metric. Moreover, curricula in such a region of the curve have quite different values with the next and previous in rank, in which case advancing of even a few rank positions results in a large performance improvement. In our case, this is particularly evident for jumpstart (in blue), especially in the Block Dude experiments, followed by cumulative return (in green) and time-to-threshold (in red). On the contrary, max-return (in yellow), always shows extended flat performance regions above the 70% of the maximum score. This feature characterizes a performance metric whose optimization problem is easier to solve. It is indeed possible to see how, apart for few curricula located at the extremes of these curves, the most of them have a relatively high max return value. In such a case, developing an efficient sequencing algorithm for curriculum learning is of minor concern since a randomly selected curriculum is likely to score more than 70% of the maximum value. This is because the domains and final tasks used for these experiments are relatively easy to train on, thus the agent can eventually learn the optimal policy with nearly all the given curricula. It is possible to notice that, instead, a randomly selected curriculum would perform poorly for the other performance metrics in the most of the cases, and consequently the search of an advantageous curriculum for their maximization would benefit from an automated task sequencing algorithm. Furthermore,



(a) Experiment 1, Block Dude

(b) Experiment 2, Block Dude

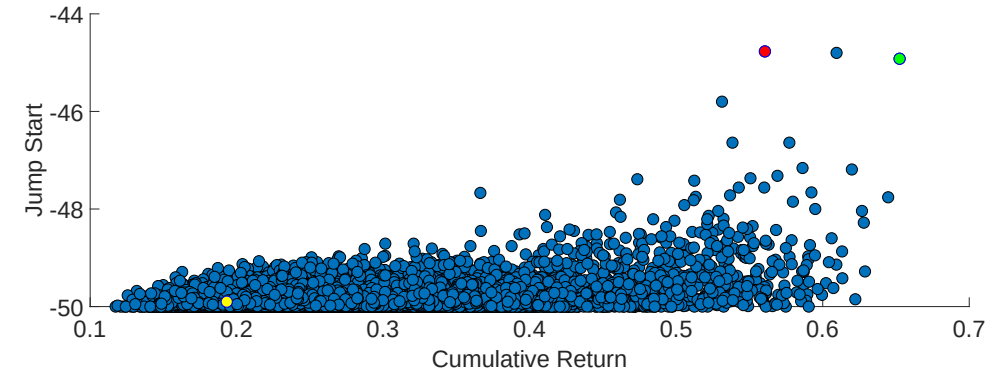
(c) Experiment 3, Gridworld

(d) Experiment 4, Gridworld

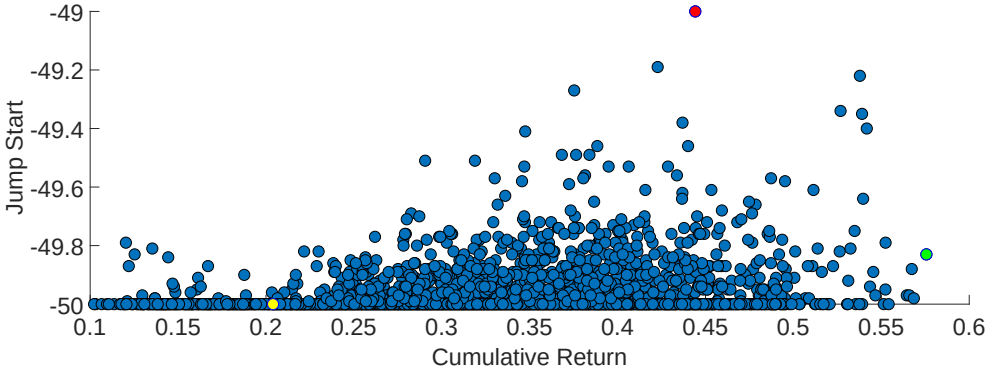
Fig. 4.4 Performance plots for the four experiments in the block Dude and Gridworld domains. The x-axis represents the curriculum rank, sorted from the first position to the last (from the highest performance to lowest), while the y-axis reports the correspondent normalized performance value. For comparison purposes, for each experiment, the four metrics are hereby drawn together (yellow max-return (MR), green cumulative return (CR), red time-to-threshold (TTT), blue jumpstart (JS)). The curriculum correspondent to a specific rank on the x-axis is different for each of the performance metrics.

these performance curves in these experiments are smooth, therefore it is possible to develop algorithms optimizing them.

It is important to underline how these conclusions are relative only to the experiments conducted in this study, therefore the shape of these functions can be extremely different in other domains. Nonetheless, we can say that selecting a random curriculum can be inappropriate for certain domains and it is important to study sequencing algorithms able to efficiently search the space of all the possible curricula in order to avoid counterproductive ones while quickly move towards the most promising areas of these performance.



(a) Experiment 1, Block Dude



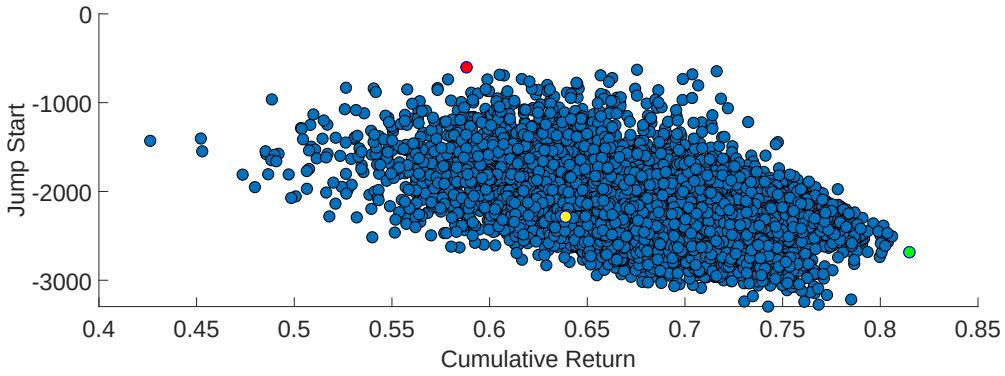
(b) Experiment 2, Block Dude

Fig. 4.5 Curricula plot in the CR/JS plane for the Block Dude experiments. Each point in the plot is a different curriculum. The green and red dots respectively represent the best curricula in terms of cumulative return and jumpstart performance, while the yellow one is the empty curriculum C_0.

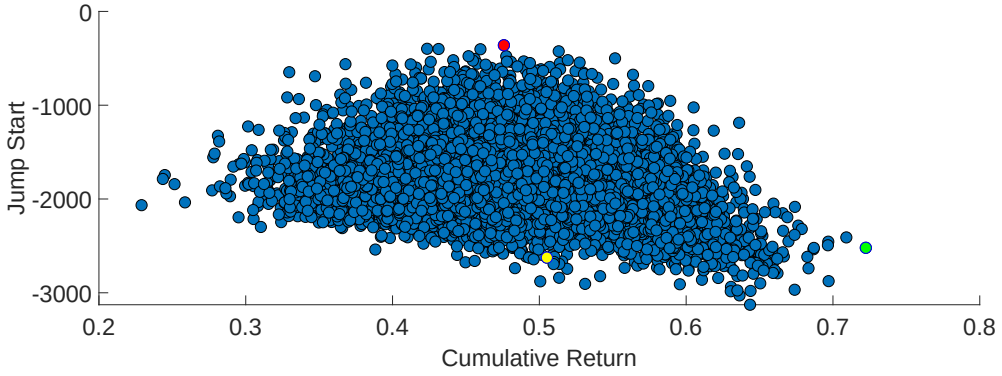
4.6.3 Decoupling Performance Optimization

In this section we want to continue our performance analysis by focusing on the critical task scenario, which is the case considered the least in literature. We aim at studying whether it is possible to maximize both the dedicated performance metrics and how these compare with the popular time-to-threshold.

We start by considering the two performance metrics we selected for the critical task scenario: jumpstart and cumulative return. Their definition shows how they are strongly connected to each other as JS is the result of the same calculations involved for CR with the only difference that the former is performed on a subset of the episodes of the latter. Nevertheless we cannot assume that these two metrics can be optimized together since starting the training with a high quality policy does not give any guarantee on the overall behaviour of the agent. Similarly, cumulative return maximization is the result of a safe exploration policy that acts during the whole learning phase without any particular focus on



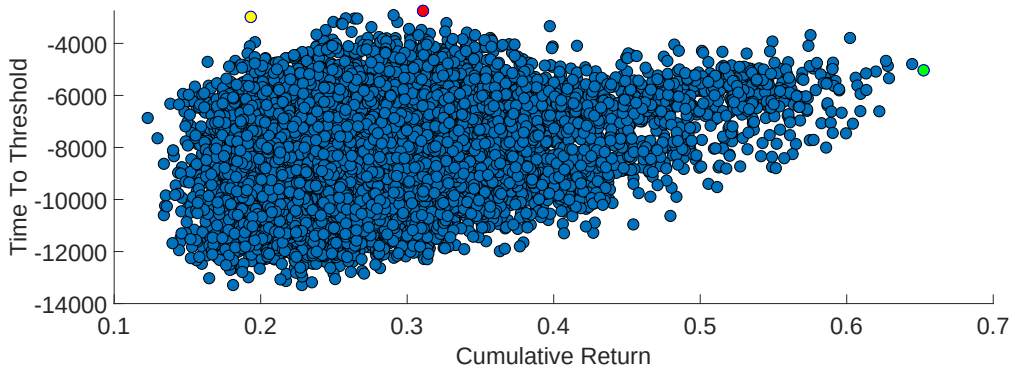
(a) Experiment 3, Gridworld



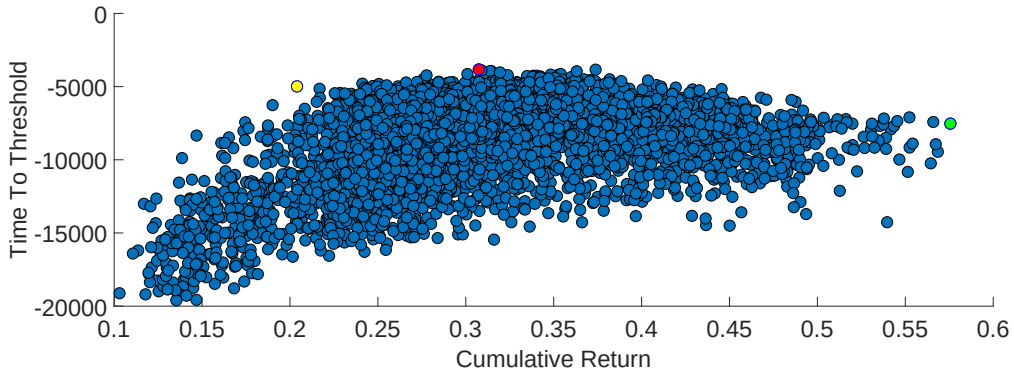
(b) Experiment 4, Gridworld

Fig. 4.6 Curricula plot in the CR/JS plane for the Gridworld experiments. Each point in the plot is a different curriculum. The green and red dots respectively represent the best curricula in terms of cumulative return and jumpstart performance, while the yellow one is the empty curriculum C_0.

any episode. We can observe this in Figure 4.5 and 4.6 where, for each experiment, all the possible curricula are plotted together on the cumulative return / jumpstart plane. Here, the x-axis and y-axis respectively span the cumulative return and jump start values assumed by all the curricula in the experiment. The cumulative return metric is hereby normalized again in $[0, 1]$, although this time, 0 corresponds to the worst possible policy never improving the agent performance during the training, while 1 stands for directly starting with the optimal policy. The green and red dots respectively represent the best curricula in terms of cumulative return and jumpstart performance, while the yellow one is the empty curriculum C_0. We can notice how finding the right curriculum is an important problem as, in these experiments, C_0 does not achieve high values on either performance metric although there exist curricula achieving worse performance than learning from scratch (see A for more information at this regard). The best curriculum for each performance also tends to score poorly on thee other one, especially in the Gridworld experiments. Furthermore, in these plots a curriculum achieving a high value on both performance is located in the



(a) Experiment 1, Block Dude

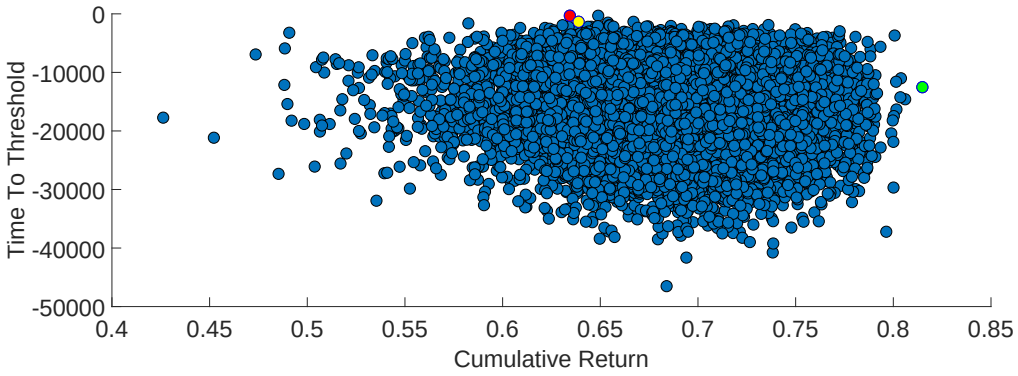


(b) Experiment 2, Block Dude

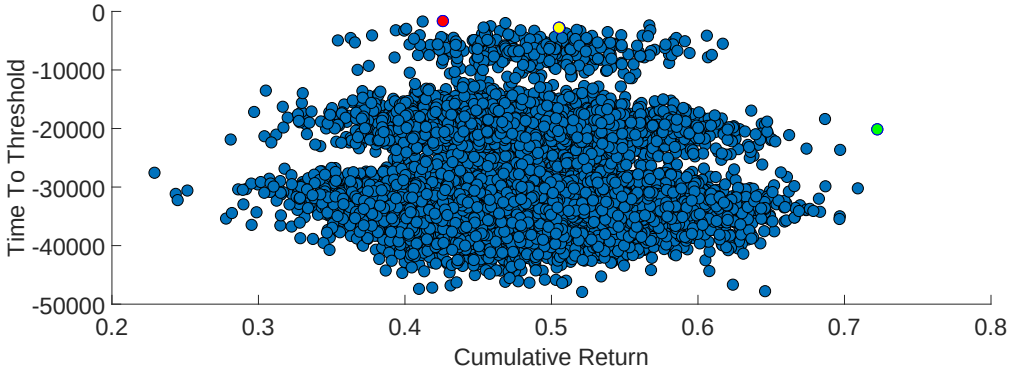
Fig. 4.7 Curricula plot in the CR/TTT plane for the Block Dude experiments. Each point in the plot is a different curriculum. The green and red dots respectively represent the best curricula in terms of cumulative return and time-to-threshold performance, while the yellow one is the empty curriculum C_0.

top-right corner of the given plane. As we can see, this region of the space does not present any curricula for the Gridwold domain. The situation is different for the Block Dude experiments where we have some elements sparsely populating this optimal performance area. Therefore we can conclude that in general we cannot assume that cumulative return and jumpstart can be maximized together, so for solving a problem in the critical task scenario we need to focus on one of them.

Since cumulative return captures the overall quality of the learning curve we decide to expand further on this one by confronting it with the most popular performance metric in curriculum learning: time-to-threshold. Figure 4.7 and 4.8 show plots similar to the ones previously described. Here each curriculum is represented by a dot in the cumulative return / time-to-threshold plane and, differently from before the red dot is the curriculum scoring the highest time-to-threshold value. Once again we can observe how, apart for experiment 3, only a minority of curricula populates the top-right corner of the plane representing the optimal region for both the performance metrics. Therefore we cannot assume that



(a) Experiment 3, Gridworld



(b) Experiment 4, Gridworld

Fig. 4.8 Curricula plot in the CR/TTT plane for the Gridworld experiments. Each point in the plot is a different curriculum. The green and red dots respectively represent the best curricula in terms of cumulative return and time-to-threshold performance, while the yellow one is the empty curriculum C_0.

cumulative return and time-to-threshold can be optimized together in any given problem, but we are usually obliged to select only one of them. Furthermore, C_0 always obtains a time-to-threshold value extremely close to the optimal one. This is due to the fact that in these experiments we do not optimize the number of actions spent over each task in the curriculum, consequently the most of the curricula perform worse than learning from scratch. Time-to-threshold optimization is best suited for cases where the sequencing algorithm also adapts the time dedicated to each task accordingly with the curriculum.

4.7 Summary

In this chapter we introduced and formulated the problem of tasks sequencing in curriculum learning for reinforcement learning applications. We then discussed three possible task scenarios for novel uses of curriculum learning, and we showed how it is possible to adapt

popular transfer learning performance metrics to formulate the correspondent optimization problem in CL. Lastly we described the experiment setting used to collect the dataset used in this thesis and performed a preliminary analysis.

In our experiments, a randomly selected curriculum would obtain an high performance value on max-return because of the domains and final tasks simplicity. In spite of this fact, it is in general extremely important to develop sequencing algorithms for the optimization of any of the given performance metrics in order to discover powerful curricula.

We observed how, optimizing multiple performance metrics with a single curriculum, represents an extremely complex challenge, and therefore we intend to further develop our study in the next chapters on curriculum learning for tasks in the critical task scenario only, as this is the least studied CR application.

In spite of this fact, it is also important to notice how while optimizing one metric it is still possible to put boundaries on the others. This would allow to discover optimal curricula for a certain performance with specific guarantees on one or more of the other metrics. We believe this to be of particular interest for real-world applications where the reinforcement learning task under study does not strictly fall into one of the described task scenarios.

Time-to-threshold is the most popular performance metric for sequencing algorithms that, like in our study, consider curricula composed of source tasks generated without putting any restrictions on their MDPs. This metric inherits two different problems from its formulation. The first one is about the definition of the performance *threshold* to be reached by the agent. This is usually domain dependent, and needs to be set differently for each single reinforcement learning task. The second issue is the formal definition of the associated optimization problem. Its aim is indeed to minimize the time spent training over the final task, after learning the chosen curriculum, before reaching the aforementioned threshold. None of the sequencing algorithms solving the TTT optimization problem, though, takes into account the time spent over the search of the optimal curriculum itself, which would clearly worsen the performance of these approaches. Time-to-threshold needs to include this issue in its formulation, or alternatively can be used for curriculum learning applications where no curricula search is performed. Moreover we noticed how a sequencing method that does not optimize the time spent on each task in the curriculum tends to generate only few beneficial curricula for the given final task.

Jumpstart and cumulative return present similar formulations and the latter can actually be expanded to include the former if needed. Both these metrics are perfectly applied to the critical task scenario and do not require to consider the time spent searching for the best curriculum. Furthermore, jumpstart shapes the behaviour of the learner only in the initial steps. This can lead to extreme scenarios where the agent starts the training with a good policy that is rapidly forgotten obliging the agent to start the learning phase from

scratch. Optimizing cumulative return provides the agent with stronger guarantees over the learning behaviour for the whole duration of the training.

In the next chapters, we first focus on general task sequencing algorithms to be applied on any of the introduced performance metrics and then we decide to develop our study further only on those most appropriate for cumulative return maximization.

Chapter 5

Algorithms for Task Sequencing

In the previous chapter we introduced four different performance metrics derived from the field of transfer learning. We performed a preliminary analysis based on the collected dataset on two artificial reinforcement learning domains and this allowed for appreciating the complexity and importance of the associated optimization problems. We concluded that it is not possible for a curriculum learning agent to sequence tasks optimizing more than one performance metric at a time and, for the sake of our study, we are particularly interested with cumulative return and its practical implications.

Nonetheless, each optimization problem for each of these metrics is defined in an equivalent way and only varies on the calculation of the performance of the learner in the final task. Because of this reason we can apply the same search algorithms for the optimization of any of these metrics as we will later see in the next sections.

In this chapter we introduce popular metaheuristic algorithms for solving the general curriculum learning optimization problem. Later, we further discuss the formulation of cumulative return and define a sequencing algorithm specifically designed for finding the best curricula for its optimization.

5.1 Metaheuristic Algorithms Adaptation

The objective functions defined in Chapter 4 do not have an explicit closed-form definition, since the actual return obtained by the agent can only be measured during learning. Indeed, the return G_f^i obtained by the agent in each episode for a given final task is a random variable, which depends on the dynamics of the task, the initial knowledge of the agent, and the exploration algorithms employed. The expectation cannot be computed exactly, and must be estimated from a number of trials. The resulting objective function does not have an explicit definition, therefore the resulting optimization problem is black-box, and it is in general nonsmooth, nonconvex, and even discontinuous. Furthermore, the

Algorithm 1 GenerateCandidates

Input: `seeds`, $\mathscr{T}$
Output: candidate set C

```
C ← ∅
for c ∈ seeds do
    E ← {m_i ∈ 𝒯 | m_i ∉ c}
    for m ∈ E do
        append m to c
        C ← C ⋃ c
    end for
end for
return C
```

optimization problem is constrained to a combinatorial feasible set. These characteristics do not allow us to resort to methods for general Mixed-Integer NonLinear Programs, or standard Derivative-Free methods. The most appropriate class of optimization algorithms for this type of problem is the class of metaheuristc algorithms, introduced in Chapter 2.

We selected four popular and representative algorithms from this class, two of which are trajectory-based, while the other two are population-based. In this section we describe the customization we performed to these otherwise general algorithms, to apply them for task sequencing. We set out to evaluate them experimentally, in order to determine which ones are the most appropriate for curriculum learning.

5.1.1 Trajectory-based methods

As a simple baseline, we use a purely *greedy algorithm*. This is meant to be an easy and fast solution which could be the only available option for particularly complex environments. The greedy search starts from a set of candidate curricula composed by all the curricula of length 1. These are then evaluated for selecting the best candidate. The best candidate is then given as input to `GenerateCandidates`, to obtain the next set of candidates. The algorithm terminates when the best candidate does not improve on the current best curriculum (hence its greedy nature). The function `GenerateCandidates` is shown in Algorithm 1. The new set of candidates is computed by appending, to each of the *seed* curricula, all tasks that do not already belong to that curriculum. For example, if $\mathscr{T} = \{m_1, m_2, m_3\}$, and `GenerateCandidates` is invoked on $\{\langle m_1 \rangle, \langle m_2 \rangle\}$, it returns $\{\langle m_1, m_2 \rangle, \langle m_1, m_3 \rangle, \langle m_2, m_1 \rangle \langle m_2, m_3 \rangle\}$. Greedy Search is the simplest deterministic local search algorithm, and is therefore easily prone to stop at a local maximum. Next we consider stochastic algorithms able to deal with more flexible definitions of locality.

In *Beam Search* [45, 58] we start from an empty sequence of source tasks. At each step we select the most promising solutions based on their performance $\mathscr{P}$ and we further

develop them by using `GenerateCandidates`. In this way, at each step, the algorithm evaluates solutions of the same length. The number of solutions to be expanded at each step is the beam width w, and the algorithm terminates once the maximum allowed length is reached. In our experiments $w = |\mathscr{T}|$, the total number of intermediate tasks in the source set.

For *Tabu Search* (TS) [29, 30], we define the fitness function $\mathscr{F}_t$ for a candidate curriculum to be equal to $\mathscr{P}$. We start the search by randomly selecting a curriculum in $\mathscr{C}_{\leq L}$. At each iteration we perform local changes to the current best solution to generate the relative neighborhood. We first generate a list of curricula R composed of all the curricula obtained by adding or removing a task from/to the last task in the current best curriculum. Then we generate all the curricula resulting from any pairwise swap of any two tasks of any curriculum in R. The size of our tabu list is T, and, when full, we empty it following a FIFO strategy. If during the search all the curricula in the neighborhood are in the tabu list the new current best solution is randomly selected. The algorithm terminates after a fixed number of iterations. In our experiments $T = 30$.

5.1.2 Population-based methods

In a *Genetic Algorithm* (GA) [31], we set the initial population as U randomly sampled curricula from $\mathscr{C}_{\leq L}$ and, similarly to Tabu Search, we set the fitness function $\mathscr{F}_g$ for a candidate curriculum to be equal to $\mathscr{P}$. At each iteration of the Genetic Algorithm we select two parents from the current population N_g with a roulette wheel selection. Given a candidate curriculum c_i, its probability of being selected as a parent is $p_i = \mathscr{F}_g(c_i)/\sum_{c\in N_g}\mathscr{F}_g(c)$. Given two parent curricula we generate a new population of U candidate curricula by applying a standard single point cross over at randomized lengths along each parent gene (sequence of tasks). Each cross over step produces two children curricula, and the process is repeated until U children curricula are created. We also included a form of elitism in order to improve the performance of the algorithm by adding the parents to the population they generated. Genetic Algorithms also include the definition of a mutation operator. In our implementation this acts on each candidate curriculum in the newly generated population with probability p_m. The mutation can be of two equally probable types: task-wise mutation, which given a candidate curriculum of length l changes each of its intermediate tasks with probability equal to $1/l$; length-wise mutation, where equal probability is given to either dropping or adding a new task at a randomly selected position of a candidate curriculum. In the case of candidate curricula composed of one task only, the dropping option for the length-wise mutation does not apply. The algorithm terminates after a fixed number of iterations. In our experiments $U = 50$ and $p_m = 0.5$.

Ant Colony Optimization (ACO) [17] is a CO metaheuristic which consists of deploying multiple agents (ants), in the given search space for depositing artificial pheromone along the most successful trails in that space, thus guiding the search towards more and more successful solutions. Each agent starts from an empty sequence of tasks. At each step an agent moves towards the goal by adding a new task to the current candidate curriculum c which represents the trail walked by the ant. Given a task m_i, its probability of being selected is $P(m_i) = [(\tau_{m_i} + K)^{\alpha} + I_{m_i}^{\beta}]/[\sum_E [(\tau_{m_j} + K)^{\alpha} + I_{m_j}^{\beta}]]$ with $E = \{m_j \in \mathscr{T} | m_j \notin c\}$.The visibility I_{m_i} is calculated as the performance improvement obtained by adding task m_i to the current candidate curriculum when positive, and zero otherwise. Parameters α and β control the influence of the pheromone versus the improvement while K is a threshold to control from what pheromone value the search starts to take it into account. Once the maximum curriculum length is reached, artificial pheromone is deposited on all curricula explored in this way, from the first to the best along the last trail, concluding in this way an iteration. The pheromone evaporation rate is specified with the parameter ρ while the maximum level of pheromone to be accumulated over a candidate solution is set to f_{max}. The algorithm terminates after a fixed number of iterations. In our experiments $\alpha = 1, \beta = 1.2, K = 5, f_{max} = 50, \rho = 0.2$ and the number of ants is 20. The parameters of all algorithms have been fine-tuned manually across all experiments.

5.2 Cumulative Return Maximization

In Chapter 4 we introduced four different performance metrics and identified three task scenarios explaining which metric would apply best in each of them. In the preliminary analysis we have also observed how the cumulative return performance metric is the most interesting one in the context of this thesis because of its real-world applications.

In this section we develop a heuristic algorithm for efficiently performing curricula search for cumulative return maximization. In order to do so we need to start by reformulating part of the optimization problem.

We have previously defined cumulative return in Equation 4.1 but its optimization would likely lead to poor performance and discover suboptimal curricula for real-world problems in the critical task scenario. This is mainly due to the fact that its definition is purely theoretical and does not take into account the challenge introduced by its application on an actual real-world task.

Cumulative return is indeed computed directly on the final tasks $\mathscr{F} \subset \mathscr{M}$, and this, in artificial domains, would both lead the search and represent the actual performance improvement introduced by the curriculum under analysis. As we will later show in this

section, for real-world domains we need to decouple these two aspects of the curriculum search.

We target the following scenario: one or more critical tasks of interest must be learned online, by limiting suboptimal actions as much as possible. The aim of the curriculum is to provide the best possible initial knowledge so as to shape exploration in the final tasks. We assume that a simulator is available to train the agent while generating the curriculum. Remember $\mathscr{F} \subset \mathscr{M}$ the finite set of MDPs constituting the *final* tasks. These are the tasks of interest, and for a curriculum to be valuable, it must provide an advantage over learning the final tasks directly.

We consider the problem of finding an optimal curriculum c^* of a given maximum length L, maximizing the cumulative return over all final tasks:

$$\mathscr{P}(c,\mathscr{F}) = \sum_{m_f \in \mathscr{F}} \mathscr{P}_{cr}(c, m_f)$$

$$\begin{aligned} &\max \mathscr{P}(c,\mathscr{F}) \\ &\text{s.t. } c \in \mathscr{C}_{\leq L} \end{aligned} \tag{5.1}$$

This optimization problem is entirely solved in simulation, that is, all tasks, including the final ones, are simulated tasks. Simulated final tasks are models of the expected real tasks, and, as previously mentioned, having more than one prevents the curriculum from overfitting to a particular simulated task.

5.2.1 An Heuristic Algorithm for Task Sequencing

While metaheuristc algorithms are quite general and broadly applicable, it is possible to devise specific heuristic methods targeted at particular problems. In this section, we introduce Heuristic Task Sequencing for Cumulative Return (HTS-CR).

Intuition

In this context the curriculum learning problem is defined as finding a powerful sequence of source tasks in order to improve the cumulative return performance over the given final task. This particular restriction on the optimization problem lets us study the shape of the most effective curricula created at this purpose. It is indeed now possible to find and analyze common features among the optimal sequences of tasks across the experiments in the two artificial domains used for our dataset.

We have previously discussed how curriculum learning is a research field derived from transfer learning. In particular, curriculum learning uses multiple transfer processes to effectively move experience along a whole curriculum, up to the final task. For each source

in a curriculum, the learning phase is initialized with the knowledge coming from the previous task in the sequence and, at the end of the training, the agent transfers the resultant experience to the next task in the curriculum.

Because of this dynamics, in curriculum learning it is extremely complicated to determine whether or not an intermediate task could actually be beneficial for the final task. In classic transfer learning, indeed, the effect of a source task can be directly measured on the target task on which it is applied. Conversely, in curriculum learning the impact of an intermediate task on the final task can take place in different ways, and in most of the cases its effect is indirect. For instance, this is the case for all those tasks in a curriculum preceding the last task in the sequence, which is the only one whose knowledge is then directly applied on the final task.

Each source composing any curriculum in this study is preceded and followed by another task in the source set, with the exception of the first and last intermediate tasks in the curriculum, which represent special cases. It is possible to deduce that, the contribution of these two particular tasks in the curriculum, can be easier to assess compared to the others. The idea is that, for example, the first task, *head* of a curriculum, is always learned without exploiting any prior knowledge, therefore it generally needs to be small and easy to learn. Conversely, the last task in the sequence, *tail* of the curriculum, has to be similar to the final task as its knowledge is directly transferred on it.

The HTS-CR Algorithm

We take advantage of the previously introduced insight: the quality of a particular task sequence is strongly affected by the efficacy of knowledge transfer, and transfer between certain pairs of tasks is much more effective than others. Therefore, our algorithm starts by considering all pairs of tasks, in order to assess which ones are good sources for which targets. It also determines which tasks are the best candidates to be *head* of the curriculum, or *tail*, that is, the last task before the final tasks. The method is shown in Algorithm 2.

This first phase, expanded in Algorithm 3, consists in evaluating all curricula of length 2 (Line 4), and sort them (Line 5), with the best curriculum first. At Line 9 and 10 the algorithm assigns a score to each task: the better the length-2 curriculum it belongs to, the lower the score. These scores are returned by the function `EvaluatePairs`.

After this initial phase, Algorithm 2 uses the computed scores to determine the order in which curricula are evaluated. The underlying intuition is the following: the most promising head and tail tasks are tried first, and shorter curricula are evaluated before longer ones. At each round r (Line 3), one more task is added to the set of candidate heads (Line 9) or tails (Line 10) alternatively. For each length up to the maximum length (Line 11), and for all pairs of tasks, one from the head set H, $\bar{h}$, and one from the tail set

Algorithm 2 HTS-CR

Input: $\mathscr{T}$, $\mathscr{F}$, and L
Output: curriculum c^* and its value v^*

1: $I \leftarrow 1, J \leftarrow 1$
2: $\langle heads, tails, V \rangle \leftarrow$ `EvaluatePairs`$(\mathscr{T}, \mathscr{F})$
3: **for** r from 1 to $2(|\mathscr{T}| - 1)$ **do**
4: **if** $(r \bmod 2) = 1$ **then**
5: $I = I + 1$
6: **else**
7: $J = J + 1$
8: **end if**
9: $H \leftarrow$ `Best`$(heads, I)$ // I tasks with lowest score
10: $T \leftarrow$ `Best`$(tails, J)$ // J tasks with lowest score
11: **for** l from 3 to L **do**
12: **for** $\bar{h} \in H$ and $\bar{t} \in T$ s.t. $\bar{h} \neq \bar{t}$ **do**
13: $\mathscr{B} \leftarrow H \cup T \setminus \{\bar{h}, \bar{t}\}$
14: $B \leftarrow$ `Permutations`$(\mathscr{B}, l-2)$
15: **for** $b \in B$ **do**
16: $c \leftarrow \langle \bar{h}, b, \bar{t} \rangle$
17: **if** $c \notin V$ **then**
18: $v \leftarrow \mathscr{P}(c, \mathscr{F})$
19: $V \leftarrow P \cup \{\langle c, v \rangle\}$
20: **end if**
21: **end for**
22: **end for**
23: **end for**
24: **end for**
25: **return** $\langle c^*, v^* \rangle \in V$ s.t. $\forall \langle c, v \rangle \in P, v \leq v^*$

T, $\bar{t}$, the algorithm generates all the permutations of the remaining tasks in $H \cup T$ (Line 14). It then appends $\bar{h}$ at the beginning, and $\bar{t}$ at the end, creating a full curriculum, which, if not considered before (Line 17) is evaluated by running the corresponding simulation, estimating the cumulative return (Line 18).

HTS-CR has no parameters other than the maximum length of the curricula to be searched, which can be exchanged for a different stopping criterion, such as a maximum budget of curricula evaluations. We intentionally left out all curricula of length 1, since our heuristic would not have any meaningful order among them. They could be evaluated in a preliminary phase in any order. Moreover length 1 curricula never achieve the highest cumulative return performance in our experiments (see Appendix A for more information at this regard). We will show experimentally in the next chapter that, after the initial cost of evaluating all curricula of length 2, the solutions found are quickly close to optimal. Finally, HTS-CR requires direct access to the final tasks in order to assess the performance

Algorithm 3 EvaluatePairs

Input: $\mathscr{T}$ and $\mathscr{F}$
Output: $\langle heads, tails \rangle$

$D, V \leftarrow \emptyset$
$heads = tails = [\{m_1, 0\}, \ldots, \{m_{|\mathscr{T}|}, 0\}]$ // dictionary // from tasks to integers
$D \leftarrow \texttt{AllPairs}(\mathscr{T})$
$V \leftarrow \{\langle d, v \rangle | d \in D \wedge v = \mathscr{P}(d, \mathscr{F})\}$ // evaluate all pairs
$V \leftarrow \texttt{Sort}(V)$ // sort wrt cumulative return, best first.
for i from 1 to $|V|$ **do**
    $\langle d, v \rangle \leftarrow V_i$ // i-th best curriculum in V
    $\langle h, t \rangle \leftarrow d$ // head and tail of d
    $heads[h] \leftarrow heads[h] + i$
    $tails[t] \leftarrow tails[t] + i$
end for
return $\langle heads, tails, V \rangle$

of a curriculum but this assumption is relaxed in the next section with the introduction of a novel methodology.

5.2.2 Methodology

Our heuristic algorithm, as other solutions in literature to task sequencing problems, requires sampling over the target task whose learning phase we are aiming to optimize. For real-world applications this is not always possible as it could be extremely time consuming or dangerous to the extent that it is more advantageous to use those samples directly on the target task. For cumulative return optimization we here demonstrate how we can direct our heuristic search towards optimal curricula for a real target task by only interacting with different simulations of it.

We envision cumulative return maximization with curriculum learning to be best applied in situations where a manufacturer needs to provide its product to different costumers with different needs. In such a scenario we would normally aim at developing a general solution able to perform well in all the different given cases, nonetheless the resultant performance would not be optimal in any of them. If the set of problem instances is very large and characterized by high variance, a general solution would be likely to perform poorly at least in some of them. On the other hand, it is impractical to develop a specific solution for each situation.

Alternatively we could approach this problem from a machine learning perspective, for instance applying state-of-the-art reinforcement learning algorithms to learn online the best solution for each given case. It is clear though, that learning such a solution can be slow and implies extensive exploration which is undesirable for real-world environments. This

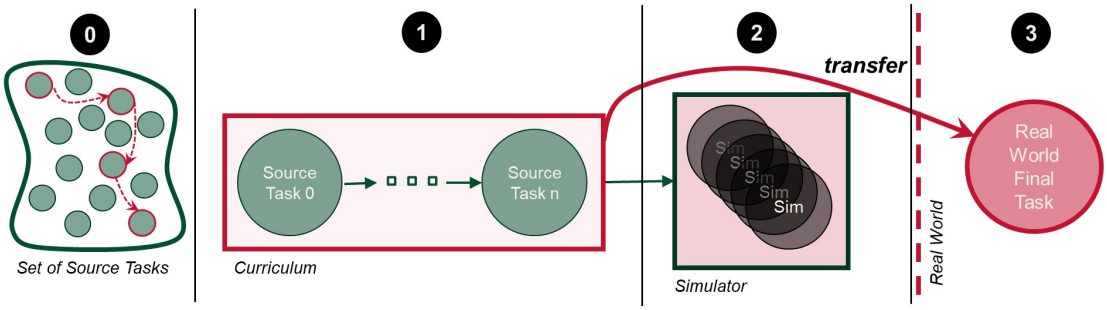


Fig. 5.1 The methodology proposed for solving real-world tasks with curriculum learning for cumulative return maximization. 0) Search a curriculum; 1) Learn the curriculum; 2) Evaluate the curriculum; 3) Transfer the knowledge into the real-world domain.

problem is not new to the field and we have previously discussed how transfer learning introduces a partial solution to this type of scenario by exploiting experience coming from intermediate tasks that can be simulated or real-world.

Figure 5.1 shows the methodology we propose to overcome the problem just described by using curriculum learning. The main idea relies on the fact that since more than one simulation for the final task is available, the larger the number of simulations the better will be the curriculum performance on the real-world task. This is because, with a big enough set of simulations, we would have a precise esteem of the expected performance of the agent learning the final task with the curriculum, therefore the chosen curriculum would be more appropriate for solving the task in the real world. Our method starts by employing any sequencing algorithm in order to propose which curriculum needs to be tested next (step 0). Once we have a candidate curriculum, the RL agent learns through it, transferring the obtained knowledge one task at a time (step 1). At this point we use the knowledge gathered at the end of the curriculum to initialize the learning of each of the simulated final tasks and test in this way the quality of the given curriculum (step 2). If the performance of the curriculum in the simulations is high enough, we can transfer its knowledge to lead exploration in the real-world (step 3). We expect the tested performance to be close to the one we will actually observe in the real-world final task as long as the set of simulations is large enough.

5.3 Summary

In this chapter we introduced a number of metaheuristic algorithms for solving combinatorial optimization problems. We then described their adaptation to the task sequencing problem in curriculum learning in order to apply them to any of the performance metrics discussed in Chapter 4. The selected methods span a variety of different approaches to combinatorial optimization and in the next chapter we aim at evaluating them so as

to identify the salient characteristics of general sequencing algorithms for curriculum learning.

We then discussed how it is possible to design promising heuristic algorithms able to rapidly search optimal solutions in the space of all the possible curricula by following curriculum learning expert knowledge and by focusing on only one specific performance metric.

Finally we introduced our particular heuristic search HTS-CR for cumulative return maximization and explained its design. Moreover we explained the proposed methodology for applying heuristic search methods for cumulative return maximization to real-world domains.

In the next chapter we will discuss the performance of HTS-CR and compare them to those of the general metaheuristic methods.

Chapter 6

Experimental Evaluation

In the previous chapters we proposed a combinatorial optimization framework for curriculum learning and discussed which algorithms are best suited for approaching the task sequencing problem in this context. We studied different performance metrics and task scenarios, explaining the reasons behind the choice of focusing on cumulative return maximization for agents in critical tasks.

In this chapter we start by performing a comparison of all the proposed metaheuristic algorithms in order to understand which of these approaches works best for curriculum learning. Later we examine their performance against the one of the heuristic algorithms we developed for the task sequencing problem for cumulative return maximization. Finally we introduce a critical task domain showing the potential effectiveness of our approach for real-world applications.

6.1 Meaheuristic Algorithms Performance

In order to compare and analyze the performance of our implementation of the four metaheuristic algorithms for combinatorial optimization introduced in the previous chapter, we test them over the dataset described in Section 4.6.1. We want to understand in this way, which features of popular metaheuristic methods are the strongest to solve the general curriculum learning optimization problem.

It is important to notice how, within the set of source tasks, we intentionally created both tasks that provide positive and negative transfer towards the final task. In this way, for each experiment in the dataset there are always extremely strong sequences of tasks but also curricula whose impact is negative on the final task performance. This allows testing the ability of the various sequencing algorithms to choose the most appropriate curricula while avoiding counterproductive ones.

	n = 12; L = 4; tot = 13345						
GW	C_0	Greedy	GA	Tabu	ACO	Beam	Opt
CR	- 0.64 -	24 0.74 -	378.58 0.77 [0.76:0.77]	**364.54** **0.79** **[0.79:0.79]**	378 0.76 [0.76:0.76]	373 0.78 -	- 0.81 -
JS	- -2283.8 -	34 -860.03 -	378.7 -1223.24 [-1309.87:-1136.61]	380.94 -842.68 [-900.18:-785.17]	378 -779.82 [-795.44:-764.21]	**373** **-738.29** -	- -601.74 -
TTT	- -1351.6 -	24 -315.6 -	380.3 -545.73 [-388.93:-702.53]	381.86 -315.6 [-315.6:-315.6]	39 -466.12 [-345.80:-586.44]	**373** **-315.6** -	- -315.6 -
MR	- -50 -	24 -26.6 -	377.16 -15.5 [-18.55:-12.45]	463.28 -5.716 [-9.87:1.56]	378 -5.33 [-6.90:-3.76]	**373** **-3** -	- 19.9 -

Table 6.1 Results of all the algorithms on the first experiment in the GridWorld domain. Green cells enhance the result of the best algorithm for a specific performance and experiment.

	n = 7; L = 7; tot = 13700						
GW	C_0	Greedy	GA	Tabu	ACO	Beam	Opt
CR	- 0.51 -	16 0.57 -	155.84 0.59 [0.58:0.61]	**165.4** **0.62** **[0.62:0.63]**	168 0.61 [0.6:0.62]	155 0.61 -	- 0.72 -
JS	- -2624 -	16 -827.68 -	156.22 -896.17 [-966.24:-826.11]	169.9 -855.99 [-920.08:-791.97]	**154** **-725.65** **[-759.65:-691.65]**	155 -773.71 -	- -360.9 -
TTT	- -2788.1 -	12 -1643.7 -	156.52 -2300.41 [-1967.20:-2633.63]	166.3 -19880.42 [-17369.97:-22390.87]	168 -1716.13 [-1680.85:-1751.42]	**155** **-1643.87** -	- -1643.87 -
MR	- -50 -	19 21.1 -	155.98 6.96 [0.95:12.98]	**165.92** **26.78** **[22.84:30.71]**	168 18.36 [14.52:22.21]	155 21.1 -	- 68.5 -

Table 6.2 Results of all the algorithms on the second experiment in the GridWorld domain. Green cells enhance the result of the best algorithm for a specific performance and experiment.

	n = 9; L = 5; tot = 18730						
BD	C_0	Greedy	GA	Tabu	ACO	Beam	Opt
CR	- 0.19 -	25 0.55 -	266.78 0.55 [0.53:0.57]	286.02 0.55 [0.53:0.57]	245 0.54 [0.51:0.57]	**244** **0.6** -	- 0.65 -
JS	- -49.9 -	25 -49.04	267.44 -49.08 [-49.18:-48.98]	315.34 -48.17 [-48.58:-47.77]	245 -48.42 [-48.75:-48.09]	**244** **-47.32** -	- -47.77 -
TTT	- -2981.7 -	18 -2749.3	267.92 -2879.87 [-2825.41:-2934.32]	262.86 -3086.52 [-2998.12:-3174.92]	245 -2793.00 [-2768.23:-2817.78]	**244** **-2749.3** -	- -2749.3 -
MR	- -10 -	31 -10 -	266.66 -10 [-10:-10]	333.76 -10 [-10:-10]	245 -10 [-10:-10]	**244** **-10** -	- -10 -

Table 6.3 Results of all the algorithms on the first experiment in the BlockDude domain. Green cells enhance the result of the best algorithm for a specific performance and experiment.

	n = 18; L = 3; tot = 5221						
BD	C_0	Greedy	GA	Tabu	ACO	Beam	Opt
CR	- 0.2 -	52 0.48 -	622.66 0.5 [0.48:0.51]	750.5 0.55 [0.55:0.56]	663 0.47 [0.46:0.49]	**613** **0.57** -	 0.58 -
JS	- -50 -	52 -49.4 -	623.66 -49.66 [-49.70:-49.62]	**695.16** **-49.36** **[-49.41: -49.32]**	663 -49.76 [-49.79:-49.73]	613 -49.38 -	 -49 -
TTT	- -4986 -	36 -3828 -	625.28 -3868.10 [-3852.95:-3883.25]	665.2 -3828 [-3828:-3828]	663 -3834.80 [-3831.12:-3838.47]	**613** **-3828** -	 -3828 -
MR	- -14 -	67 -14 -	621.34 -14 [-14:-14]	752.1 -14 [-14:-14]	663 -14 [-14:-14]	**613** **-14** -	- -14 -

Table 6.4 Results of all the algorithms on the second experiment in the BlockDude domain. Green cells enhance the result of the best algorithm for a specific performance and experiment.

In Table 6.1, 6.2, 6.3 and 6.4 we show the results of the experiments on GridWorld and Block Dude respectively. The header row shows the number of candidate tasks, maximum length, and the total number of curricula of each experiment. Each cell contains the average number of curricula evaluated, the value of the objective function, and the 95% confidence interval of that value, for the corresponding metric. We also included the optimal curriculum (last column of each experiment) and learning with no curriculum (denoted as C_0). The cumulative return metric is hereby normalized in $[0,1]$ where 0 corresponds to the worst possible policy never improving the agent performance during the training, while 1 stands for directly starting with the optimal policy.

Our implementation of Tabu Search, Genetic Algorithms and Ant Colony Optimization are stochastic and non-terminating, so that in the limit they always find the optimal solution. For a fair comparison, and for the cases of practical interest, we interrupted them at the first iteration (for instance, generation in GA) in which they evaluated a number of curricula equal to or greater than Beam Search at its optimal value, since Beam Search is deterministic.

From the tables it is clear how curriculum learning can be employed for optimizing different performance metrics for learning the final task. From our study, Beam Search outperforms all the metaheuristics in almost all the experiments, and when it does not, the solution is often close to the best one. Tabu Search, notably the other trajectory-based algorithm, is better in some cases. However, since Beam Search is deterministic, it may be preferable as there is no variation between runs. It is also important to notice that the performance of Beam Search degrades faster as the maximum length increases, rather than as the number of candidate tasks increases. Beam Search is also not global, therefore it

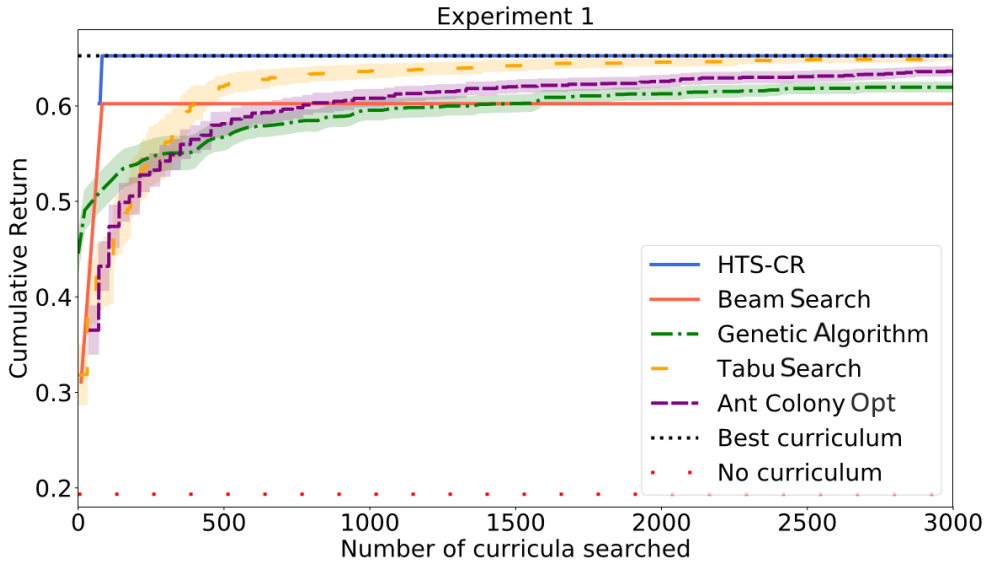
finds solutions only in a small part of the search space that often excludes the globally optimal solution.

We can conclude here that trajectory-based algorithms work better than population-based ones for searching sequences of tasks in curriculum learning. Beam Search is the algorithm achieving the best performance over our dataset and it is preferable both for its stability (being a deterministic solution) and for the ease of implementation. Nonetheless, Tabu Search is a powerful alternative and with a deeper understanding of curriculum learning it can be possible to develop even stronger implementations of this algorithm to leverage its features.

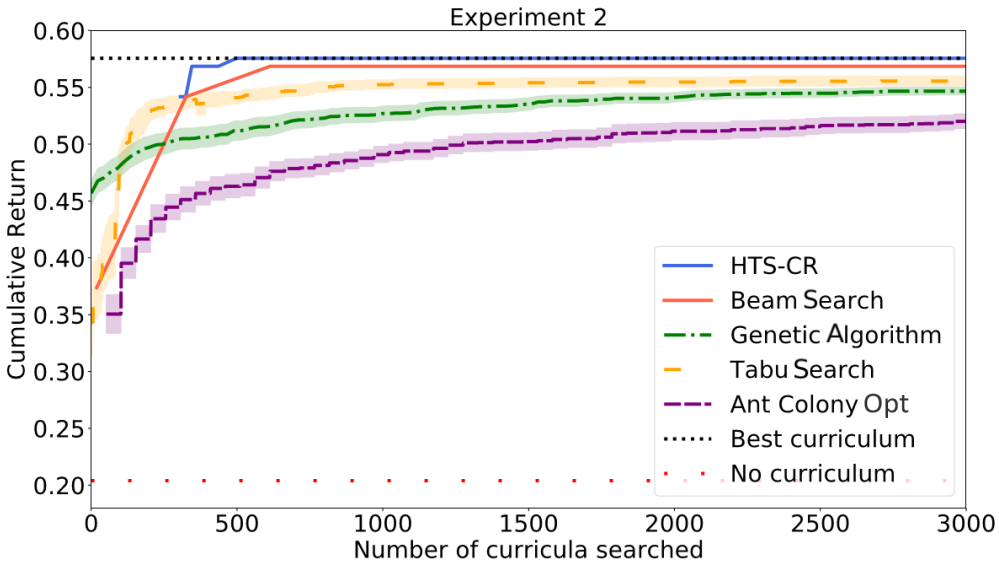
We can also draw some further conclusions about the performance metrics. Time-to-threshold, the objective used the most in the literature, appears to be the easiest to optimize in our experiments. This can be noticed by observing how Greedy Search is always able to find optimal solutions for this very metric. We decided not to indicate this algorithm as the best one for this performance because of the fact that its success is due to the particular shape of the optimization problem, that always sees the best curriculum to be composed by one only intermediate task. Because of this reason Greedy Search always finds the optimal solution during the first curricula evaluations it performs which includes all the curricula of length $l = 1$ (this is also the case for Beam Search). As we previously discussed in Chapter 4, time-to-threshold is best used with sequencing algorithms that optimize the time spent on each task in the curriculum. As we do not perform such actions in these experiments, the formulation of time-to-threshold naturally promotes short curricula (refer to Appendix A for further information). Lastly, BlockDude appears to be too simple for max-return to be a viable objective, since all curricula, including no curriculum, eventually learn the optimal policy. On the contrary, GridWorld is a more complex domain and it is therefore more complicated for any agent to find the optimal policy. For this domain, different curricula result in the discovery of different policies, with a significant improvement over learning from scratch.

6.2 HTS-CR Evaluation

The previous results strengthen the choices behind the design of our heuristic algorithm. On the one hand it is deterministic and simple to implement as Beam Search, on the other hand it is also global similarly to Tabu Search, combining in this way the virtues of these two trajectory based methods. In this section we perform a deeper analysis of the task sequencing problem associated with cumulative return maximization and discuss the overall behaviour of the previously introduced metaheuristic methods and our specifically designed heuristic algorithm.

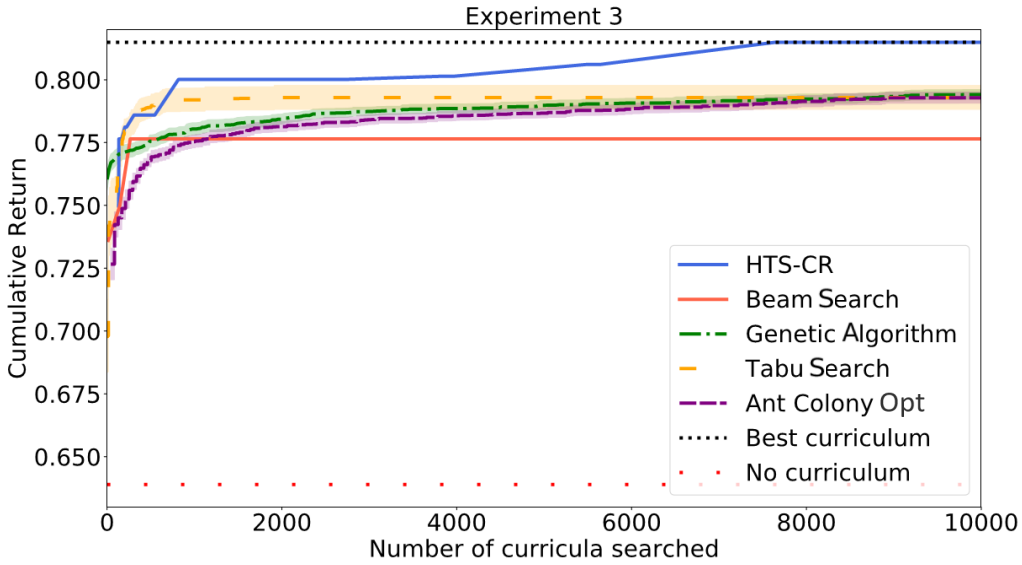


(a) Experiment 1, Block Dude

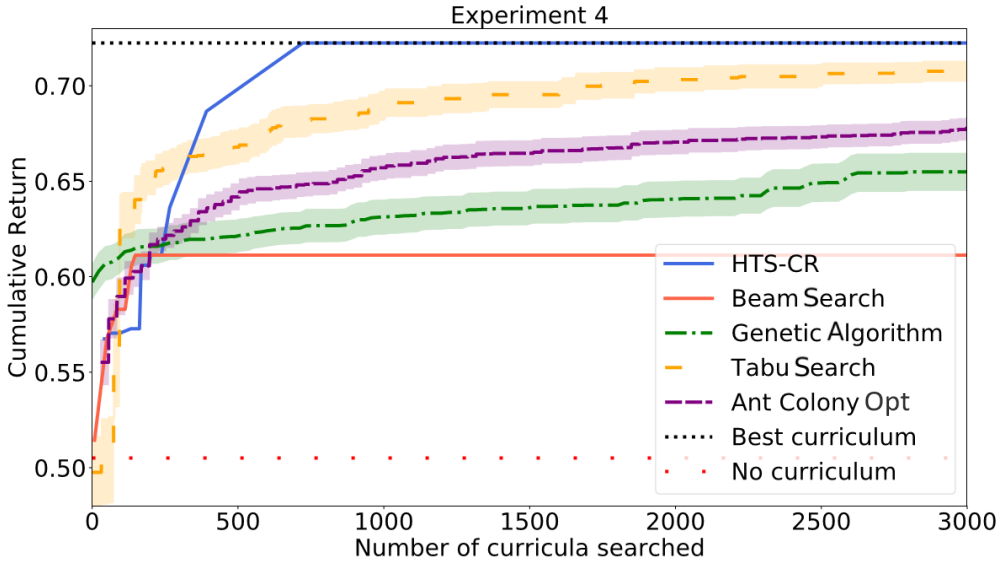


(b) Experiment 2, Block Dude

Fig. 6.1 Comparison of HTS-CR against the combinatorial optimization methaeuristic algorithms on the Block Dude domain. On the x-axis we show the number of curricula evaluated by each algorithm and on the y-axis the correspondent best cumulative return value encountered by each algorithm after the specified number of evaluations.



(a) Experiment 3, Gridworld



(b) Experiment 4, Gridworld

Fig. 6.2 Comparison of HTS-CR against the combinatorial optimization methaeuristic algorithms on the Gridworld domain. On the x-axis we show the number of curricula evaluated by each algorithm and on the y-axis the correspondent best cumulative return value encountered by each algorithm after the specified number of evaluations.

In Figure 6.1 and 6.2 we compare HTS-CR against the metaheuristic methods described in Section 5.1. The four plots, one for each experiment, show the value of the best curriculum over the number of curricula evaluated by each algorithm. Curricula were evaluated by having the agent learn each one multiple times, and averaging the results to estimate the objective in Equation 4.1. The cumulative return was normalized in $[0,1]$ for ease of comparison across the experiments, where 1 is the return accumulated by the optimal policy at every episode. As Tabu Search, Genetic Algorithms and Ant Colony Optimization are stochastic methods, their performance were averaged over 70 runs and plotted showing the 95% confidence interval. In all the experiments HTS-CR has an initial offset of $n(n-1)$ evaluations spent to consider all the possible pairs, whereas all the metaheuristics immediately start finding possible solutions. Nevertheless, HTS-CR quickly outperformed all the other algorithms, and always found the globally optimal curriculum the fastest, showing the benefit of the initial pair evaluations.

Both Beam Search and Tabu Search show better performance than the population-based algorithms especially during the initial evaluations where they rapidly discover curricula of always higher performance. With these plots it is also clear that Beam Search often gets stuck in a sub-optimal solution and therefore its performance worsens with the growing of the number of iterations, while all the other methods keep improving the quality of the solution they found.

6.3 Real World Domain

We discussed different possible applications of curriculum learning, and showed how it is possible to employ curricula for the optimization of different features of the training of an agent learning the final and most important task. Among the three task scenarios we introduced in Section 4.2, we decided to focus on critical tasks since in our dataset, the metric associated with this case, cumulative return, results challenging and with promising practical results.

Problems in the critical task scenario are usually real-world and in the most of the cases a simulation of the target environment is available. This allows for the creation and design of curricula in simulation for then applying them to the real-world problem. In this section we describe the new domain we used for performing real-world experiments and show a practical application of curriculum learning in such domain.

6.3.1 MGEnv Domain

MGEnv is a simulated micro-grid domain modeled out of real data from the PecanStreet Inc. database. The domain includes historical data about hourly energy consumption and

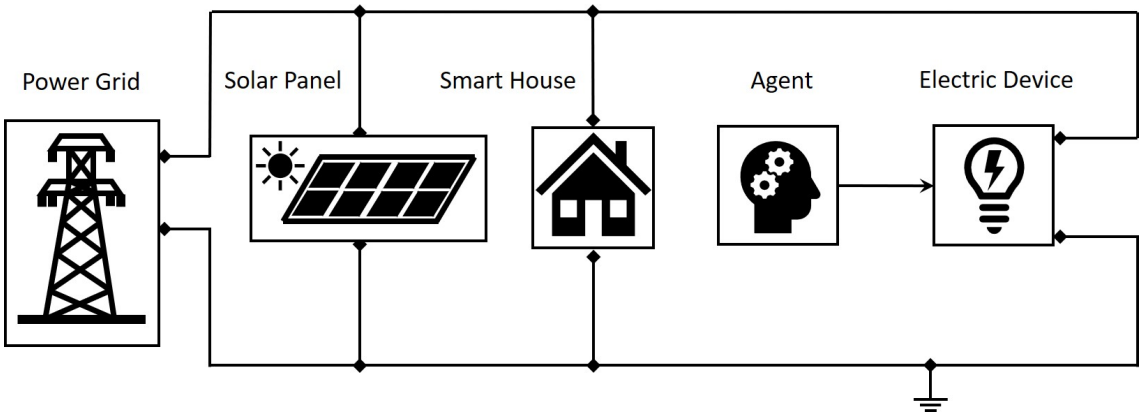


Fig. 6.3 A graphical representation of our micro-grid domain, MGEnv

solar energy generation of different buildings from January 2016 to December 2018. A task in this domain is defined by the combination of three elements: the model of the electrical device to optimize; the user's monthly schedule, specifying the days in which the user wishes to run the device; the month of simulation, with the energy generation and base consumption of the given building. The device we used behaves as a time-shifting load: once started it runs for several time steps and cannot be interrupted before the end of its routine. This is the most challenging type of device present in the database. The goal is to find the best time of the day to run the given device, optimizing direct use of the generated energy, while respecting the user's device schedule. The agent receives a reward of 30 when the device energy consumption is fully covered by the energy generated by the building, -10 when energy not generated by the building is used and -200 if the device is not run accordingly to the user schedule.

6.3.2 Experiments

We developed a DeepRL agent for the MGEnv domain with an Actor-Critic architecture, using Proximal Policy Optimization (PPO) [71], and Progressive Neural Networks [66], for value function transfer. Since this is a curriculum learning implementation of PPO, before the learning agent starts training on a new task in the curriculum, its replay buffer is emptied from all the old experience samples.

In Appendix A we provide a detailed description of all the tasks designed for the experiments in the MGEnv domain in order to ease their reproducibility, whereas in this section we only discuss their high level design for then focusing on the results. All the tasks in this domain are for the control of the same electric device. We created 10 final tasks using the same month and user schedule, but different buildings. We divided them into a training (validation) and test set of 5 tasks each. Without curriculum learning, a

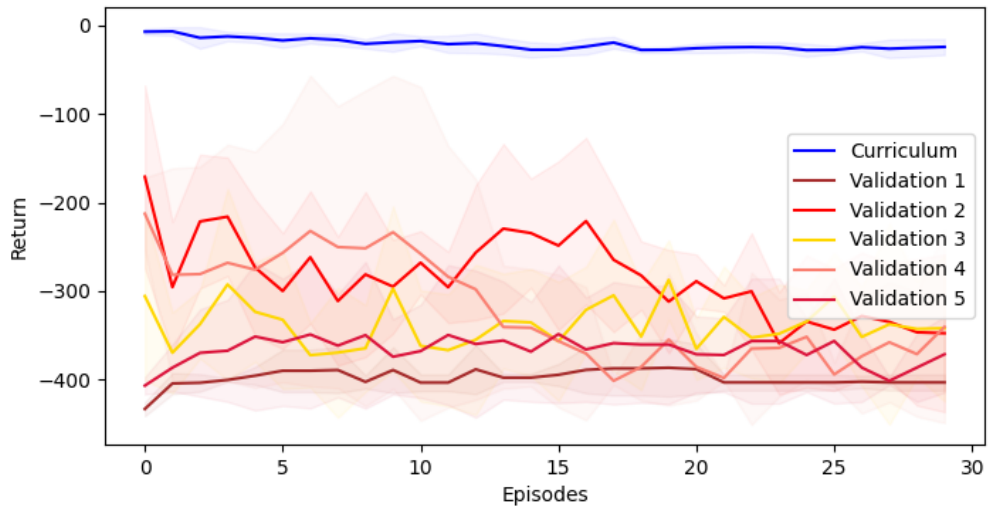


Fig. 6.4 Average return over the test tasks in the MGEnv domain. The blue line corresponds to initializing learning in the final tasks with the curriculum, while all the others represent the performance of initializing it with a single training (validation) task

natural approach would be to learn the optimal policy for one of the source tasks in the training set, and transfer the optimal value function to a target task in the test set.

We created $n = 5$ intermediate tasks, by selecting a combination of schedule, month and building from a set of 3 schedules, 5 months and 3 buildings. Some of these tasks are easier to learn than others, providing the basis for the curriculum. We optimized the curriculum with HTS-CR over the training set with a maximum length $L = 4$, repeating each evaluation 5 times to estimate the cumulative return. The best curriculum was found after 37 curriculum evaluations. We then took the value function after the tail of the curriculum, before the final tasks, and transferred it to each of the 5 tasks in the test set. We evaluated the performance of the agent initialized from the curriculum, and compared it to the behaviour of the agent initialized from single-task transfer.

Figure 6.4 shows the results of this experiment.[1] The curriculum was generated using all the training tasks together, and evaluated over each one of the 5 test tasks separately. Therefore, the plotted results are the average over 5 runs with 95% confidence intervals. Single-task transfer, on the other hand, was trained on each one of the training tasks and evaluated on each test task separately. Consequently we have 5 different curves for single-task transfer, one for each training task used for initializing the agent learning in the final task. Each learning curve is the average over 5 runs with 95% confidence

[1]The results showed in figure 6.4 differ from those originally included in our publication [28]. Here, indeed, we represent each single-task transfer curve separately, but, most importantly, the duration of a learning episode corresponds to one day instead of an entire month. We believe that this different design better represents a real-world scenario.

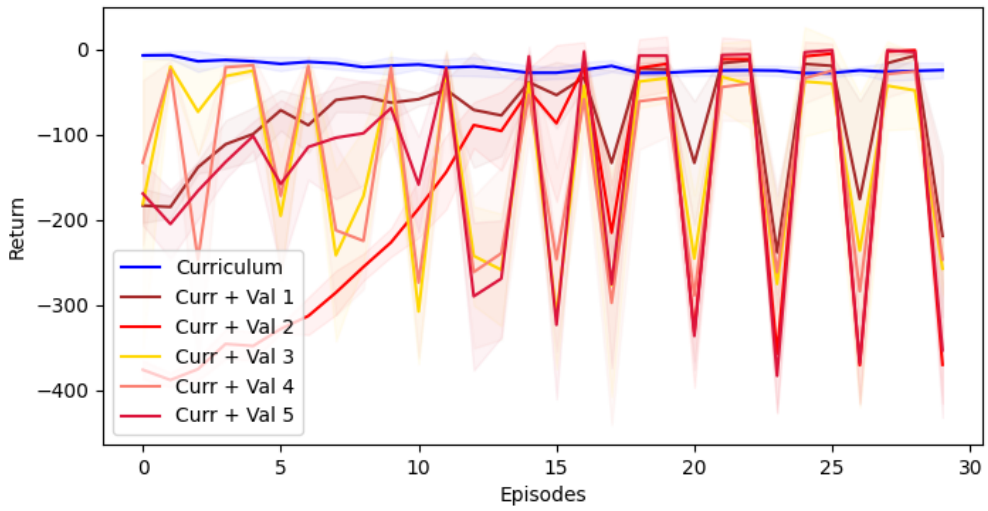


Fig. 6.5 Average return over the test tasks in the MGEnv domain. The blue line corresponds to initializing learning in the final tasks with the curriculum, while the others represent the agent learning the final tasks with the discovered curriculum modified by adding one of the training tasks to its tail

intervals. We show that, on average, initializing learning from the curriculum achieves a higher cumulative return than initializing from the optimal value function of any of the training tasks. The results show that a curriculum generated from simulated tasks can indeed generalize to similar real tasks (recall that all the data in these simulations is from real buildings) and provide a significant improvement of the behaviour during learning. Furthermore, we previously discussed how cumulative return and jumpstart are the two best suited performance metrics for tasks belonging to the critical task scenario, although optimizing one of them does not necessarily mean improving both the performance values. Nevertheless, it is possible to notice in figure how, for this domain, cumulative return maximization also brings a clear advantage on jumpstart.

Instead of using the training tasks only for searching the curriculum to initialize the agent training in the final tasks, it is also possible to include them as last task (tail) of the curriculum found in this way. Potentially, this choice could enhance the cumulative return improvement brought by the curriculum even further. Figure 6.5 shows the different performance of the agent learning the final tasks with the curriculum modified by adding one of the training tasks to its tail. All the curricula generated in this way present an extremely fluctuating behaviour towards the end of the learning phase, therefore the original curriculum still results to be the best initialization for the agent learning in the final task. This phenomenon is due to the fact that training and final tasks are specifically designed to be similar one another. Including training tasks in the curriculum generates a

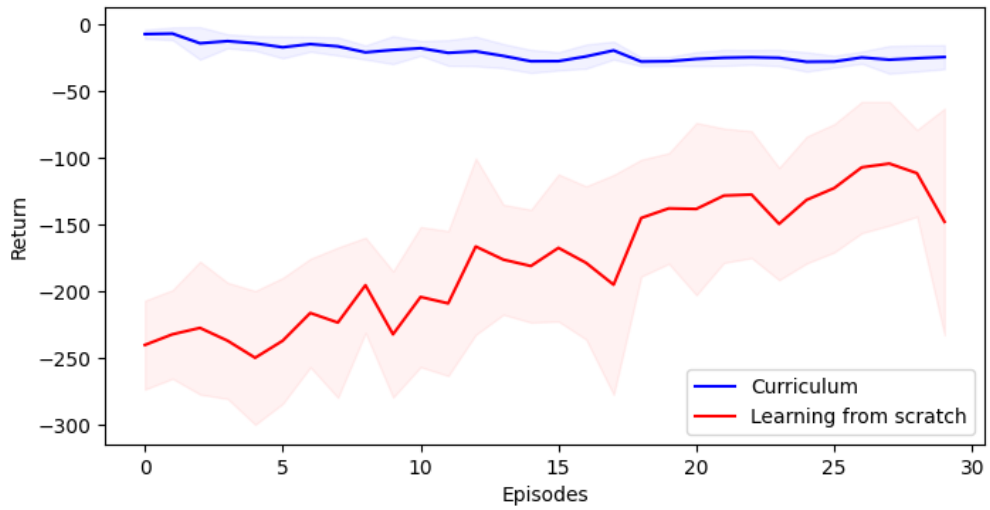


Fig. 6.6 Average return over the test tasks in the MGEnv domain. The blue line corresponds to initializing learning in the final tasks with the curriculum while the red line represents the performance of an agent learning the final task from scratch

value function that is overfit to the selected training task, hence it is harder for the agent to update it during the learning phase in the final tasks. This result proves the effectiveness of the methodology proposed in Chapter 5.

For completeness, in Figure 6.6 we also show the results of learning the final tasks with the discovered curriculum against not using any available previous knowledge to initialize the agent (learning from scratch). As expected, the knowledge coming from the curriculum gives a strong advantage to the learning agent compared to training in the final tasks without exploiting any previous experience.

Finally, we also discuss the performance of curricula composed by only one intermediate task. These curricula are not explored by HTS-CR as in our experiments the curricula achieving the best cumulative return performance are always composed by multiple tasks. Figure 6.7 shows how HTS-CR finds a curriculum that substantially outperforms all the curricula composed by only one intermediate task.

6.4 Summary

In this chapter we evaluated the previously introduced metaheuristic algorithms for combinatorial optimization on the task sequencing problem in curriculum learning. We observed how in general, for any of the CL performance metrics of Chapter 4, trajectory-based methods like Beam Search or Tabu Search are the most efficient in finding beneficial curricula.

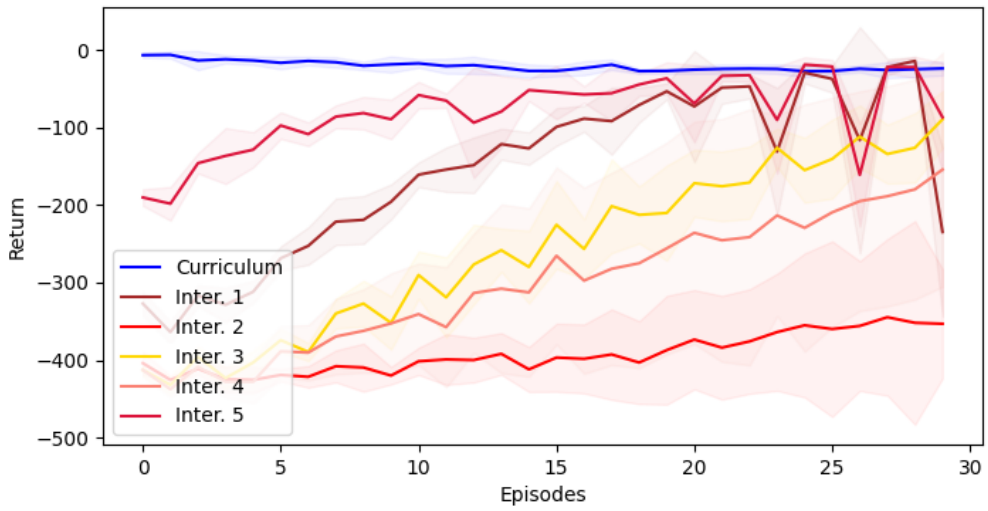


Fig. 6.7 Average return over the test tasks in the MGEnv domain. Initializing the learning agent with a curriculum (in blue) is compared against initializing it with only one intermediate task from the available ones

We then compared our heuristic algorithm HTS-CR against all the metaheuristic methods for cumulative return maximization with curriculum learning. In our experiments, HTS-CR rapidly moves the focus of the search towards promising curricula converging to the globally optimal solution the fastest.

Finally we tested HTS-CR on a real-world domain exploiting the methodology described in Chapter 5 searching for a curriculum offline by using the available simulations of the given final task. We showed how curriculum learning can lead to safe and robust exploration policies when online reinforcement learning is needed.

Chapter 7

Conclusions

7.1 Contribution

In this work we introduced a new framework for curriculum learning in reinforcement learning. We identified three different task scenarios for novel uses of curriculum learning and discussed which performance metrics are best suited for each of them. In *complex tasks* we are interested at discovering the highest valued policy at any time during training in order to use it for solving the problem defined in the final task. *Time sensitive tasks* are the most studied in literature, and in this setting, in order to find strong curricula, task sequencing algorithms need to optimize the time spent over each source task composing the curriculum. We focus on *critical tasks* where online reinforcement learning is necessary in order to optimize the solution provided by the curriculum to the actual final task to solve. In this context a curriculum is used to shape the exploration of the agent in such a way that it minimizes the number of expensive actions it takes during the learning phase.

We adapted several metaheuristic methods for combinatorial optimization to the task sequencing problem in curriculum learning in order to search for curricula given any performance metric. We also developed our heuristic search algorithms HTS-CR to find curricula for the critical task scenario. HTS-CR searches for sequences of tasks maximizing the cumulative return performance over the final task as this is the metric selected for this type of task scenario.

We performed an evaluation of all the metaheuristic techniques in order to understand which features of these algorithms are the strongest at searching curricula optimizing any of the given performance metrics. In our experiments, trajectory-based methods outperform population-based methods finding better curricula in a reasonable number of algorithm iterations. We also compared the performance of HTS-CR against all the metaheuristic methods solving the task sequencing problem for cumulative return maximization and

observed how our heuristic search converges to optimal solutions faster than any other algorithm under analysis.

Finally, we proposed a methodology for applying task sequencing algorithms for cumulative return maximization to real-world problems. In our experimental domain, a microgrid environment, we noticed how employing curriculum learning can have a great impact on the agent performance over the final task, which, in our case, starts the training with a high valued policy and avoids expensive and potentially unsafe actions throughout the whole learning phase.

7.2 Limitations

We started our study performing an analysis of the proposed performance metrics for curriculum learning. For this purpose we created a dataset composed of four experiments in two different domains. These experiments represent only some aspects of the overall complexity of the problem of task sequencing in curriculum learning, and therefore it is not possible to draw general conclusions over the shape of the optimization problem associated with each performance metric. A more diverse experimental domain set would be of great importance for this purpose and could help designing extremely dynamic curriculum learning algorithms.

We discussed how the effectiveness of a curriculum is directly correlated with the function approximator used for learning each task and the transfer learning method chosen for passing experience from one task to another. In this work we used two types of function approximators and two transfer learning algorithms for value function transfer. We combined them in two different ways and proved how the performance of HTS-CR are consistent over them. Nevertheless, there might be some other function approximator and transfer learning algorithm that together worsen the efficiency of the sequencing algorithms studied in this work.

Lastly, for extremely complex *critical* tasks, creating a set of source tasks potentially beneficial for a curriculum and having an accurate simulation of the final task can be difficult. In such a case a good designer choice can be to increase the dimension of the source set and the number of simulations in the test set. In this way the sequencing algorithm would take a longer time before finding any beneficial curriculum which can still be impractical for some real-world applications.

7.3 Future Work

Curriculum learning in reinforcement learning is still at its early stage of development and there are many aspects of this field that need to be investigated before being able to employ it in the design of learning agents for future technologies. In this work we analyzed the task sequencing problem in curriculum learning trying to understand what characteristics of this problem show the biggest potential for improvement in order to focus our research on one of them.

As we previously mentioned, a global understanding of the salient features characterizing a positive curriculum for learning a final task would be of crucial importance for this field. This means creating a large dataset of multiple experiments on many different domains evaluating each curriculum at varying of the type of function approximator used for learning each task in the curriculum and the transfer learning algorithm for transferring knowledge along the sequence of tasks.

In this study a curriculum is strictly defined as a sequence of tasks with no repetitions. In Chapter 3 we discussed how there exist other definitions of curricula with fewer restrictions on how to combine the intermediate tasks to generate a curriculum. For instance, it is particularly important to consider algorithms defining curricula as graphs. In this way transfer of knowledge could happen concurrently from different source tasks letting in this way possible to design extremely more complex curricula.

In order to be able to fully exploit the knowledge coming from each source task in a curriculum, CL algorithms would benefit from research focused on function approximators and transfer learning algorithms specifically designed for curriculum learning applications. In this way, each intermediate task could have a great impact on the final task. For this purpose, continuous curriculum learning [7] is a promising research direction because of its formulation which allows to consider in only one algorithm both the time the agent needs to spend over each task it encounters, and a method for sequencing source tasks.

Lastly, in future works, it will be important to evaluate curriculum learning algorithms, especially in the context of cumulative return maximization, on real-world domains such as robotics where the impact of this field can help overcome challenges such as bridging the reality gap.

References

[1] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058.

[2] Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning*, 23(2-3):279–303.

[3] Audibert, J.-Y., Munos, R., and Szepesvári, C. (2009). Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902.

[4] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.

[5] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2020). Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations (ICLR)*.

[6] Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73.

[7] Bassich, A., Foglino, F., Leonetti, M., and Kudenko, D. (2020). Curriculum learning with a progression function. *arXiv preprint arXiv:2008.00511*.

[8] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

[9] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

[10] Brafman, R. I. and Tennenholtz, M. (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231.

[11] Brown, R. (1970). Derivational complexity and order of acquisition in child speech. *Cognition and the development of language*.

[12] Brys, T., Harutyunyan, A., Taylor, M. E., and Nowé, A. (2015). Policy transfer using reward shaping. In *The 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[13] Clegg, A., Yu, W., Erickson, Z., Tan, J., Liu, C. K., and Turk, G. (2017). Learning to navigate cloth using haptics. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2799–2805. IEEE.

[14] Connell, J. H. and Mahadevan, S. (1993). Rapid task learning for real robots. In *Robot Learning*, pages 105–139. Springer.

[15] Da Silva, F. L. and Costa, A. H. R. (2018). Object-oriented curriculum generation for reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

[16] Domingo, C. (1999). Faster near-optimal reinforcement learning: Adding adaptiveness to the e 3 algorithm. In *International Conference on Algorithmic Learning Theory*, pages 241–251. Springer.

[17] Dorigo, M., Maniezzo, V., and Colorni, A. (1991). The ant system: An autocatalytic optimizing process.

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

[19] Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., and Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160.

[20] Even-Dar, E., Mannor, S., and Mansour, Y. (2002). Pac bounds for multi-armed bandit and markov decision processes. In *International Conference on Computational Learning Theory*, pages 255–270. Springer.

[21] Fachantidis, A., Partalas, I., Tsoumakas, G., and Vlahavas, I. (2013). Transferring task models in reinforcement learning agents. *Neurocomputing*, 107:23–32.

[22] Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. (2019). Curriculum-guided hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 12602–12613.

[23] Fernández, F., García, J., and Veloso, M. (2010). Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871.

[24] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

[25] Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning (ICML)*.

[26] Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning*, pages 482–495.

[27] Foglino, F., Christakou, C. C., and Leonetti, M. (2019a). An optimization framework for task sequencing in curriculum learning. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 207–214. IEEE.

[28] Foglino, F., Coletto Christakou, C., Luna Gutierrez, R., and Leonetti, M. (2019b). Curriculum learning for cumulative return maximization. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI.

[29] Glover, F. (1989). Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206.

[30] Glover, F. (1990). Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32.

[31] Goldberg, D. E. (1989). Genetic algorithms in search. *Optimization, and Machine-Learning*.

[32] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[33] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[34] Hosu, I.-A. and Rebedea, T. (2016). Playing atari games with deep reinforcement learning and human checkpoint replay. *Proceedings of the Workshops on Evaluating General-Purpose AI (EGPAI)*.

[35] Jain, V. and Tulabandhula, T. (2017). Faster reinforcement learning using active simulators. *Proceeding of NIPS'17 Workshop: Teaching Machines, Robots, and Humans*.

[36] Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600.

[37] Kakade, S., Kearns, M. J., and Langford, J. (2003). Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 306–312.

[38] Karpathy, A. and Van De Panne, M. (2012). Curriculum learning for motor skills. In *Canadian Conference on Artificial Intelligence*, pages 325–330. Springer.

[39] Katayama, M., Inoue, S., and Kawato, M. (1998). A strategy of motor learning using adjustable parameters for arm movement. In *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol. 20 Biomedical Engineering Towards the Year 2000 and Beyond (Cat. No. 98CH36286)*, volume 5, pages 2370–2373. IEEE.

[40] Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232.

[41] Khan, F., Mutlu, B., and Zhu, J. (2011). How do humans teach: On curriculum learning and teaching dimension. In *Advances in neural information processing systems*, pages 1449–1457.

[42] Kim, T.-H. and Choi, J. (2018). Screenernet: Learning self-paced curriculum for deep neural networks. *arXiv preprint arXiv:1801.00904*.

[43] Lazaric, A. (2012). Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer.

[44] Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

[45] Lowerre, B. T. (1976). *The HARPY speech recognition system*. PhD thesis.

[46] MacAlpine, P. and Stone, P. (2018). Overlapping layered learning. *Artificial Intelligence*, 254:21–43.

[47] Mann, T. A. and Choe, Y. (2012). Directed exploration in reinforcement learning with transferred knowledge. In *European Workshop on Reinforcement Learning*, pages 59–76.

[48] Mannor, S. and Tsitsiklis, J. N. (2004). The sample complexity of exploration in the multi-armed bandit problem. *Journal of Machine Learning Research*, 5(Jun):623–648.

[49] Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. (2019). Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*.

[50] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[51] Mnih, V., Szepesvári, C., and Audibert, J.-Y. (2008). Empirical bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*, pages 672–679.

[52] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*.

[53] Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2016). Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574.

[54] Narvekar, S., Sinapov, J., and Stone, P. (2017). Autonomous task sequencing for customized curriculum design in reinforcement learning. In *(IJCAI), The 2017 International Joint Conference on Artificial Intelligence*.

[55] Narvekar, S. and Stone, P. (2019). Learning curriculum policies for reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 25–33. International Foundation for Autonomous Agents and Multiagent Systems.

[56] Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287.

[57] Ortner, R. (2008). Online regret bounds for markov decision processes with deterministic transitions. In *International Conference on Algorithmic Learning Theory*, pages 123–137. Springer.

[58] Ow, P. S. and Morton, T. E. (1988). Filtered beam search in scheduling. *The International Journal Of Production Research*, 26(1):35–62.

[59] Pavlov, I. P. (1927). Conditioned reflexes. *London: Oxford University Press*.

[60] Peng, B., MacGlashan, J., Loftin, R., Littman, M. L., Roberts, D. L., and Taylor, M. E. (2018). Curriculum design for machine learners in sequential decision tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(4):268–277.

[61] Pinker, S. (1989). Learnability and cognition: The acquisition of argument structure.

[62] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org.

[63] Ren, Z., Dong, D., Li, H., and Chen, C. (2018). Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning. *IEEE transactions on neural networks and learning systems*, 29(6):2216–2226.

[64] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. pages 4344–4353.

[65] Rohde, D. L. and Plaut, D. C. (1999). Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition*, 72(1):67–109.

[66] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[67] Ruvolo, P. and Eaton, E. (2013). Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pages 507–515.

[68] Sanger, T. D. (1994). Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE transactions on Robotics and Automation*, 10(3):323–333.

[69] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. *International Conference on Learning Representations (ICLR)*.

[70] Schmidhuber, J. (2013). Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313.

[71] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[72] Shao, K., Zhu, Y., and Zhao, D. (2018a). Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 3(1):73–84.

[73] Shao, K., Zhu, Y., and Zhao, D. (2018b). Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, pages 1–12.

[74] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

[75] Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, 13(3):94.

[76] Soni, V. and Singh, S. (2006). Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, volume 6, pages 494–499.

[77] Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Evolving neural network agents in the nero video game. *Proceedings of the IEEE*, pages 182–189.

[78] Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888.

[79] Strehl, A. L. and Littman, M. L. (2005). A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd international conference on Machine learning*, pages 856–863.

[80] Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. (2018). Intrinsic motivation and automatic curricula via asymmetric self-play. In *6th International Conference on Learning Representations, ICLR 2018*.

[81] Sutton, R. S. and Barto, A. G. (2017). *Reinforcement learning: An introduction*.

[82] Svetlik, M., Leonetti, M., Sinapov, J., Shah, R., Walker, N., and Stone, P. (2017a). Automatic curriculum graph generation for reinforcement learning agents. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[83] Svetlik, M., Leonetti, M., Sinapov, J., Shah, R., Walker, N., and Stone, P. (2017b). Automatic curriculum graph generation for reinforcement learning agents. In *Thirty-First AAAI Conference on Artificial Intelligence*. Association for the Advancement of Artificial Intelligence.

[84] Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103.

[85] Szita, I. and Lőrincz, A. (2008). The many faces of optimism: a unifying approach. In *Proceedings of the 25th international conference on Machine learning*, pages 1048–1055.

[86] Szita, I. and Szepesvári, C. (2010). Model-based reinforcement learning with nearly tight exploration complexity bounds.

[87] Taylor, M. E. and Stone, P. (2005). Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59.

[88] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

[89] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

[90] Thorndike, E. L. (1898). Animal intelligence: an experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i.

[91] Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., et al. (2016). Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pages 3486–3494.

[92] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

[93] Whiteson, S. and Stone, P. (2003). Concurrent layered learning. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 193–200. ACM.

[94] Wiewiora, E. (2003). Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208.

[95] Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022.

[96] Wu, Y. and Tian, Y. (2017). Training agent for first-person shooter game with actor-critic curriculum learning. In *International Conference on Learning Representations (ICLR)*.

[97] Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., and Levine, S. (2017). Collective robot reinforcement learning with distributed asynchronous guided policy search. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–86. IEEE.

[98] Yang, B.-H. and Asada, H. (1996). Progressive learning and its application to robot impedance learning. *IEEE transactions on neural networks*, 7(4):941–952.

Appendix A

Tasks Details

In this appendix we provide a thorough description of all the experiments performed in the context of this work. We hereby detail the different domains and all the reinforcement learning tasks. Finally we discuss features characterizing positive and negative curricula in order to better understand the shape of the task sequencing problem in curriculum learning.

A.1 Domains

In this section we describe the three domains used in the experiments of this thesis discussing the main elements defining the complexity of a learning problem within them. Since we always used function approximators for computing the value function of each task, some of the domain features are referred to as inputs of the function approximator used in the domain under analysis.

A.1.1 Block Dude

Block Dude is a puzzle game where the agent has to stack boxes in order to climb over walls and reach the exit. The available actions are moving left, right, up, pick up a box and put down a box. The agent receives a reward of -1 for each action taken.

The variables used as input to tile coding are distance from the exit, distance from each box, distance from each edge of the map, direction of the agent (binary) and whether or not it is holding a box (also binary). It is important to notice how, in order to improve transferability, this state representation is agent-centric therefore all distances are computed with respect to the agent.

A.1.2 Gridworld

GridWorld is an implementation of an episodic grid-world domain. Each cell can be free, or occupied by a *fire*, *pit*, or *treasure*. The agent can move in the four cardinal directions, and the actions are deterministic. The reward is -2500 for entering a pit, -500 for entering a fire, -250 for entering the cell next to a fire, and 200 for entering a cell with the treasure. The reward is -1 in all other cases. The episodes terminate under one of these three conditions: the agent falls into a pit, reaches the treasure, or executes a maximum number of actions (50).

The variables fed to tile coding are the distance from the treasure (which is global and fulfills the Markov property), and distance from any pit or fire within a radius of 2 cells from the agent (which are local variables and alone do not fulfill the Markov property but allow the agent to learn how to deal with these objects when they are close, and transfer this knowledge). Also in Gridworld we used an agent-centric representation in order to improve transferability, therefore all distances are computed with respect to the agent.

A.1.3 MGEnv

MGEnv is a simulated micro-grid domain modeled out of real data from the PecanStreet Inc. database. The domain includes historical data about hourly energy consumption and solar energy generation of different buildings from January 2016 to December 2018.

A task in this domain is defined by the combination of three elements: the model of the electrical device to optimize; the user's monthly schedule, specifying the days in which the user wishes to run the device; the month of simulation, with the energy generation and base consumption of the given building. The device we used behaves as a time-shifting load: once started it runs for several time steps and cannot be interrupted before the end of its routine. This is the most challenging type of device present in the database. The goal is to find the best time of the day to run the given device, optimizing direct use of the generated energy, while respecting the user's device schedule.

The agent receives a reward of 30 when the device energy consumption is fully covered by the energy generated by the building, -10 when energy not generated by the building is used and -200 if the device is not run accordingly to the user schedule.

A.2 Tasks

In this section we provide a detailed description of each reinforcement learning task designed for each domain in our experiments. More specifically, we give a graphical representation and relative number of episodes for all the tasks in the artificial domains,

while we discuss the different features characterizing the MGEnv tasks to show the intuition behind their design.

A.2.1 Artificial Domains

Each task in Block Dude and Gridworld was manually designed in order to create a diverse set of tasks for each of the two domains.

It is possible to group these tasks by the type of strategy that the agent is expected to learn by solving the particular problem associated with the given task. For instance, tasks in the Block Dude domain can be grouped by the number of movable boxes in the environment or by the initial relative position of the agent and the exit block, while in Gridworld it is possible to use the number of fires and pits. These type of problem features foster the learner to discover specific strategies to solve a task which can later be useful to approach a different task in a curriculum or even the final task directly.

The tasks and relative strategies are of various complexities, and it is often the case that a particular task is the direct development of another one. For example, in Block Dude a task with three movable boxes for overcoming a tall obstacle is the evolution of a task with just one movable box for overcoming a shorter obstacle, or in Gridworld a task with multiple fires is a more complex version of a task with only one fire. Furthermore tasks teaching the same strategy vary in the actual environment dimension therefore in the number of possible states to visit.

By creating tasks in this way it is possible to investigate which features of the intermediate tasks are the most useful ones for solving a specific final task.

Some of the designed tasks are particularly complex to solve for a pure reinforcement learning agent without using any prior knowledge. These tasks were designed to be the final tasks of the different experiments, but it is also the case that one or multiple of them were included in the set of intermediate tasks in order to investigate their effect in a curriculum. The experiments in Block Dude and Gridworld were used for exploring all the possible curricula given a set of intermediate tasks, the final task and the maximum allowed length of a curriculum in the experiment under consideration. This fact implies that, in many curricula, the most complex tasks are followed by completely unrelated easier tasks, or they can even be the head of a curriculum. In these cases, the agent learning the given task can take a very large number of episodes before converging or get stuck in a local minimum because of bad initialization. In order to avoid wasting time testing extremely bad curricula, the number of episodes to spend on each intermediate task is determined in such a way that guarantees the learner convergence to the optimal policy when learning the given task from scratch. The number of episodes was calculated running several instances of each task and fixing it to a value that ensures that the agent could solve

the task more than 80% of the times. Because of this reason it is not always the case that the agent converges to the optimal policy for the task it is facing in the curriculum, and the transferable knowledge coming from the task under analysis can drastically change across different curricula.

Block Dude tasks

Figure A.1 shows a graphical representation of all the reinforcement learning tasks designed for the curriculum learning experiments in the Block Dude domain. The maximum number of actions for solving a problem in this domain is 50 for each task. This value corresponds to the maximum episode length in terms of learning steps. If during training the agent learns to solve a given task, it will be able to conclude an episode in a shorter amount of time. As we previously discussed, the number of episodes, instead, was determined for each task separately in order to guarantee convergence of an agent learning to solve the given task from scratch:

- 20 episodes: 1
- 30 episodes: 2 - 3 - 4
- 40 episodes: 5 - 7 - 8 - 13
- 50 episodes: 6 - 9 - 10 - 16 - 22
- 60 episodes: 17
- 80 episodes: 20
- 100 episodes: 11 - 14 - 15 - 21
- 150 episodes: 12 - 18 - 19

Gridworld tasks

Figure A.1 shows a graphical representation of all the reinforcement learning tasks designed for the curriculum learning experiments in the Gridworld domain. Also in this domain, the maximum number of actions in a learning episode is fixed to 50 for all tasks. Differently from Block Dude, in Gridworld an agent can take a shorter amount of time to conclude a learning episode in two cases: success by reaching the position of the treasure, failure by falling into a pit. The number of episodes was determined separately for each task:

- 30 episodes: 1 - 2 - 3
- 50 episodes: 17

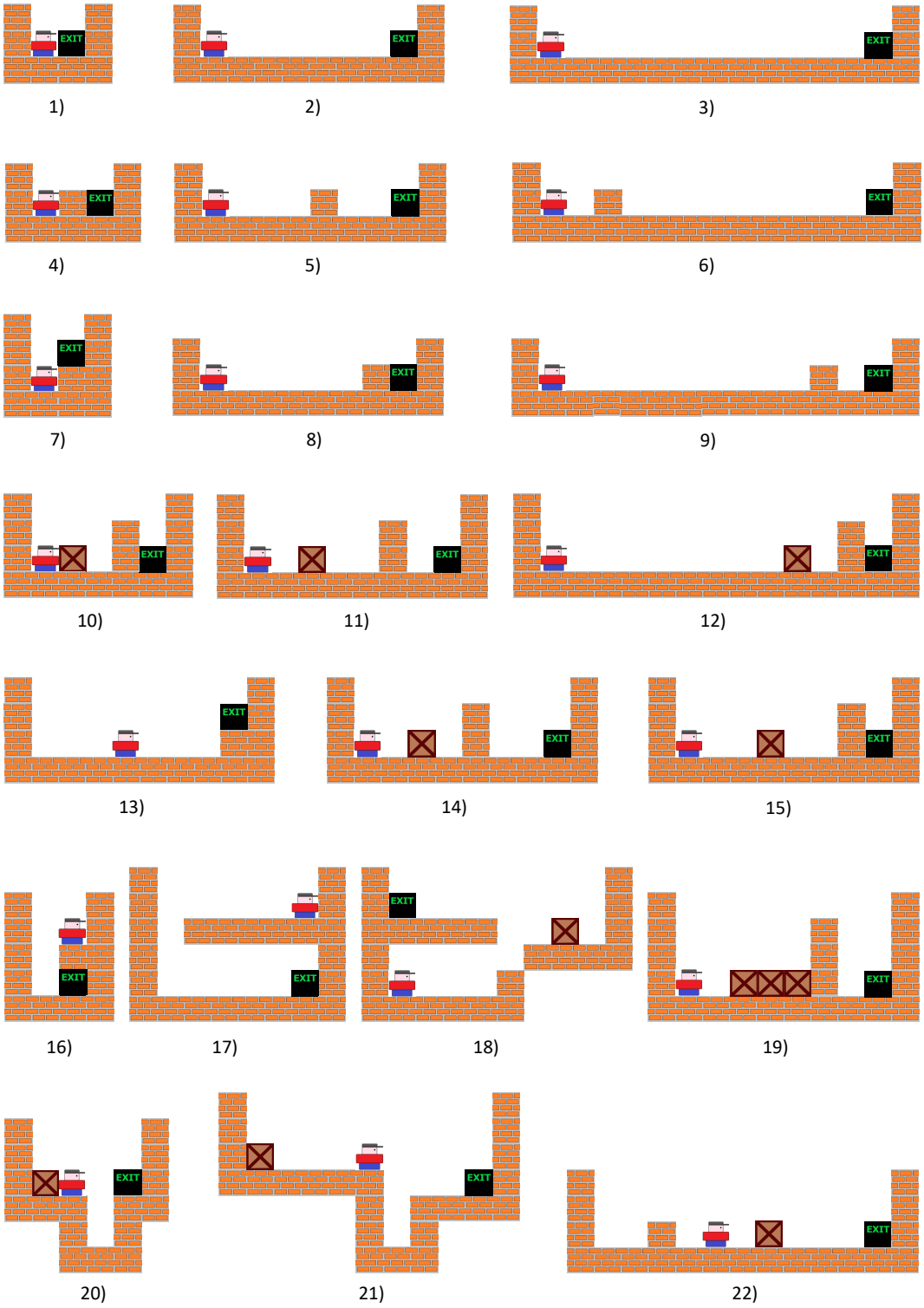


Fig. A.1 All the tasks designed and used for the experiments in the Block Dude domain

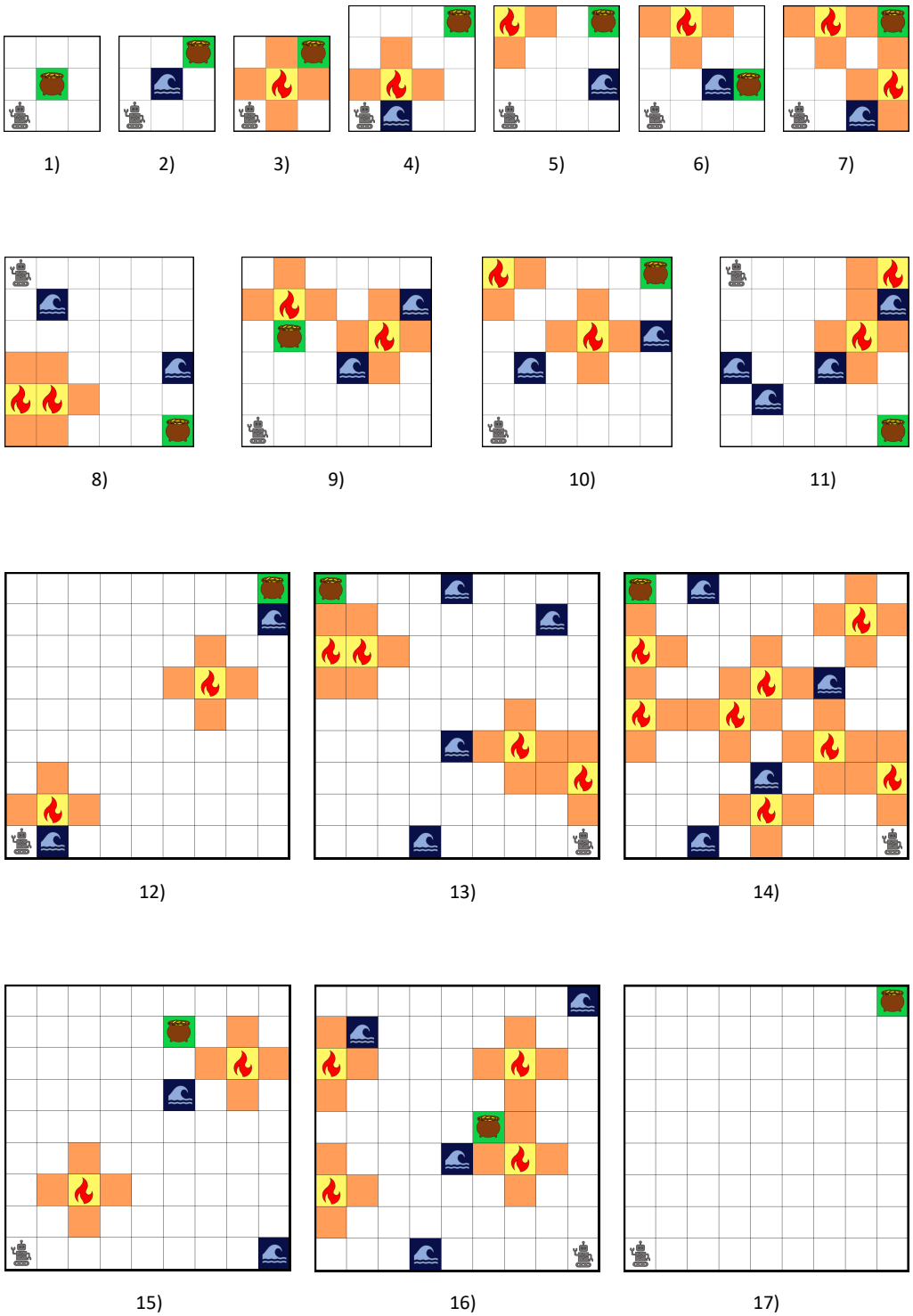


Fig. A.2 All the tasks designed and used for the experiments in the Gridworld domain

Intermediate Tasks			Training/Validation Tasks			Test Tasks		
Task ID	House ID	Month	Task ID	House ID	Month	Task ID	House ID	Month
1	9942	01/2016	6	9631	01/2017	11	9737	01/2017
2	9982	01/2016	7	9939	01/2017	12	9729	01/2017
3	9939	01/2016	8	9982	03/2017	13	9741	01/2017
4	9982	03/2016	9	9931	05/2017	14	9982	09/2017
5	9942	02/2016	10	9982	02/2017	15	9737	02/2017

Table A.1 Task description for the MGEnv domain

- 80 episodes: 4 - 5 - 6
- 100 episodes: 7
- 150 episodes: 8 - 9 - 10
- 300 episodes: 11 - 12 - 13 - 15 - 16
- 400 episodes: 14

A.2.2 MGEnv tasks

The MGEnv domain does not allow us to give a graphical representation of the different tasks used in our real-world experiments. Table A.1 provides all the details regarding the reinforcement learning problems designed for the MGEnv domain divided into three different sets (intermediate, training and test) following the formulation of the methodology described in Chapter 5.

It is possible to notice how the tasks in this domain do not vary in the type of user schedule or device to be run as we decided to fix them in order to improve knowledge transferability among the different tasks. Therefore, reinforcement learning problems in this domain differ only on the type of building (house) and the month during which the experiment is performed. Each different building in this domain has a different base energy consumption. Houses whose base consumption is high represent a more challenging problem for the agent to find a suitable time slot to run the required device. Also the month directly influences the complexity of a task as, for instance, winter months are more difficult than summer months.

In order to design a realistic experiment, tasks in the training and test sets are all from year 2017 as they need to come from the same distribution, while all the intermediate tasks are from one year earlier, 2016. This choice aims at modeling a scenario where we have access to historical data available for training our agent on, and define a curriculum to initialize the learner for future problems.

A.3 Experiments

In this section, for each experiment, we provide the final task and set of intermediate tasks defining the task sequencing problem in our experiments. No additional detail is provided for tasks in the MGEnv domain as the previously discussed Table A.1 already gives all the information needed for reproducing the experiments.
Experiment 1 in the Block Dude domain has a number of source tasks $n = 9$ and a maximum curriculum length $L = 5$:

- **Intermediate Tasks**: 3, 7, 9, 10, 13, 16, 19, 20, 22
- **Final Task**: 21

Experiment 2 in the Block Dude domain has a number of source tasks $n = 18$ and a maximum curriculum length $L = 3$:

- **Intermediate Tasks**: 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 15, 16, 17, 19, 20, 21
- **Final Task**: 18

Experiment 3 in the Gridworld domain has a number of source tasks $n = 12$ and a maximum curriculum length $L = 4$:

- **Intermediate Tasks**: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16
- **Final Task**: 13

Experiment 4 in the Gridworld domain has a number of source tasks $n = 7$ and a maximum curriculum length $L = 7$:

- **Intermediate Tasks**: 1, 6, 7, 9, 11, 13, 17
- **Final Task**: 14

A.3.1 Curricula

The experiments in the artificial domains explore all the curricula that can be generated by sequencing any of the available intermediate tasks for a maximum of L tasks in a curriculum. Because of this reason we can easily identify the best and worst curricula for a given experiment. Table A.2 report the list of the best and worst five curricula for each performance metric and experiment. Each curriculum is hereby represented by the IDs of the intermediate tasks composing it. Each list is sorted from the best curriculum to the worst one, therefore the first curriculum in the list is always the best one and the last is always the worst. Nevertheless, in some experiments, multiple curricula result to be the

	Best Curricula				Worst Curricula			
	CR	JS	TTT	MR	CR	JS	TTT	MR
Exp 1	[7,3,20] [16,3,20] [16,3,9,20] [16,9,20] [16,3,7,20]	[7,3,9,20] [16,3,7,9,20] [7,3,20] [7,3,16,9,20] [3,16,7,20]	[7] [16] [] [13,7] [13,16]	[3,9,10,20,19]	[22,19,10,13,9] [9,13,7,10,16] [10,7,20,19,13] [13,22,9,20,7] [20,10,19,22,13]	[22,13,7,19,9]	[19,10,22,20,9]	[7,19,13,16,3] [13,16,7,20,3] [10,9,16,19,3] [22,16,13,10,3] [13,20,10,9,3]
Exp 2	[5,17,15] [8,17,12] [10,17,15] [1,17,15] [5,17,12]	[4,17,11] [16,12,17] [8,17,15] [16,15,17] [3,21,11]	[1] [8] [4] [16] [1,4]	[1,2,7]	[19,11,14] [19,12,14] [2,21,6] [19,12,11] [11,19,14]	[1,2,15]	[1,2,20]	[17,2,20] [1,21,6] [3,21,6] [2,21,6] [11,6,21]
Exp 3	[6,15,5,1] [12,4,3,2] [10,3,2,9] [3,16,6,1] [9,4,3,1]	[6,3,2,8] [5,1,12,8] [9,1,6,8] [4,2,3,8] [1,9,8]	[1] [3] [] [4] [1,2]	[10,4,5,6] [8,3,5,10] [8,15,3,12] [8,9,3,10] [5,15,12,3]	[2,5,3,10] [3,5,4,6] [2,12,3,10] [4,2,12,10] [2,16,4,10]	[12,8,6,16] [5,6,10] [6,3,10,9] [5,9,16,8] [6,3,10,5]	[4,5,16,1]	[15,2,3,10] [1,16,10,3] [16,1,4,10] [12,1,10,4] [1,5,10,3]
Exp 4	[1,13,9,17,7] [11,9,17,13] [1,6,17,13,9,7] [1,11,17,13,7,6] [9,17,1,13,11,7]	[13,17,1,7,9,6,11] [7,17,9,11] [13,17,7,1,6,9,11] [13,17,1,6,7,9,11] [13,1,17,6,11]	[17] [9,1] [6,1] [1,7] [9]	[1,11,9,17,13,7] [17,13,1,7,6,9] [17,13,9,1,6,7] [9,13,6,17,11,7] [13,1,11]	[13,9,6,11,7,1] [6,7,13,9,11,17,1] [6,1,11,13,7,9,17] [6,13,11,9,7,1] [11,6,13,7,9,1]	[17,6,1,13,9,11,7] [6,1,13,9,7] [11,1,17,13,9,7,6] [1,13,7] [1,13,9,7]	[1,6,7,9,13,17,11]	[11,7,6,13,9,1] [17,13,9,7,11,6,1] [7,9,17,11,6,13,1] [7,6,13,11,9,1] [9,6,13,11,1]

Table A.2 List of best (on the left) and worst five curricula (on the right) for each performance metric and experiment in the artificial domains.

best or worst for a specific metric as they obtain the same performance value. For these cases we provide only one sample curriculum among all the available ones.

This table can help us identifying some general trait characterizing the curriculum learning problem. For instance, we can easily notice how long curricula are not necessarily achieving high performance values, but rather a careful sequencing of just part of the intermediate tasks can define extremely powerful curricula.

By focusing only on one performance metric we can notice even more interesting features. For optimizing time-to-threshold, shorter curricula are preferable since the higher the number of intermediate tasks in the curriculum, the greater the time the agent will take to learn all of them, to the extent that in experiments 1 and 3, the empty curriculum (learning from scratch) is the third best one. For both cumulative return and jumpstart the relative position of intermediate tasks identifies the best curricula. For example some tasks are often either the head or the tail of one of the best curricula for these performance metrics (this observation was at the origin of the design of HTS-CR). For what regards max return, instead, there are no major features defining a positive or negative curriculum and a more in depth analysis is required to draw stronger conclusions about curricula optimizing this metric.