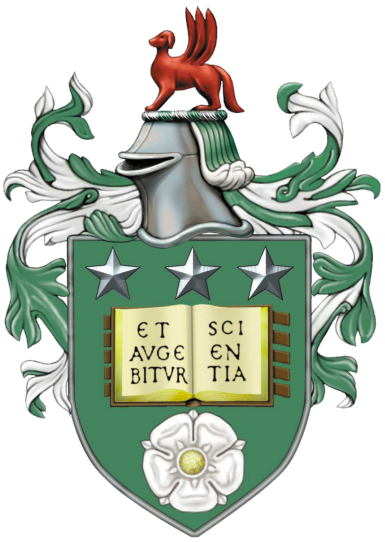
Efficient Meta-Reinforcement Learning

Ricardo Luna Gutierrez

Submitted in accordance with the requirements for the

degree of Doctor of Philosophy



The University of Leeds

School of Computing

March 2022

The candidate confirms that the work submitted is his/her/theirown, except where work which has formed part of jointly authored publications 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.

Luna Gutierrez, R.; and Leonetti, M. 2020. "Information-theoretic Task Selection for Meta-Reinforcement Learning". In Conference on Neural Information Processing Systems (NeurIPS).

Luna Gutierrez, R.; and Leonetti, M. 2021. "Meta-Reinforcement Learning for Heuristic Planning". In the International Conference on Automated Planning and Scheduling (ICAPS).

Acknowledgements

First and foremost, I thank my supervisors, Doctor Matteo Leonetti and Professor Anthony Cohn, for their invaluable guidance and support throughout the years. Thank you for all of our countless meetings and constant advice. And thank you for believing in me, it was a great privilege to be your student.

I thank my colleagues of the Robotics Lab, who were always willing to help and share their expertise. I'm grateful for all the interesting conversations and moments that we shared, you always made me feel welcome.

I thank my friends, who were with me during this journey, even though many of you were thousands of miles away, you always were there to share a good laugh.

Thanks to my parents, who have supported me my entire life, no matter where I go. I also thank my wife Cinthia for her constant love and patience, always pushing me and supporting to become better.

Thanks to my examiners, Netta Cohen and Stefano V. Albrecht for reviewing my thesis and give me their priceless feedback.

Finally, I want to acknowledge CONACYT-SENER for funding my research. Without their support none of this would be possible.

Abstract

In Meta-Reinforcement Learning (meta-RL) agents are trained on a set of tasks to prepare for and learn faster in new, unseen, but related tasks. The standard practice to build training sets in meta-RL is to use dense coverage of task distributions, generating a very large set of training tasks.

This thesis introduces a novel framework for meta-RL, in which models have access to a limited number of tasks to train on. With this framework in mind we propose task selection methods as well as an application that can benefit from it.

We introduce ITTS, a task selection method that select tasks that are different from one another and relevant a set of tasks sampled from the target distribution. The output is a smaller training set which can be learnt faster and performs better than training with all the available tasks. We experimentally evaluate the performance of ITTS in a variety of domains and show that ITTS improves the final performance of the agents in all of them.

We build insight on the learnt behaviours by meta-RL and propose FETA, a task selection method that improves over ITTS. FETA is a simpler and more cost efficient tasks selection method that filters tasks by taking advantage of policy transfer between tasks. We experimentally evaluate FETA and demonstrate that even tough the task selection process is more efficient, FETA performs equally or better than ITTS.

Finally, we make the first connection between meta-RL and heuristic planning, showing that heuristic functions meta-learned from planning problems can outperform both popular domain-independent heuristics and heuristics learned by supervised learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Reinforcement Learning has received a lot of attention in recent years due to impressive results in complex tasks such as Go [1], Starcraft [2] and Dota 2 [3], demonstrating the capabilities of Reinforcement Learning (RL) to surpass human performance in challenging tasks. Such accomplishments can be ascribed to the use of expressive neural networks in conjunction with RL methods. That success, however, came at a high cost, since massive quantities of training experience and computational resources were required. To achieve expert level performance in a game like Dota 2, for example, over 10,000 years of playing time were used to train the RL agents.

Standard RL methods require a large number of training iterations to learn a policy that is good enough to solve a task. Because no prior knowledge is carried over, this procedure is repeated each time the RL agent faces a new task. Humans, on the other hand, learn and build on previous experience; we use previously acquired information to tackle novel problems and swiftly adapt to new situations. For RL to have a real impact in the real world, it must be able to replicate these traits of the human intelligence.

The concept of building on prior knowledge to quickly adapt to new tasks or *learn to learn*, also known as meta-learning, has it roots in the 90s [4, 5]. However, recent

advances in deep learning and the fast growth of computer resources have sparked a new generation of meta-learning research in the scope of deep reinforcement learning, which is rapidly expanding [6, 7, 8, 9].

In Meta-Reinforcement Learning (meta-RL) agents are trained on a collection of tasks so that they can learn shared features across the tasks, and exploit the obtained knowledge to carry out unseen tasks that share properties with the training set. Despite the fact that various techniques utilize distinct optimization objectives and model architectures, meta-RL is usually presented as two learning systems. A low-level system that is in charge of adapting to new tasks and has a relatively short learning time, and a high-level system that learns slowly through a series of training tasks to enhance the low-level system [9]. Meta-RL has shown promising results in real-world tasks [10, 11], lending it credence.

Training tasks in meta-RL are designed with the intention of being representative of a family of test tasks, or a skill the agent is expected to learn. A common framework consists in modeling the range of tasks the agent may encounter as a distribution over all possible tasks. The standard practice in meta-RL is to use dense coverage of task distributions, generating a very large set of training tasks, which go up to the thousands [9], to meta-train a policy that can quickly adapt to new, unseen, tasks. However, dense sampling of tasks is highly computationally expensive, and in some cases infeasible. Standard meta-RL methods have not considered these scenarios where a limited number of training tasks is available. Furthermore, they have so far not considered that the training tasks may not be equally informative, beneficial, or promoting generalization. For Meta-RL to be considered practical, methods must be developed to constrain and overcome this task hunger issue, allowing meta-RL to succeed in settings when training tasks and resources are limited.

1.1 Task Selection

Tasks selection has been widely employed in different RL contexts, such as transfer learning [12, 13, 14, 15, 16, 17] and curriculum learning [18, 19, 20, 21, 22]. The goal of task selection is to create a training set that the performance of RL agents and improves the training process, by reducing training costs or increasing sample efficiency. Although task selection yielded promising results in the previously stated contexts, it has not been investigated in meta-RL.

In cases when there are few training tasks available or the training process must be limited, providing the agent access to a small fixed set of training tasks, task selection might be beneficial. Furthermore, even if the aim is to densely sample from a task distribution, the number of tasks sampled is always finite, and task selection can still be useful in many of those instances.

In this thesis, we make the first attempt to develop and employ task-selection methods to improve meta-RL agents' performance. We propose, ITTS, an Information-theoretic task selection algorithm for meta-RL. ITTS filters a set of training tasks identifying a subset of tasks that are both different enough from one another, and relevant to tasks sampled from the target distribution which we refer to as validation tasks. The outcome is a smaller training set, which can be learnt more quickly and results in better performance than the original set.

The results obtained with ITTS demonstrate the great impact that task selection can have on meta-RL agents, improving significantly their performance in all the 6 domains it was evaluated; however, ITTS has a few shortcomings. For ITTS to work and calculate its difference metric, the optimal policies of all the available tasks must be learnt. Moreover, ITTS makes use of a hyperparameter threshold to identify when two tasks are different enough, which needs to be correctly tuned

for ITTS to successfully work.

Building on the insight we got from ITTS and to address its shortcomings, we developed FETA, a different and more efficient task selection method. FETA selects tasks by leveraging the policy transfer properties of RL to discard potentially redundant data, improving the performance and sample efficiency of the meta-learned policy. During its task selection process FETA does not require the optimal policies of all the *available* tasks and only learns the optimal policies of the tasks that are being *selected*. During FETA's experimental evaluation we show that its simplicity does not affect performance, since it is able to perform similarly or better than ITTS.

1.2 Heuristic Planning

In recent years there has been a increasing interest in learning heuristic functions for classical planning with the aid of Deep Learning [23, 24, 25, 26, 27, 28]. These approaches are based on supervised learning, and learn from the optimal plans of previously solved planning problems. A crucial problem of heuristic learning is generalizing across different instances of the same planning domain, so that previous instances can inform the search on new, unseen, instances. Learning from optimal plans implies obtaining only information about a very limited area of the state space (the states along the optimal plan), and requires a high number of solved planning instances to achieve satisfactory generalization.

RL algorithms, on the other hand, learn a *value function* for tasks modelled as Markov Decision Processes (MDPs). The value function guides the exploration of the agent just like a heuristic function for the planner, but the estimate of the value function is made increasingly accurate through learning, and eventually

converges to the lowest cost (or equivalently, highest reward) from any state to a goal state. Since the optimal value function v^* is also the best possible heuristic, it seems natural to consider RL as a method to learn heuristic functions. Value functions, however, are specific to a particular task so for each new planning instance presented we would need to learn a new value function from scratch.
In this thesis, we make the first connection between meta-RL and classical heuristics, to overcome this RL shortcoming, and learn a heuristic function that can generalize to many different instances of a domain. Moreover, by employing task-selection to filter and select an ideal set of training tasks, we are able to learn good heuristic functions using a small number tasks and reduce training costs, an important feature in classical planning.

1.3 Contributions

The contributions of this thesis are as follows:

- We define the small sample framework for meta-RL, in which agents have access to a limited number of training tasks, and study the effect that tasks selection can have in those scenarios.
- We propose an Information-theoretic method for task selection in meta-RL (ITTS) that enhances the performance of meta-RL agents and reduces training costs.
- We identify ITTS shortcomings and propose FETA to address them. We show that FETA is able to perform equally of better than ITTS while being a more simple and efficient task selection process.
- We propose the use of meta-RL and task selection to learn domain-dependent

heuristic functions for classical planning, and experimentally show its effectiveness.

1.4 Thesis Outline

In Chapter 2 we introduce basic concepts and terminology to allow better understanding of the presented work. In Chapter 3 we present relevant literature, surveying different meta-RL algorithms, presenting different task selection methods that have been proposed for RL and discussing different approaches that have been proposed to learn heuristic functions using Deep Learning. In Chapter 4 we introduce ITTS, describing the task selection process and performing an experimental evaluation in different domains. In Chapter 5 we analyze and build insights on the policies learnt by a meta-RL method and introduce FETA, our improved task selection method. In Chapter 6 we present our approach to learn heuristic functions with meta-RL and proper task selection. In Chapter 7 we conclude this thesis, highlighting the contributions presented and discussing future work.

Chapter 2

Background

2.1 Reinforcement Learning

We consider the traditional RL setup, where a task is represented as a Markov Decision Process (MDP) $m = \langle S, A, r, p, p_0, \gamma, H \rangle$, where S is the set of states, A is the set of actions, $r : S \times A \to \mathbb{R}$ is a reward function, $P(s_{s_{t+1}}|s_t, a_t)$ is a transition distribution, P_0 is the initial state distribution, $0 \leq \gamma \leq 1$ is the discount factor and H the horizon. The agent's behavior is described by a policy $\pi(a|s)$ that returns the probability of taking action a in state s. An MDP m and a policy π induce an on-policy distribution $d_\pi^m(s)$ as the fraction of time steps spent in s during an episode. We assume that tasks are episodic, which means that the agent ultimately reaches an absorbing state that can never be left and from which the agent only receives 0 rewards.

With a possible trajectory $\tau := (s_0, a_0, ..., s_{H-1}, a_{H-1}, s_H)$, we define the expected return as $G(\tau) = \mathbb{E}[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)]$. The goal of the agent is to calculate an optimal policy π^*, which maximizes the expected return. The value function $v_\pi(s) = \mathbb{E}_\pi[G(h)]$ represents the expected return obtained by taking actions according to policy π starting from state s.

2.1.1 Policy Gradient

Policy gradient methods are a class of RL methods that aims to directly model and optimize the policy [29]. The policy is parametrized by a differentiable function with parameters θ and is learned *on-policy*, which means that the policy utilized for evaluation and exploration is the same policy that we are improving trough training. In policy gradient methods the objective is maximize the expected return by following its gradient, using the following update rule:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta). \tag{2.1}$$

where J(θ) is defined as:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0} \gamma^t r(s_t, a_t)] \tag{2.2}$$

The gradient of this loss function $J(\theta)$ is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\Psi_t \nabla_\theta log \pi_\theta(a_t|s_t)], \tag{2.3}$$

where Ψ_t varies depending on the implementation. One of the most commonly used is:

$$\Psi_t = A^\pi(s_t, r(s_t, a_t), s_{t+1}) = r(s_t, a_t) + v^\pi(s_{t+1}) - v^\pi(s_t) \tag{2.4}$$

The implementation shown in equation 2.4 makes use of the *advantage function* $A^\pi(s_t, r(s_t, a_t), s_{t+1})$, which estimates how much better or worse is action a_t, taken by the agent at state s_t, when compared to the average return expected at s_t. If the learning approach learns the value function v^π, it falls into a class of methods called *Actor-Critic*, where the *Actor* represents the policy and the *Critic* the value

function.

2.2 Deep Reinforcement Learning

In Deep Reinforcement Learning (Deep RL) the policy and/or value function are modeled by neural networks (NN). In the standard Deep RL training cycle an agent collects samples by acting in the environment following certain policy, and the acquired samples are then utilized to update the parameters of the NN via back-propagation. By intrinsically learning key abstract properties to estimate the value function or build a policy, NNs are able generalize over large and continuous state spaces.

However, NN have high sample complexity and require data to be independent and identically distributed. This property makes it difficult for a sequential process like RL because the data is highly correlated, which can cause problems in the stability of the learning process. Much RL research has been devoted to resolving these issues, and methods for doing so have been developed, one of which is Proximal Policy Optimization, a popular method that we employ extensively in this thesis.

2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [30] is an on-policy Actor-Critic RL learning algorithm. A problem presented in Actor-Critic methods is that the stability of the learning process is highly sensitive to changes in the policy and, as a result, the generated transition samples, since both the policy and the value function are updated based on the samples collected by the agent. A large faulty policy update may result in a stream of transition samples with weak or no reward signal from the environment, causing the estimate of the advantage function to become less

accurate, further altering the policy and making recovery difficult (*catastrophic forgetting*).

To address this issue, PPO provides a simple first-order optimisation mechanism for limiting policy updates by clipping the policy ratio. PPO optimizes a policy π_θ, represented as a neural network with parameters θ, using gradient ascent on the objective function:

$$L(s_t, a_t, \theta_k, \theta) = \mathbb{E}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(a_t|s_t)), \\ clip(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}}(a_t|s_t)]$$

where π_{θ_k} is the policy at the start of an episode, before the weights are updated. The hyperparameter ϵ ensures that the policy does not change drastically, and takes small values, usually in $[0.1, 0.2]$.

The goal of this formulation is to ensure that policy changes are minor at each gradient step, lowering the likelihood of catastrophic forgetting and making learning more stable.

2.4 Meta-Reinforcement Learning

An agent in the Meta-Reinforcement Learning framework has access to a variety of different tasks to train on, allowing it to learn about the domain and adapt fast to new tasks. Given a potentially infinite set tasks $\mathcal{M}$ and a distribution over the tasks $p(\mathcal{M})$, the agent is presented T tasks $\mathcal{T} = \{m_i\}_{i=1}^{T}$ to train on. A common meta-RL problem formulation is that the agent is expected to learn an optimal policy of a new task $m_j \sim p(\mathcal{M})$, such that $m_j \notin \mathcal{T}$ within a "few samples". Since, in general, it is not possible to established whether a learned policy is, indeed,

optimal, we formulate our objective in terms of achieving sufficient transfer, as measured by a binary function $f : \Pi_{\mathcal{T}} \times \mathcal{T} \rightarrow \{0, 1\}$, where $\Pi_{\mathcal{T}}$ is the set of all possible policies of the tasks in $\mathcal{T}$. The function $f(\pi, m_j) = 1$ if the meta-learned policy π is a *sufficient* policy for m_j. The function f is application dependent. Examples of possible implementations are: computing whether π is optimal for m_j, whether it achieves an expected return above a threshold, or whether it reaches a goal state. Since we consider only episodic tasks, we will measure samples in terms of episodes, expecting the learned policy π to be such that $f(\pi, m_j)$ on test tasks m_j within a number of episodes N_e.

Although, at an architectural level, most of meta-learning methods can be described as a combination of two learning systems, a lower-level system which is responsible for adapting to new tasks and has relatively fast learning time, and a higher-level system that slowly learns across the set of training tasks to improve the lower-level system [7], the optimization objective varies between each Meta-RL method.

Meta-RL can be divided into two categories [8]: gradient-based methods and context-based methods. Gradient-based methods use hyperparameters, meta-learned loss functions or policy gradients to learn from sampled transitions from tasks [6, 31, 32, 33, 34, 35, 36, 37]. Context-based methods, on the other hand, train models to utilize prior states and actions sequences or latent variables as a form of task-specific *context* [7, 8, 9, 38]. In gradient-based methods the agent adapts to the new tasks through further learning, adapting their parameters, while context-based methods adapt by feeding in experience into a latent space representation.

The commonly assumed meta-RL framework does not guarantee that the set of tasks in $\mathcal{M}$ are at all related, or that transfer is even possible. This is in general

left to the intuition of the designer, and much ingenuity has been employed in existing meta-RL applications to generate appropriate set of tasks and corresponding distributions $p(\mathcal{M})$ [39].

2.5 Planning

We address learning heuristics for classical planning, that is, for deterministic, sequential planning problems, in part of this work. The classical MDP-based RL framework is more broad, and permits immediate extensions to probabilistic planning. One of the most popular ways to represent deterministic and fully observable planning tasks is PDDL [40, 41].

A PDDL-defined planning job is divided into two parts: domain description and problem description. A planning domain is a tuple $\langle D, A \rangle$, where D is a collection of predicates, which define the state of the objects we are interested and can take a true or false value, and A is a set of parametrized actions, which define how the world can be changed. These actions are restricted by a set of precondition predicates that must be met in order for the action to be executed, and they provide a description of the effects that will be applied to the current state if the execution is successful.

A planning problem is described as $\langle O, I, \mathcal{G}, c \rangle$ where O is a set of domain objects, which are the different components presented in a planning task, I is the starting state, $\mathcal{G}$ is a set of goals, and $c(s, a, s')$ is the cost of the transition landing in s' after performing action a in state s. For shortest planning problems we assume $c(s, a, s') = 1$ for every transition. The starting state is defined as the set of objects O and predicates D that are true before any action is done.

A heuristic function $h(s)$ estimates the cost of achieving the goal from state s, and

is used to guide the planning search by selecting states with low cost estimations. When a heuristic never overestimates the cost of attaining a goal, it is deemed admissible, and an A* planner would find an optimal solution when using it [42]. Since an optimal value function v^* gives the expected cost to achieve the goal under the optimal policy, $h(s) = v^*(s)$ would be a perfect heuristic, and a planner such as A* would only expand states along the optimal plan by following it.

In classical planning, the h^{max}, h^{add} and LM-cut heuristics, which can be found in the FastDownward system [43, 44], are some of the most prominent heuristic functions. All these heuristics work by considering a relaxed version of the problem, meaning that once a predicate has been achieved and its value has changed to *true*, it stays achieved during all the planning search. h^{max} and h^{add} take a state in the planning graph and use all possible actions to make every predicate in that state true and the heuristic value they create is based on the cost of the actions required to achieve that. The heuristic for a given state for h^{add} is the total cost of achieving every predicate in that state, whereas the heuristic for a given state for h^{max} is the cost of the most costly predicate in that state. LM-Cut on the other hand, is calculated by iteratively computing h^{max}, finding a disjunctive action landmark, and reducing the cost of these actions until the the value of h^{max} becomes zero.

Chapter 3

Related Work

In meta-RL, agents are trained with the aim of preparing them to perform and adapt swiftly to tasks that have not been seen by the agent before but that share some properties with the tasks in the meta-RL agent's training set. Many successful reinforcement learning applications, such as navigation tasks [6, 7, 9, 38], classic control tasks [31, 34, 45, 46], and locomotion tasks [6, 8, 47], have demonstrated the potential of using meta-learning in RL. In this section, we first present a survey of the most relevant methods in meta-RL. We then present different task selection methods that have been employed in RL to improve performance and sample efficiency. Finally, we describe different deep learning techniques that have been developed for heuristic planning, a research topic in which meta-RL has not been studied but that we investigate in this thesis.

3.1 Gradient-based methods

In meta-RL, gradient-based methods use hyperparameters, meta-learned loss functions or policy gradients to learn from sampled tasks' transitions. This category of methods adapts to the new tasks through further learning, adapting their parameters. Since gradient-based approaches will be used extensively in this work,

we describe the most important methods in this category below.

3.1.1 Model-Agnostic Meta-Learning (MAML)

MAML [6] is one of the most popular approaches in meta-learning. MAML is a model-agnostic meta-learning approach that works with any model that learns by gradient descent and can be used to a variety of problems including classification, regression, and reinforcement learning. In MAML, a model's parameters are explicitly trained on variety of tasks, during the meta-training process, so that a small number of gradient steps with little training data from a new task produces maximally effective behaviour on a new task at test time.

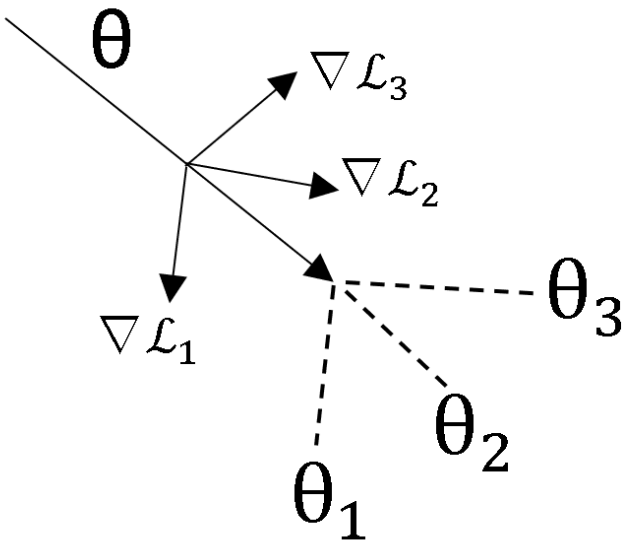


Figure 3.1: Diagram adapted from the MAML original paper [6], optimizing meta-parameters θ to quickly adapt to new tasks.

In a model or neural network represented by a parametrized function f_θ with parameters θ, given a task m_i, the model's parameters θ become θ' when adapting to the task m_i. One or more gradient steps on task m_i are used to compute the updated parameters θ'. In the case of one gradient step, this update is calculated

as follows:

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}_{m_i}(f_\theta) \tag{3.1}$$

where $\mathcal{L}_{m_i}$ is the loss computed using samples from m_i and α is the step size, which can be fixed as a hyperparameter. This step is known as the *inner* loop.
The above equation optimizes for one task. However, to achieve good generalization, we must find the optimal parameters θ from which we can achieve fast adaptation to a new task. To achieve this, the model's parameters θ are optimized across a large set of tasks $\mathcal{T}$. This meta-optimization (*outer* loop) is done via stochastic gradient descent as follows:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{m_i \in \mathcal{T}} \mathcal{L}_{m_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{m_i}(f_\theta)}) \tag{3.2}$$

where β is the meta step size. During meta-training, this optimization is repeated many times collecting new samples in each instance of the loop.
The optimization loops are general and can be applied to both supervised learning and reinforcement learning. In RL, the loss is defined as the negative return, with the goal of maximizing it.

3.1.2 First-Order MAML (FOMAML)

MAML's meta-gradient update comprises a gradient through a gradient as shown in equation 3.1, therefore its meta update relies on second order derivatives. FOMAML is a modified version of MAML, with the same goal and training procedure, that omits second derivatives making computation less expensive and simpler to implement [48, 6].

For MAML computing k inner gradients, $k \geq 1$, starting from the initial meta parameters θ is done as:

$$\begin{aligned}
\theta_0 &= \theta \\
\theta_1 &= \theta_0 - \alpha \nabla_\theta \mathcal{L}(\theta_0) \\
\theta_2 &= \theta_1 - \alpha \nabla_\theta \mathcal{L}(\theta_1) \\
&... \\
\theta_k &= \theta_{k-1} - \alpha \nabla_\theta \mathcal{L}(\theta_{k-1})
\end{aligned}$$

while the outer loop samples new samples of data to update the meta-objective:

$$\theta \leftarrow \theta - \beta g_{\text{MAML}} \tag{3.3}$$

where the MAML gradient g_{MAML} is defined as:

$$\begin{aligned}
g_{\text{MAML}} &= \nabla_\theta \mathcal{L}(\theta_k) \cdot \nabla_{\theta_{k-1}} \theta_k ... \nabla_{\theta_0} \theta_1 \cdot \nabla_\theta \theta_0 \\
&= \nabla_\theta \mathcal{L}(\theta_k) \cdot (\prod_{i=1}^{k} \nabla_{\theta_{i-1}} \theta_i) \cdot I \\
&= \nabla_\theta \mathcal{L}(\theta_k) \cdot (\prod_{i=1}^{k} I - \alpha \nabla_{\theta_{i-1}} (\nabla_\theta \mathcal{L}(\theta_{i-1})))
\end{aligned}$$

FOMAML simplifies the meta update to the derivative of the last inner gradient update ignoring the second order derivatives as:

$$g_{FOMAML} = \nabla_{\theta_k} \mathcal{L}(\theta_k) \tag{3.4}$$

3.1.3 Reptile

Reptile [48] is a simple meta-learning method similar to FOMAML, that has been used mainly for supervised learning, but that can be extended to RL. Learning is used in Reptile to initialize the parameters of a neural network so that when a new task is provided, the model can generalize to the new task learning from a limited number of examples.

Reptile's optimization works directly with the parameters of the model used. With a neural model parametrized by θ that trains on T tasks, Reptile's update rule is:

$$\theta \leftarrow \theta + \beta \frac{1}{T} \sum_{i=1}^{N} (\theta_i' - \theta) \tag{3.5}$$

where β is meta step size and θ_i' are the parameters obtained after updating θ using data sampled from a task m_i with Stochastic Gradient Descent (SGD) or the Adam optimizer. Similar to MAML, Reptile's update rule is run in a loop, collecting new samples from the training tasks each iteration.

Although, Reptile's optimization process is simple, it has shown similar performance to FOMAML in some applications [48].

3.1.4 Evolved Policy Gradients (EPG)

EPG [33] encode knowledge obtained from past experiences implicitly through a learned loss function instead of encoding it explicitly through a behaviour policy. The aim is to learn a loss function that a RL agent could use to learn quickly a novel task.

Similar to most meta-learning methods, EPG consist of two optimization loops. An inner loop in which the agent learns to solve a task by minimizing a loss function provided by the outer loop. And an outer loop that adjusts the parameters of the

loss function to maximize the agent's final returns after learning in the inner loop. While the inner loop is optimized in the standard way with SGD, the optimization of the outer loop cannot be written explicitly as a differentiable equation. To face this optimization challenge, EPG turned to evolution strategies [49].

In EPG a set of N agents is trained with a loss function $L_{\phi+\sigma\epsilon_n}$ parameterized by ϕ adding Gaussian noise $\epsilon_n \sim \mathcal{N}(0, \mathbf{I})$ of standard deviation σ.

In the inner loop EPG's updates the policy parameters θ through SGD according to the loss function $L_{\phi+\sigma\epsilon_n}$ using trajectories τ sampled from steps $i - M, ..., i$ for each agent as follows:

$$\theta_i \leftarrow \theta - \alpha \nabla_\theta L_{\phi+\sigma\epsilon_i}(\pi_\theta, \tau_{i-M,...,i}) \tag{3.6}$$

At the end of the inner loop, each policy π_i is evaluated in multiple random sampled trajectories to calculate the mean return G_n. The outer-loop then uses the returns $\{G_n\}_{n=1}^N$ from all agents to update the loss function parameters ϕ as:

$$\phi \leftarrow \phi - \beta \frac{1}{\sigma N} \sum_{n=1}^{N} \epsilon_n G_n \tag{3.7}$$

where β is the outer loop learning rate. At test time, the parameters ϕ are fixed while parameters policy parameters θ are updated by training on the test task.

3.1.5 Meta-Gradient Reinforcement Learning

In RL the return function G_t depends on a few hyperparameters that are often set and held fixed during training such as the discount factor γ and the bootstrapping parameter λ. This bootstrapping parameter λ can be found in some RL algorithms and defines how much credit or weight is assigned to further back states and actions. In the Meta-Gradient RL [32] framework, parameters $\eta(\gamma, \lambda)$ are considered

meta-parameters which can be learned during training. The idea is that the return G_t becomes a function with meta-parameters η that adapt online when the agent interacts with the environment, allowing the return G_t to adjust to a specific new task and the changes in the learning context, to improve the performance of Deep RL agents on large scale applications.

In the training process, policy parameters θ are updated following an underlying RL algorithm which results in new parameters θ'. The update is defined as:

$$\theta' = \theta + f(\tau, \theta, \eta) \tag{3.8}$$

where τ is a sequence of trajectories. The gradient of these updates is represented as $(d\theta/d\eta)$ and illustrates how the meta-parameters η influenced the new parameters.

With a meta-objective $J(\tau, \theta, \eta)$, their training process starts with a collection samples τ to update policy parameters θ which results in the updated parameters θ'. The updated policy $\pi_{\theta'}$ is used to collect a new set of samples τ' and performance is measured with fixed parameters $\overline{\eta}$ as $J(\eta', \theta', \overline{\eta})$.

The chain rule is used in meta-gradient RL to determine the gradient of the meta-objective with regard to parameters η:

$$\frac{\partial J(\eta', \theta', \overline{\eta})}{\partial \eta} = \frac{\partial J(\eta', \theta', \overline{\eta})}{\partial \eta} \frac{d\theta'}{d\eta} \tag{3.9}$$

which is simplified to get:

$$\Delta_\eta = -\beta \frac{\partial J(\eta', \theta', \overline{\eta})}{\partial \eta} \frac{\partial J(\eta, \theta', \eta)}{\partial \eta} \tag{3.10}$$

To optimize the meta-objective, meta-parameters η are updated using an optimization method such as SGD, which updates η in the direction of the meta-gradient.

3.1.6 Meta-Reinforcement Learning of Structured Exploration Strategies (MAESN)

One of the main reasons for RL sample inefficiency is poor exploration, in which the agent spends much time in states that do not provide any relevant information to achieve the desired objective or goal. MAESN [47] is a meta-RL algorithm that adapts to new tasks by injecting learnt structured stochasticity into a latent space to facilitate effective exploration and following the policy gradient. The objective of MAESN is to obtain high quality exploration strategies by integrating learnt time-correlated noise via its meta-learned latent space, and actively training both the policy parameters and the latent exploration space for quick adaptation.

MAESN conditions the policy on a per-task learnt latent distribution $z \sim q_{w_i}(z)$ with variational parameters w_i for tasks $m_i = 1,2,\ldots,T$ resulting in policy $\pi(a|s,z)$. The latent variable is used to inject temporally correlated, coherent stochasticity into a policy to improve exploration of an agent when facing a new task. The idea is to achieve coherent exploration by randomly sampling from useful behaviours and omit behaviours that might not be useful.

During the meta-training procedure MAESN optimizes both the policy parameters θ and the parameters w_i to maximize the expected reward after a policy gradient update. Additionally the Kullback–Leibler divergence (KL-divergence) between the per-task latent distribution $q_{w_i}(z)$ and prior $p(z)$ is added to the loss function, to reduce overfitting over the training set, so at meta-test time sampling from the prior $p(z)$ for a new task still produces effective exploration.

For each meta-training iteration MAESN performs an inner gradient update on

parameters θ and w_i training on task $m_i \in \mathcal{T}$, where $\mathcal{T}$ is the set of training tasks, which results in parameters θ' and w'_i, optimizing both parameters as follows:

$$\max_{\theta, w_i} \sum_{m_i \in \mathcal{T}} \mathbb{E}_{\substack{a_t \sim \pi(a_t|s_t;\theta'_i,z'_i) \\ z'_i \sim q_{w'_i(.)}}} \left[\sum_t R_i(s_t) \right] - \sum_{m \in \mathcal{T}} D_{KL}(q_{w_i}(.)||p(z)) \tag{3.11}$$

$$w'_i = w_i + \alpha_w \cdot \nabla_{w_i} \mathbb{E}_{\substack{a_t \sim \pi(a_t|s_t;\theta_i,z_i) \\ z_i \sim q_{w_i(.)}}} \left[\sum_t R_i(s_t) \right] \tag{3.12}$$

$$\theta'_i = \theta + \alpha_\theta \cdot \nabla_{\theta} \mathbb{E}_{\substack{a_t \sim \pi(a_t|s_t;\theta_i,z_i) \\ z_i \sim q_{w_i(.)}}} \left[\sum_t R_i(s_t) \right] \tag{3.13}$$

where α are the per-parameter step sizes.

3.2 Context-based methods

The second category of meta-RL methods are context-based methods. Context-based methods, as opposed to gradient-based, train models to use past states and action sequences or latent variables as a form of task-specific context. These methods adapt by feeding in experience into a latent space representation which conditions the agent's behaviour. Context-based methods do not require to adapt their parameters during the test phase, making them more desirable for applications such as heuristic planning, where the goal is to discover a successful plan as quickly as feasible. To better understand how they work, we will describe relevant methods in this category.

3.2.1 RL^2

In the RL^2 [7, 9] framework the previous reward r_{t-1} and previous action a_{t-1} are integrated with the current state s_t to form the observation, at time step t, to be fed to the training model. The purpose is to allow the model to learn the

interactions between states, actions and rewards in the current domain, building a context that identifies key properties of the task at hand and conditions the policy to previous interactions.

RL^2 makes use of an Long short-term memory network (LSTM) [50] in which its hidden states serve as memory for monitoring the observed trajectories. The agent is trained with a set of different tasks in an episodic manner. At the start of each training episode, a task $m \in \mathcal{T}$ is sampled and the internal state of the LSTM is reset. The agent then interacts with the environment executing its policy and collecting experience (in the form of state, action, and reward samples) as results of these interactions. The collected experience is used to train the network weights to learn a policy and a value function that enable the agent to maximize the sum of the rewards obtained over all the episodes. The idea is to train the LSTM to represent the process that underlies learning each task so that it can learn efficiently when faced with new, unseen tasks. At test time, adjustment of the value function and policy to the new task takes place observing the first few transitions, as the internal memory fills up and creates the context to adapt the value function and policy to the current task, without training.

3.2.2 Simple Neural Attentive Meta-Learner (SNAIL)

SNAIL [38] is meta-learning method which is applicable to both supervised-learning and RL. In contrast to RL^2, it does not make use of Recurrent Neural Networks to serve as memory, but instead they make use of an architecture which combines temporal convolutions (TC) [51] and a soft attention mechanism [52]. The TCs allow the meta-learner to collect contextual information from previous experience, whereas attention is utilized to pinpoint individual pieces of information within that context.

SNAIL is composed of a few building blocks [53]. The first is a *dense* block that performs a single causal 1D-convolution to the input and then concatenates the result with its input. The second is a *TC block* that is made up of a succession of dense blocks with dilation rates of the temporal convolutions which rise exponentially. The third block is an *attention block* which learns to focus on the most important aspects of previous experience.

During meta-training, SNAIL is presented with a series of state-action-reward tuples $(s_1,\text{-},\text{-}),\ldots,(s_t,a_{t-1},r_{t-1})$ and generates a distribution across actions at each time t depending on the current state (or state) s_t as well as past states, actions, and rewards. Similar to RL^2, at test time SNAIL does not update its parameters but instead absorbs experience through its hidden space to produce context and adapt to new tasks.

3.2.3 Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables (PEARL)

PEARL [8] is an off-policy meta-RL method that integrates an existing off-policy RL algorithm with online inference of probabilistic context variables. In contrast to the previous context-based methods described, it does not use recurrence but instead achieves fast adaptation by learning a probabilistic latent representation of prior experience. PEARL can be considered a meta-learning variant of posterior sampling [54]. In posterior sampling, the agent maintains a posterior distribution of possible MDPs and iteratively samples a MDP from this distribution and acts optimally according to it, updating the distribution to the collected experience and improving the agent's exploration. For the meta-RL case PEARL represents their distribution over MDPs as a distribution over Q-functions.

PEARL conditions an agent policy on a latent probabilistic context variable Z as

$\pi_\theta(a|s,z)$. This variable Z stores information on how the current task should be completed, allowing the policy π_θ to adjust its behavior according to the task. To learn to infer Z an inference network $q_\phi(z|c)$, parametrized by ϕ, is trained which estimates the posterior $p(z|c)$, where c is experience(context) collected from the task. This posterior can be seen as a Gaussian.

For training the RL agent Soft-Actor-Critic (SAC) [55] is used. During meta-training, PEARL leverages data from a set of training tasks to learn to infer the value of Z from recent experience in the current task, while it optimizes the agent's policy to solve the task given samples from the posterior over Z. The inference network is optimized to model the Q value functions with gradients from the critic and an information bottleneck $\mathcal{N}$ that constraints mutual information between Z and c. The intuition behind this bottleneck is to limit z to contain only information from c that is useful to adapt to the current task, therefore reducing overfitting to the training tasks.

While the parameters ϕ of the inference network Z are optimized during meta-training, the latent context for a new task is simply inferred from collected experience at meta-test time, in a feedforward manner without updating its parameters, allowing the agent to adapt to the task at hand due to the information contained in Z which conditions policy $\pi_\theta(a|s,z)$.

3.2.4 Small Sample Meta-RL

The previously described methods have been shown to be successful in different applications; however, none of them has considered the performance of their method in a few-task setting, where training sets might need to be carefully built. In this work, we focus on a small sample framework in which agents have access to a limited number of tasks to train on. In this framework, each training task has

a substantial influence on the agent's final performance, hence creating the best possible selection of training tasks is critical.

With this framework in mind, we propose task selection methods that filter a set of available training tasks to a build better and smaller training set, which can be learnt faster and results in better performance.

We use MAML (and FOMAML) [6] and RL^2 [7, 9] for our experimental evaluations, since they are some the most popular and simplest methods in each category. The simplicity of these methods allows us to focus our evaluations on what is of interest in this work, which is the quality of the training sets selected by our task selection methods.

3.3 Task Selection for Reinforcement Learning

3.3.1 Transfer Learning

Transfer learning in RL can be described as an approach that enables an agent to leverage acquired knowledge from prior interactions in a *source task* to achieve better or faster performance in a separate but related *target task*. The knowledge acquired from the source task can give the agent information to avoid less promising states in the target task, improving its exploration and reducing the time required to learn the task. However, for transfer to be *positive*, meaning that transferring knowledge from the source task to the target task improves the performance of the agent compared to training from scratch without previous information, both tasks, the source and the target, must be related.

To ensure positive transfer between source tasks and target tasks, different approaches for measuring task similarity, task clustering and task selection have been proposed [56, 57]. For instance, Ammar et al. [12] proposed the use of

Restricted Boltzmann Machines (RBM) to measure similarity between a pair of tasks. In their work they use each task in a set of source tasks to train a RBM that represents the task transitions in a richer feature space. The trained RBMs were used to reconstruct samples from a new task, and calculated a reconstruction error by measuring the Euclidean distance between the original sample and its reconstruction, the lower the error, the more similar two task are.

Mahmud et al. [17], proposed to measure similarity between tasks by extracting the value of the initial state on the source task and target task using value function learnt on the source task. They calculated the difference between the values and took it as a measurement of similarity; two tasks were considered to be closely related if the difference between the values was small.

Karimpanal and Bouffanais [14], proposed to store the value weights of a model which has been trained on a source task. The stored weights are used as input in a Self-Organizing Map (SOM), which is capable of extracting characteristic features of the weights vectors associated with the source tasks. Once all source tasks have been assimilated by the SOM, it is used to aid in the training of the target task, by transferring the value function weight vector from the source task that is most similar, to the target task value function weight vector.

Carroll and Seppi [16], proposed to compare two tasks by counting the number of states in which their polices are identical; the larger the count, the more similar the two tasks are. In the same work, they propose using the mean squared error between the stored values of two tasks state by state as a different measurement of similarity, providing that both tasks have the same state space.

Sinapova et al. [58], proposed a framework for selecting source tasks in the absence of a defined model or target by using attribute-value pairs associated with each task to determine the expected benefit of transfer given a source-target task pair.

The idea is to employ parameters or features that characterize two tasks to help an agent learn the advantages of transferring knowledge from one task to the other. Although these methods have been proved to be beneficial for single task transfer scenarios, there are some distinctions between the transfer learning framework and the main topic of this thesis, which is meta-RL. In transfer learning, a prior is obtained by learning on a source task without considering the use of a meta-objective, while in meta-RL the prior is defined by the outer-loop optimization that evaluates the benefit of the prior when learning a new task [39]. Moreover, in transfer learning, we look to build pairs of tasks that are closely related so that transferring information from a source task to a target task speeds up training and improves performance. However, it has been shown that in multi-task settings, training on a set of closely related tasks can cause the learning model to overfit, reducing performance and generalization [59]. One of the main purposes of meta-RL is to build an agent that can generalize to a large number of tasks, and utilizing a set of closely related tasks to train the model can hurt this quality.

3.3.2 Curriculum Learning

Transfer learning transfers knowledge from one or more source tasks directly to a target task, assuming that the transfer occurs in one single step, directly from the source to the target. In curriculum learning, on the other hand, a curriculum comprised of a sequence of tasks is created, in which transfer occurs in various steps, transferring from one task in the sequence to the next one, until reaching the final task [60]. Tasks are usually organized by difficulty; the agent starts by training on the easiest problem in the set and then transfers the learned information to train on the next, more difficult task; this process continues until the final task, the most difficult of all, is achieved. The goal is to be able to learn the final task

faster and/or better than learning the task from scratch.

In the framework of curriculum learning, optimal task selection has been studied [18, 19, 20, 21, 22, 61]. Tasks are selected and ordered to build an optimal task sequence to train for a given set of test tasks. These approaches select and sort tasks to be of increasing complexity, so as to identify the best sequence that maximizes sample efficiency and/or final performance in the test tasks. Foglino et al. [21], for example, provided a method to automatically discover the best curriculum for a target task by doing a heuristic search of the many potential task sequences that may be formed from a given collection of source tasks. These various sequences are assessed by comparing their performance in a series of simulated target tasks (placeholders of the real target tasks used only for evaluation), therefore determining the best performing sequence from the given source task set. In contrast, Svetlik et al. [20] propose a method to build a curriculum sequence without task execution. Instead they use task descriptors and a heuristic measure of task similarity to select and build their curriculum.

Although task selection have shown great performance in the framework of curriculum learning, training is sequential and tasks are used one at a time with the aim to improve the performance on one final target task. Many of these methods test different combinations of tasks doing evaluations in different steps of the training process. However, most of standard meta-RL methods train a meta-policy that contains information of all the training tasks, sampling from large batches of training tasks at the same time and ignoring its order. Moreover, the main focus of Curriculum Learning is not generalize to a potentially large number of unseen test tasks but create a curriculum of tasks that allows an agent to achieve good performance in complex tasks and/or reduce the training time required to achieve the desired performance.

3.3.3 Meta-Learning

For standard meta-RL approaches, the most common practice to build a training set by iteratively sampling from a task distribution until the whole task space is densely covered (dense sampling) [6, 9, 38]. Although dense sampling works well in simulated environments where task can be generated indefinitely, this practice has a high computational cost, and in challenging domains, dense sampling may not be possible.

There have been several efforts in the field of supervised learning to develop strategies that improve meta-learning by leveraging the training set. Task scheduling strategies, for example, have been proposed to change the frequency or order in which tasks are sampled in order to improve the generalization and performance of the meta-learning model [62, 63, 64, 65]. Furthermore, unsupervised meta-learning has been investigated, with methods developed to generate and augment training sets in an unsupervised manner [66, 67]. Similarly, recent research proposes to improve a training task set by generating extra tasks using task interpolation between pairs of randomly selected training tasks [68]. Although these methods enhance meta-learning performance in supervised learning contexts, it is not investigated in RL, where task scheduling and generation may be more challenging due to the nature of the tasks.

In the scope of meta-RL, unsupervised meta-RL methods have been proposed, with the goal of generating sets of training tasks by modifying reward functions [69] or employing information maximization between a latent task variable and the meta-learner's data distribution in domains with pixel observations [70]. An important disadvantage of these methods is that they interfere with the meta training process, making them dependent of the method employed in meta-training.

Although the strategies discussed previously serve to increase meta-learning performance, many of them do so by increasing the number of tasks used for training. Instead of creating more tasks to better the meta-learning training process, we use the opposite approach in this thesis. We investigate and suggest strategies for making the most of a small training set in meta-RL, minimizing computation through effective task selection, therefore reducing the number of tasks on which the meta-RL agent is trained. Furthermore, because our approaches prepare the original training set before meta-training, our suggested methods are agnostic to the method utilized for training.

3.4 Learning Planning Heuristics

Building planning heuristics that enable forward search algorithms to create high-quality plans is a continual focus of AI planning research. With the great success in planning competitions and applications [43, 44, 71, 72] that heuristic search algorithms have shown, we can understand where this interest stems from. The heuristic function, which calculates the cost of reaching the goal from any state, is a key component of these algorithms.

Much of the recent research in heuristic planning has been focused on developing methods that make use of Deep Learning to learn heuristic functions. Most of these methods improve or combine on existing heuristics [23, 25, 26, 28, 73, 74]. All these methods make use of supervised learning to learn a heuristic function. Although the generalization and quick adaptation properties of meta-RL could greatly benefit heuristic planning, there has not been any research on meta-RL in this scope.

For instance, Samadi, Felner, and Schaeffer [25] propose to combine different

heuristics using a neural network. In their work, a neural network is build to learn the cost of reaching a goal given k number of heuristics. For each tasks in the training set, k heuristic values and their precomputed optimal plans are fed into the neural network with the aim of learning a heuristic function which optimally combines the heuristics used for training and improves their individual performance.

Groshev et al. [26] combine convolutional neural networks and graph neural networks to learn heuristics. For training their model they make use of images, obtained from a hand-coded state translation, and precomputed plans obtained with the Fast-Forward planner [43], as input.

Toyer et al. [27] propose ASNet, a weight sharing neural model dedicated to planning with the ability to generalize to different planning problems from the same domain. To learn their heuristic function, during their training process, they make use of heuristic values obtained from LM-CUT [40] as input of their model.

Shen, Trevizan, and Thiébaux [28] propose Hypergraph Networks (HGNs) which generalises Graph Networks [75] to hypergraphs. HGNs learns a heuristic by training on delete-relaxation hypergraphs of a planning problem and state-values pairs $(s, h^*(s))$ obtained from an optimal heuristic function h^*.

Despite the fact these prior methods have shown good performance, one of the disadvantages of supervised learning approaches is that they make use of optimal heuristics and precomputed planning problems to train their models. This standard practice ignores suboptimal states that could provide relevant information during training to the learning model used to build the heuristic, which can harm generalization and performance.

In RL, an agent's exploration is guided by a value function, just as a planner's exploration is guided by a heuristic function. This value function is learnt by

exploring the whole state space, gradually improving during training until it converges to lowest cost from any state to a goal. However, because value functions are task-specific, we would have to learn a new value function for each new planning task provided. This is a shortcoming that can be addressed with meta-RL.

In this thesis, we propose to use meta-RL instead of supervised learning in the context of heuristic planning, and to learn by exploration on a limited number of instances from the planning domain, rather than from precomputed plans. The exploration allows the agent to learn values of sub-optimal states as well, which we hypothesise to provide valuable information to transfer.

3.5 Summary

In this chapter we introduced the most relevant methods in meta-RL; we described each method's relevant properties and learning procedure. Although these methods tackle the meta-learning problem in different ways, there are still challenges to be resolved. One of these is sample efficiency, training meta-RL agents that are able to obtain good performance with a small set of training tasks. Task selection in RL has been widely studied in different fields of RL such as transfer learning and curriculum learning, demonstrating that proper task selection can improve the performance of a RL agent at test time, by learning a better transferable representation of the problem. Despite the benefits of task selection demonstrated by these methods, it has received little attention in the field of meta-RL, where dense sampling of task distributions is standard practice.

Moreover, we introduce approaches that have employed deep learning to learn an heuristic function for classical planning. One common trend of these methods is the use of supervised learning to learn from precomputed plans and build a heuristic

function, which ignores relevant information that suboptimal states could provide to the learning model.

Chapter 4

Information-theoretic Task Selection for Meta-Reinforcement Learning

4.1 Introduction

As previously discussed, the usual approach for building training sets in meta-RL is to dense sample from a task distribution, which is computationally costly and may be infeasible in some challenging domains.

In this chapter, we study meta-RL in a few-task settings and show that when a limited set of training tasks is available for training, not all tasks are necessarily beneficial, and selecting a subset of training tasks may lead to a better performance at test time.

We introduce an Information-Theoretic Task Selection (ITTS) algorithm that we developed, that filters the set of training tasks identifying a subset of tasks that are both *different* enough from one another, and *relevant* to tasks sampled from the target distribution. The method is independent of the meta-learning algorithm used. The goal of ITTS is to build a smaller training set, which can be learned more quickly and results in better performance, a higher return, than the original set.

Task selection is performed before meta-learning and in conjunction with an existing meta-learning algorithm. We identified five domains in the literature that have been used to assess existing meta-RL algorithms, and evaluated ITTS in the same settings. The results show that task-selection improves the performance of two popular meta-RL algorithms (RL^2 [9] and MAML[6]) in all domains. We also introduce a sixth domain as an example of a real-world application on device control for micro grids, and use it to validate our approach in a realistic setting.

4.2 Task Selection for Meta-Reinforcement Learning

ITTS is executed in conjunction with an existing meta-RL algorithm, and therefore falls into the meta-RL framework introduced in Chapter 2. We make two further assumptions: that training tasks $\mathcal{T}$ can be learned to convergence, and therefore their optimal policies $\{\pi_i^*\}_{i=1}^T$ are available; and that the state space of training tasks can be sampled, which will allow us to estimate the difference and relevance as introduced in this section.

ITTS takes as input two sets of tasks sampled from $p(\mathcal{M})$. The first set of T tasks is the training set $\mathcal{T}$ common to all meta-learning algorithms. The second set, $\mathcal{F} = \{m_j\}_{j=1}^K$ of K tasks, such that $m_j \sim p(\mathcal{M})$ and $\mathcal{T} \cap \mathcal{F} = \emptyset$ is the *validation* set. The intuition behind ITTS is that the most useful subset of $\mathcal{T}$ contains tasks that are both *different* enough from one another, to reduce overfitting and increase generalization, and *relevant* for the validation tasks, discarding tasks that may not be beneficial for the the target or test tasks in which we are interested our agent to perform. In the rest of this section we translate this intuition into a heuristic algorithm.

We take an information-theoretic perspective to measure the difference and relevance of training tasks, based on the policies that the agent learns for them. We define the difference between two training tasks m_1 and m_2 as the average KL divergence of the respective policies over the states of the validation tasks:

$$\delta(m_1, m_2) \coloneqq \frac{1}{K} \sum_{m_j \in \mathcal{F}} \frac{1}{|S_j|} \sum_{s \in S_j} \sum_{a \in A} \pi^*_{m_1}(a|s) \log \frac{\pi^*_{m_1}(a|s)}{\pi^*_{m_2}(a|s)}. \tag{4.1}$$

Maximizing a policy's entropy has been demonstrated to be effective at enhancing its generalization [76]. Since we want to compare two different distributions and calculate how different are to each other the natural choice is the relative entropy or KL divergence, which have been previously used to reduce overfitting in meta-RL [8, 47]. Despite employing a measure like the Jensen-Shannon Divergence [77] may appear to be a better alternative due to its symmetric nature, we found out that there was no difference in the method's performance when using it while being more computationally expensive.

To define the relevance of a task m_1 to a task $m_j \in \mathcal{F}$, we consider the optimal policy of m_1, $\pi^*_{m_1}$, and its transfer to m_j. The policy obtained after l episodes of learning in m_j starting from policy $\pi^*_{m_1}$ will be denoted as $\pi^l_{m_1,m_j}$. We define relevance as the expected difference in entropy of the policies before and after learning, over the states of the validation tasks, with respect to the on-policy distribution before learning:

$$\rho_l(m_1, m_j) \coloneqq \mathbb{E}_{s \sim d^{m_j}_{\pi^*_{m_1}}, \pi^l_{m_1,m_j}} \left[H(\pi^l_{m_1,m_j}(a|s)) - H(\pi^*_{m_1}(a|s)) \right]. \tag{4.2}$$

The ITTS algorithm is shown in Algorithm 1. Before execution, n states are sampled uniformly from the tasks in $\mathcal{F}$ and stored in a set of validation states S_v. In practice, for continuous state and action spaces, the states can be sampled

on-policy, using the policy of the validation task from which the samples are being collected. These state samples will be used to estimate δ from Equation 4.1, in place of the sum over all states for all validation tasks. The set of training tasks $\mathcal{T}$, validation tasks $\mathcal{F}$, and sample states S_v are given in input to the ITTS algorithm.

Algorithm 1 Information-Theoretic Task Selection

1: **Input:** $\mathcal{T}$ all available task and $\mathcal{F}$ validation tasks, S_v sample states, ϵ difference threshold, i iterations of initial policy , l learning episodes.
2: **Output:** $\mathcal{C}$ optimal meta training source tasks
3: $\mathcal{C} \leftarrow \{\}$
4: **for** m **in** $\mathcal{T}$ **do**
5: different $\leftarrow true$
6: **for** c **in** $\mathcal{C}$ **do**
7: $\delta_c \leftarrow \frac{1}{n} \sum_{s \in S_v} D_{KL}(\pi^*_m(a|s), \pi^*_c(a|s)) \geq \epsilon$
8: different $\leftarrow$ different $\wedge\, \delta_c$
9: **end for**
10: relevant $\leftarrow$ RelevanceEvaluation$(\pi_m, \mathcal{F}, i, l)$
11: **if** different $\wedge$ relevant **then**
12: $\mathcal{C} \leftarrow \mathcal{C} \cup \{m\}$
13: **end if**
14: **end for**

The subset of selected training tasks $\mathcal{C}$ is initialized with the empty set (line 3). Each task $m \in \mathcal{T}$ is then evaluated for difference from the tasks in $\mathcal{C}$ and relevance with respect to the validation tasks. The algorithm computes an estimate of $\delta(m, c)$ for tasks $m \in \mathcal{T}$ and $c \in C$ and tests whether it is greater than or equal to a parameter ϵ (line 7).

ITTS then proceeds to check for relevance (Algorithm 1 line 10), as shown in Algorithm 2. The agent executes the optimal policy of a training task m in a validation task $f \in \mathcal{F}$ for a number of episodes, generating a set of n traversed states, which are stored in S_e (Algorithm 1 line 7). This set is a sample of the states according to the on-policy distribution, used in estimating the expectation in Equation 4.2. Then the agent learns for l episodes, starting from the transferred policy π^*_m, resulting in a learned policy $\pi^l_{m,f}$ (Algorithm 1 line 8). The entropy of

Algorithm 2 RelevanceEvaluation

1: **Input:** π_m, $\mathcal{F}$, i learning epochs in validation task, l learning episodes
2: **Output:** $isRelevant$
3: $isRelevant \leftarrow False$
4: **for** f **in** $\mathcal{F}$ **do**
5: $\eta_b \leftarrow 0$, $\eta_a \leftarrow 0$
6: **for** 1 **to** i **do**
7: $S_e \leftarrow \text{execute}(\pi^*_m, f)$
8: $\pi^l_{m,f} \leftarrow train(\pi^*_m, f, l)$
9: $\eta_b \leftarrow \eta_b + \frac{1}{n} \sum_{s \in S_e} H_{\pi_m}(s)$ //Equation 4.2
10: $\eta_a \leftarrow \eta_a + \frac{1}{n} \sum_{s \in S_e} H_{\pi^l_{m,f}}(s)$ //Equation 4.2
11: **end for**
12: $\hat{\rho}_l(m, f) \leftarrow \eta_a - \eta_b$
13: **if** $\hat{\rho}_l(m, f) \leq 0$ **then**
14: **return** true
15: **end if**
16: **end for**
17: **return** false

both the transferred policy and the learned policy is evaluated on the set of states S_e (Algorithm 1 line 9 and 10) by:

$$H_\pi(s) = -\sum_a \pi(a|s) \log \pi(a|s).$$

The learning process is repeated i times, sampling i policies which are used to estimate the expectation with respect to the learned policy in Equation 4.2. If learning produced on average a policy of lower entropy (therefore an information gain) we consider the training task that provided the transfer policy as *relevant* (line 14 in Algorithm 1).

If a task $m \in \mathcal{T}$ is different from all the currently selected tasks in $\mathcal{C}$ and relevant for at least a validation task in $\mathcal{F}$ it is added to $\mathcal{C}$ (line 12 in Algorithm 2).

4.3 Experimental Evaluation

The main aim of this evaluation is twofold: to demonstrate that task selection is indeed beneficial for meta-RL, and show that applying ITTS to existing meta-RL algorithms consistently results in better performance on test tasks. We also analyze the effect of the main algorithm parameter, ϵ, on the results, and perform an ablation study to show that both difference and relevance contribute to the performance of ITTS.

We used RL^2 [9] and MAML[6] to execute in conjunction with ITTS, each one being a popular algorithm from the category of context-based and gradient-based method respectively. However, instead of following the standard meta-training process in which an agent has continuous access to a dense distribution of training tasks, a fixed set of tasks is sampled for training. We surveyed the literature to identify domains used to demonstrate meta-RL algorithms, with the following two characteristics: available source code, and programmatic access to the task distribution $p(\mathcal{M})$ to generate further tasks. We identified five such domains: CartPole [31, 34, 45], MiniGrid [78], Locomotion(Cheetah) [6], Locomotion(Ant) [6], and KrazyWorld [35]. CartPole and MiniGrid are less computationally demanding, and have been used for the parameter and ablation studies. The other domains are more complex control problems and have been used, in addition to the previous ones, to evaluate the effectiveness of ITTS in the same setting as either RL^2 or MAML[6]. We also introduce a sixth domain, MGEnv, as a representative of a realistic application in micro-grid control.

In the rest of this section, we first introduce the domains, then evaluate the effect of the threshold ϵ, show the results of the ablation study, and lastly we apply ITTS to RL^2 and MAML in their respective domains.

4.3.1 Domains

In this section, we describe the experimental domains as well as the technique used to produce the tasks common to train, validation, and test sets. We limited the number of training tasks in each domain so that the generation and training until convergence repeated for 5 times would not exceed 72 hours of computation on an 8-core machine at 1.8GHz and 32GB of RAM. As a consequence, simple domains like CartPole have more training tasks than more complex domains, like MGEnv. In every domain we used $K = 5$ validation tasks.

4.3.1.1 CartPole

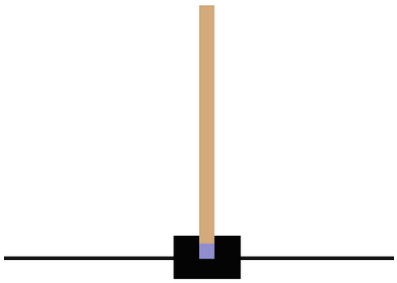


Figure 4.1: Representation of the CartPole domain

CartPole, from OpenAI gym [79], is a classic control task in which a pole is attached by an unactuated joint to a cart which moves along a track. The goal is to prevent the pole from falling over. The cart is controlled by applying a positive or negative force, and a reward of $+1$ is obtained for each time step the pole remains upright. For this domain, 60 different training tasks were created by sampling the parameters of the environment from a uniform distribution. The parameters are: the length and the mass of the pole, the mass of the cart, the intensity of gravity, the

quantity of force applied to the cart in an action, and the degrees from the vertical position in which the pole is considered as fallen.

4.3.1.2 MiniGrid

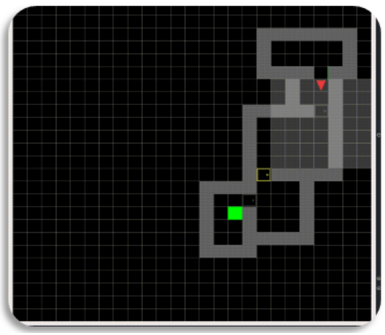


Figure 4.2: Representation of the MiniGrid domain

MiniGrid is an open-source grid world package proposed as an RL benchmark [80]. In the grid world used for our experiments, the agent navigates a maze with the purpose of reaching a goal state. The agent receives a reward between 0 and 1 when it reaches the objective, proportional to the number of time steps taken to reach it. For this domain, 34 training tasks were created by randomly changing the shape of the maze, the initial position, and the goal.

4.3.1.3 Locomotion

The locomotion domains are borrowed from the MAML paper[6]. In these domains two simulated robots, a planar cheetah and a 3D quadruped ("ant"), are required to run at a particular velocity. The reward is the negative absolute value between the current velocity of the robot and the target velocity. This target velocity is chosen uniformly at random between 0.0 and 2.0 for the cheetah and 0.0 and 3.0 for the ant. 40 training tasks were created for each domain. The target velocity on test tasks was chosen randomly and uniformly.

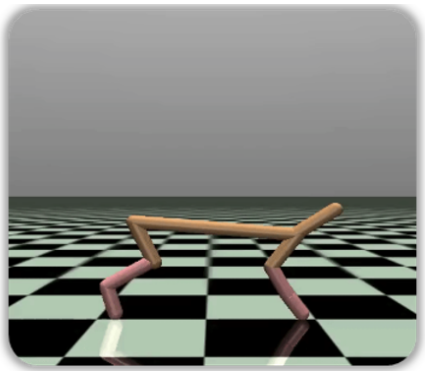


Figure 4.3: Representation of the Cheetah domain.

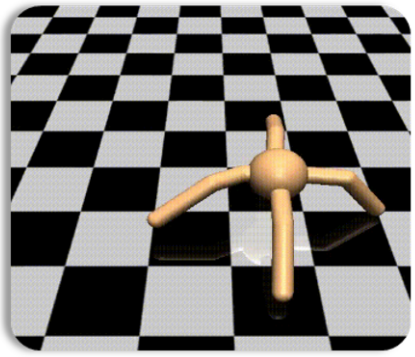


Figure 4.4: Representation of the Ant domain.

4.3.1.4 KrazyWorld

KrazyWorld [35] is a grid world domain proposed to test adaptation of meta-RL agents. In this domain the agent explores the environment looking for goal squares which provide a reward of +1 while evading obstacle squares which delay or kill the agent. A colour is assigned to each type of square for the agent to able to identify them. For each generated task, the position of the agent, the position of the goal and obstacle squares, and the colour assigned to them is randomly chosen. For this domain, 40 different tasks were used to build the training set.

4.3.1.5 MGEnv

The purpose of this domain is to validate our approach in a real-world scenario. We aim to demonstrate that our method is suitable for this kind of tasks, where the parameters of the environment can be highly variable and suboptimal performance is highly costly.

MGEnv is a simulated micro-grid domain created out of real data from the PecanStreet Inc. database. The domain includes historical data about energy consumption and solar energy generation of different buildings in the USA. Tasks in this domain are

defined as a combination of three elements: the model of the electrical device to optimize, the user's schedule, specifying if the device must be run in given day, and the building data, containing the energy generation and consumption of the given building. The devices used for the experiments behave as time-shifting loads, meaning that, once started, they run for several time steps and cannot be interrupted before the end of their routine. The goal of the agent is to find the best time slot to run the given device, optimizing direct use of the generated energy, while following the user's device schedule. The agent receives a positive reward when it uses energy directly from the building generation and a negative reward when consuming energy from external sources. The reward amount depends on the quantity of energy used from both sources. A reward of -200 is obtained if the device is not run accordingly to the user schedule. The simulator replays real data of energy generation and consumption of the building (other than of the simulated controlled device), and is therefore as close as possible to running the device in the real building at that time.

For this domain 24 tasks were used to build the set of training tasks. These tasks represented 24 buildings with different energy consumption, energy generation, schedule, and devices' energy consumption and running time. Test tasks were selected by choosing buildings that had an energy generation and consumption levels at most 80% similar to the ones presented in the validation set, while the devices' properties were different for all the tasks in both sets.

4.3.2 Results

A full experimental run proceeds as follows: tasks are selected according to ITTS, the meta-learning algorithm is run to obtain a meta-policy, and the policy is evaluated in the test tasks. RL^2 [7, 9] was used as the meta-RL algorithm in KrazyWorld

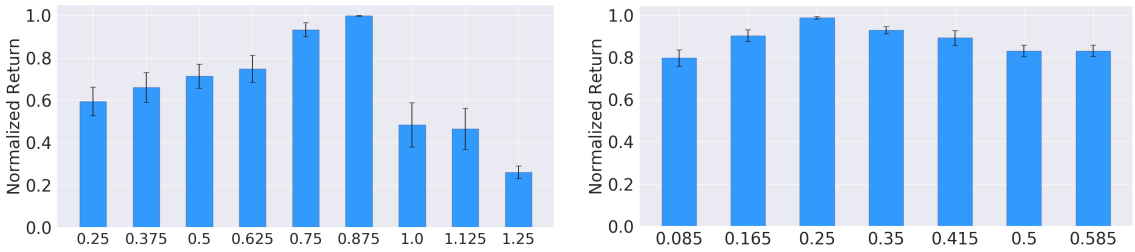
and MGEnv, in addition to the parameter and ablation studies. MAML was used in Ant and Cheetah. In both cases, we aimed at reproducing the results of the papers where the domain is presented. For all domains we show the average performance over all the runs. This accounts for slight discrepancies with respect to some of the original papers when they show the best model, rather than the average. In all the plots the error bars are 95% confidence intervals.

4.3.3 Parameter Evaluation

We start by studying the effect of the main threshold parameter, ϵ, of the algorithm in the two least computationally expensive domains, which allows us to repeat the learning process 10 times over 5 test tasks, with a total of 50 different test tasks. The threshold determines when a task is considered different enough from another task, that is, their difference measured as in Equation 4.1 is greater than or equal to ϵ. The results are shown in Figure 4.5a and 4.5b for CartPole and MiniGrid respectively. The plots show that the optimal value is domain dependent, but rather easy to determine, since the return in the test tasks with respect to the parameter ϵ is convex. A parameter of $\epsilon = 0$ corresponds to ignoring task difference, and considering only task relevance. The optimal parameters established this way have been used in the rest of the experiments. For better comparison between domains, the ϵ values shown in the figures were normalized by the number of actions in each domain. Returns are also normalized between 0 and 1.

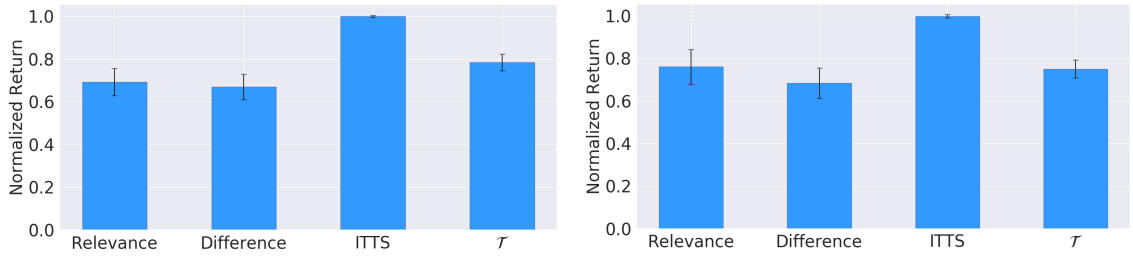
4.3.4 Ablation Study

In this experiment we aim at establishing that both difference and relevance indeed contribute to the transfer. We evaluated the agent using only relevance, only difference, and both (ITTS). The results are shown in Figure 4.6a for CartPole



(a) CartPole (b) MiniGrid

Figure 4.5: Results of parameter evaluation. Values shown on the x-axis represent the normalized values used for ϵ while the y-axis shows normalized returns



(a) CartPole (b) MiniGrid

Figure 4.6: Ablation study. The plot shows average performance on test tasks of the agents trained using only relevance, only difference, and both (ITTS). "$\mathcal{T}$" is the performance obtained using all training tasks, without task selection. The error bars are 95% confidence intervals.

and Figure 4.6b for MiniGrid. The results were obtained, similarly to the previous experiment, by evaluating the agent in 5 test tasks for 10 runs. In both domains, neither difference nor relevance alone can obtain the same performance than when used in combination (ITTS).

4.3.5 Transfer Results

Lastly, we evaluate the effect of ITTS on existing meta-RL algorithms on all six domains. In addition to the meta-RL algorithm with and without ITTS (using all the tasks in $\mathcal{T}$), we also consider as a baseline a random subset of the training tasks, and using the validation set $\mathcal{F}$ as the training set (without ITTS). The results are shown for CartPole in Figure 4.7, for MiniGrid in Figure 4.8, for Ant in Figure 4.9,

CartPole	MiniGrid	KrazyWorld	Ant	Cheetah	MGEnv
14	12	24	20	16	9

Table 4.1: Number of tasks selected by ITTS.

for Cheetah in Figure 4.10, for KrazyWorld in Figure 4.11 and for MGEnv in Figure 4.12. The agents were evaluated over 5 test tasks per run, with results averaged over 5 runs (25 different test tasks in total). The results of random selection are averaged over 4 random subsets of random size. In these plots as well, the shaded area is the 95% confidence interval. The number of tasks selected by ITTS are shown in Table 4.1. The results show the consistent performance improvement achieved by ITTS over the baselines. Interestingly, the set of training tasks is not always significantly better than the set of validation tasks (when used for training), despite the latter is much smaller. This also confirms that more tasks do not necessarily improve the final performance, and appropriate tasks must be selected.

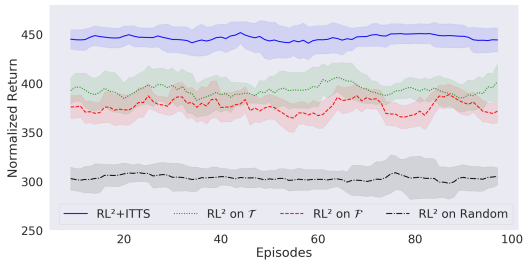


Figure 4.7: Results on CartPole domain.

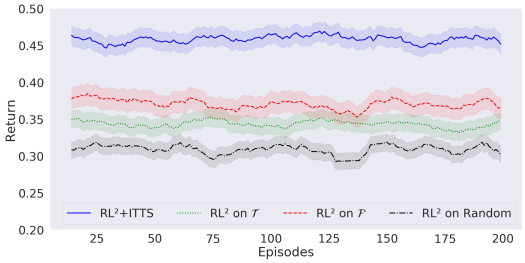


Figure 4.8: Results on MiniGrid domain.

4.4 Summary

We introduced an Information-Theoretic Task Selection algorithm for meta-RL, with the goal of improving the performance of an agent in unseen test tasks. We experimentally showed that an agent trained using a subset of tasks selected by

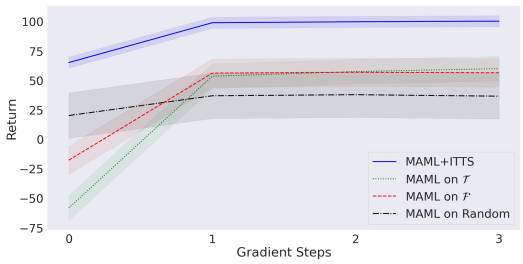


Figure 4.9: Results on Ant domain. 20 rollouts per gradient were used.

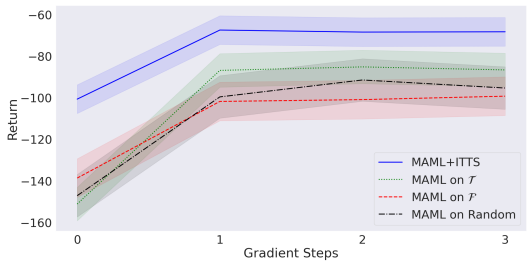


Figure 4.10: Results on Cheetah domain. 20 rollouts per gradient were used.

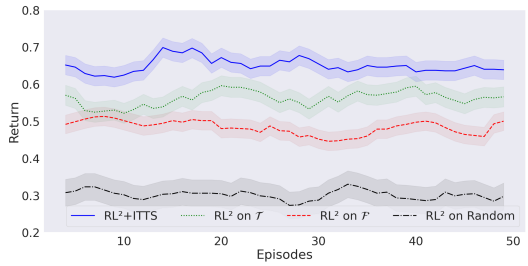


Figure 4.11: Results on KrazyWorld domain

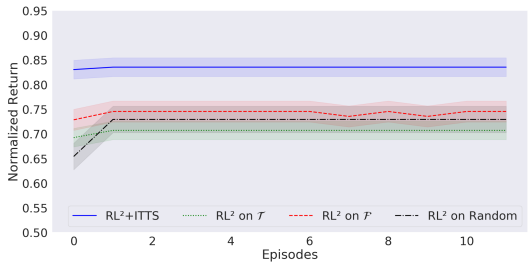


Figure 4.12: Results on MGEnv domain. Returns are normalized

our algorithm outperforms agents that were trained using all the available training tasks or random subsets of these tasks. We also showed how both *difference* and *relevance* are important in ITTS for boosting the performance of the agent.

The presented results unequivocally demonstrate the potential of task selection in meta-RL, which has been so far overlooked. However, for our method to work, the policies of all the tasks in the original training set must be learnt, since they are required to calculate the different metrics used to filter tasks. Moreover, the heuristic nature of our algorithm raises the question of a theoretical understanding of the role of tasks on generalization in meta-RL. In the next chapter, we gain insight into the behaviors learnt with meta-RL and suggest a novel task selection method that addresses some of the shortcomings of ITTS.

Chapter 5

Few-Task Meta-Reinforcement Learning

5.1 Introduction

We have shown how task selection can improve the performance of meta-RL when agents have access to a limited set of training tasks. ITTS allows us to filter a set of tasks and build a training set that give us better performance than training with all the available tasks.

Although ITTS has shown great advantages, it has some shortcomings. ITTS requires access to the optimal policies of all the available tasks. Depending on the domain, learning the optimal polices could have a large computational cost. Moreover, ITTS requires a set of validation tasks and a difference threshold for its task selection process, which when not defined correctly, can harm the task selection process.

In this chapter, we build insight into the meta-RL learning procedure in a simple scenario, showing the behaviour learnt by a meta-RL method, and propose a new task selection method which we call Few Tasks Meta-RL (FETA), which similarly to ITTS, aims to build a set of training tasks that improves the performance of

meta-RL agents in scenarios in which a small set of training tasks is available. FETA selects tasks by leveraging the policy transfer properties of RL to discard potentially redundant data, improving the performance and sample efficiency of the meta-learned policy. The FETA sequential filtering process does not require the optimal policies of all the available tasks $\mathcal{T}$, but only the ones in the set of selected training tasks $\mathcal{C}$. FETA achieves similar or better performance than ITTS when training using MAML as meta-RL training algorithm, while having a much smaller computational cost.

5.2 Meta-Policy Analysis

To build insight and further understand the convergence properties of meta-learnt policies, we investigate policies obtained with FOMAML, a gradient-based meta-learning approach, in a simple scenario where the policy behavior can be seen and "dense" sampling over all possible tasks is feasible.

The experimental domain is a 6 by 6 Gridworld with the initial state set at the top left corner and the goal spawning in 10 distinct spots, 4 in the 4 cells at the top right corner, 4 in the 4 cells at the bottom right corner, and 2 goals in the 2 cells at the bottom left corner as shown in Figure 5.1. In this domain, the agent's sole purpose is to navigate across the grid to reach the goal. The 4 possible actions are moving south, moving north, moving east and moving west. Ten different tasks were used for this experiment, each task with a different goal position.

Using tabular value functions, we trained a meta-RL agent using a training set $\mathcal{T}$ containing all ten tasks, to obtain an optimal policy $\pi_{\mathcal{T}}$, which contains information from all possible tasks of the domain. Figure 5.2 shows the heat map of the policy $\pi_{\mathcal{T}}$ learnt by the agent. Each cell in the map shows the probability of taking that

action in that state. We can see that agent discovers certain paths, which seem to favour specific goals, partitioning the state space in three different zones. Since the initial state is constant, the state space partitioning is particular clean in this case. From the learned behaviour displayed in the figure, we could identify individual tasks, distributed along the grid, that could give the agent enough information to cover the three different zones and create similar paths, allowing us to train an agent which learns a similar policy using a fewer tasks.

We now place the same experiment in our few-task framework. We must be able to learn equivalent behavior with fewer tasks in order to demonstrate the validity of the framework. From figure 5.2 one can easily identify and handpick tasks whose combined policies could cover the whole task space. With that in mind, we built a training set $\mathcal{C}$ containing 3 handpicked tasks, 1 task per corner, excluding the top left one.

To better illustrate the information that each task in $\mathcal{C}$ is providing, we calculated the KL-Divergence between each optimal policy $\{\pi_j^*\}_{j=1}^{\mathcal{C}}$ and the meta-policy $\pi_{\mathcal{T}}$, state by state. Figure 5.3 shows the results obtained from this evaluation. For each state, the corresponding color of the optimal policy π_j^* with the lowest KL-Divergence towards $\pi_{\mathcal{T}}$ in that state, was plotted. We can see that these 3 tasks in $\mathcal{C}$ are enough to partition the policy in the three different zones, that we identified in Figure 5.2, in which the agent expects that goals might be allocated, containing enough information to teach the agent the desired behaviour.

Furthermore, we train a meta-RL agent on $\mathcal{C}$ to obtain policy $\pi_{\mathcal{C}}$ and evaluate how different it is from $\pi_{\mathcal{T}}$. To carry out this evaluation, we calculate the KL-Divergence between $\pi_{\mathcal{C}}$ and $\pi_{\mathcal{T}}$, state by state. Figure 5.5 shows the results of the evaluation while Figure 5.4 shows the heat map of the policy $\pi_{\mathcal{C}}$ learnt by the agent. For comparison, the KL-Divergence between the two opposite deterministic policies,

meaning that each policy guides the agent to a completely different direction, is 6.5. We can see that although $\pi_{\mathcal{C}}$ was trained with only 3 tasks compared to the 10 that were used to train $\pi_{\mathcal{T}}$, both policies are closely related, and even when some information is lost due to the removal of the tasks, the agent still learns a proper exploration strategy which allows it to achieve quick adaptation.

Finally, we evaluated the success rate of both policies in all the 10 original tasks, testing policies $\pi_{\mathcal{T}}$ and $\pi_{\mathcal{C}}$ for 100 episodes per task. In this evaluation, both policies were able to obtain a success rate of 100%, meaning that both agents were able to reach the goal in all the testing episodes.

We may conclude from the results of these evaluations that even when some tasks are eliminated from the training set, provided the remaining tasks are appropriately selected and distributed, the meta-RL agent can learn a policy comparable to the optimal with fewer tasks.

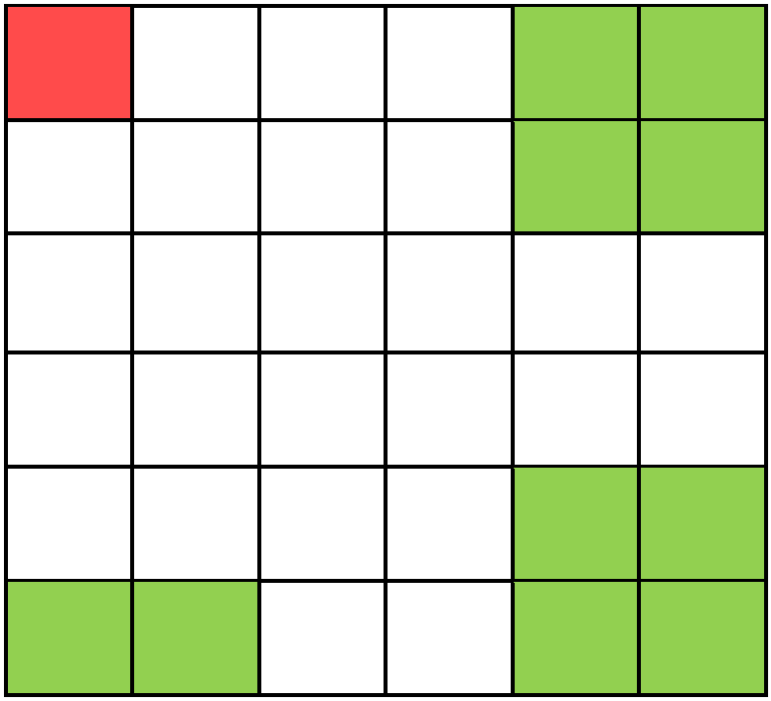


Figure 5.1: Representation of the Gridworld domain. Green squares represent the goals while the red square the initial position of agent. One goal per task.

Figure 5.2: Map of the probability distribution over the actions of $\pi_{\mathcal{T}}$ on the Gridworld domain. Each cell group represents a state (position) on the grid, while each number in a cell the probability of taking the action in the plotted direction.

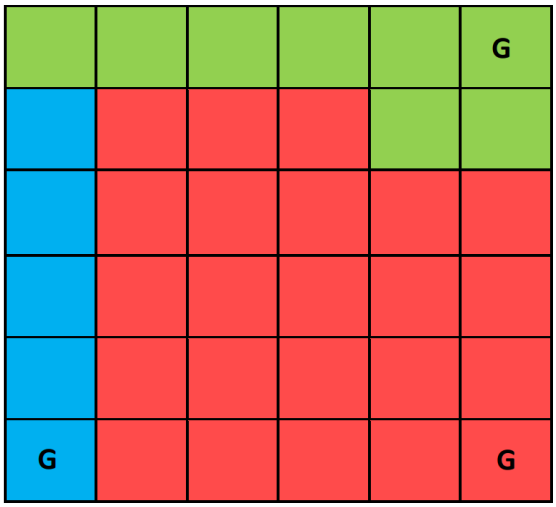


Figure 5.3: Map of similarity between optimal policies of handpicked tasks $\mathcal{C}$ and the meta-policy trained on all available tasks $\mathcal{T}$. Each handpicked task is represented by a different color. The color plotted shows which optimal policy is closest to the meta-policy in that sate. G represents the position of a goal.

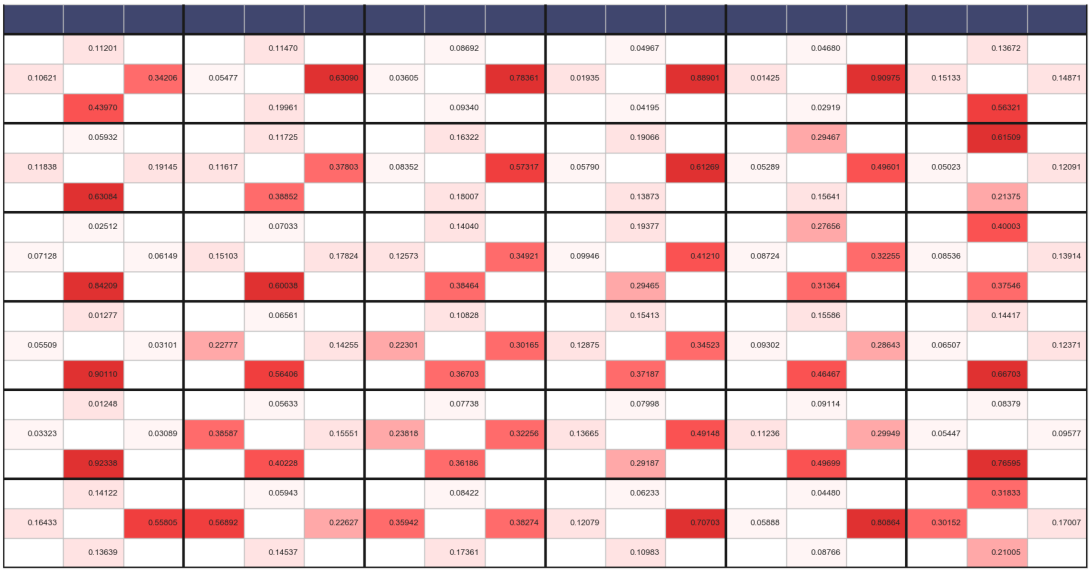


Figure 5.4: Map of the probability distribution over the actions of $\pi_{\mathcal{C}}$ on the Gridworld domain. Each cell group represents a state (position) on the grid, while each number in a cell the probability of taking the action in the plotted direction.

0.092	0.087	0.113	0.051	0.920	0.206
0.118	0.258	0.193	0.309	0.068	0.571
0.265	0.389	0.268	0.342	0.790	0.176
0.059	0.220	0.255	0.376	0.356	0.129
0.006	0.446	0.222	0.349	0.120	0.762
0.288	1.00	0.168	0.132	0.416	0.526

Figure 5.5: Map of the KL-Divergence between policies $\pi_{\mathcal{T}}$ and $\pi_{\mathcal{C}}$.

5.3 Filtering Tasks in Meta-RL

Handpicking tasks whose policies could cover the whole task space for a domain such as the Gridworld presented in Section 5.2 is not difficult; nevertheless, most domains worth addressing are not as simple to visualize and analyze. For that reason, we propose an automated approach to automatically identify those tasks whose combined policies are close to the meta-policy on all tasks and discard tasks that provide redundant information, which we call Few Tasks Meta-RL (FETA).

We make two assumptions when using FETA: the optimal policies of the tasks in the training set $\mathcal{T}$ can be learnt, and a binary function $\mathcal{R} : \Pi_{\mathcal{T}} \times \mathcal{T} \rightarrow \{0, 1\}$, where $\Pi_{\mathcal{T}}$ is a set of all possible policies of the tasks in $\mathcal{T}$, which measures successful transfer, can be defined. The intuition behind this function comes from meta-learning itself. $\mathcal{R}$ takes as input a policy π_i and a target task m; transfer is considered successful if policy π_i is able to adapt to the new task m after a few training steps, implying that the agent was able to achieve a goal or attain a particular cumulative reward. The function outputs a *true* flag if transfer was successful and *false* otherwise. This function $\mathcal{R}$ can bee seen as a measure of difference between the optimal policies of two tasks, which differently from the KL-Divergence we used on ITTS, takes into account the learning algorithm since it is used during the transfer process.

FETA takes as input a set of training tasks $\mathcal{T}$ and a successful transfer function $\mathcal{R}$ and outputs a set of selected training tasks $\mathcal{C}$. The principle behind FETA comes from the the transfer properties of RL. If the optimal policy of a task $m_i \in \mathcal{T}$ is transferred to a different task $m_j \in \mathcal{T}$ and is able to complete it successfully in a few adaptation steps, both tasks can be considered similar; thus, task m_j may be deemed redundant and does not provide new information to the meta-RL

agent, and as shown in chapter 4 can potentially degrade the meta-RL agent's performance.

Algorithm 3 Task Filtering

1: **Input:** $\mathcal{T}$ available training, $\mathcal{R}$ transfer function, N_e adaptation episodes.
2: **Output:** $\mathcal{C}$ optimal meta training tasks
3: $\mathcal{C} \leftarrow \{\}$
4: **for** m **in** $\mathcal{T}$ **do**
5: similar $\leftarrow false$
6: **for** c **in** $\mathcal{C}$ **do**
7: similar$\leftarrow$ similar **or** $\mathcal{R}$ (π_c^*,m,l)
8: **end for**
9: **if not** similar **then**
10: $\mathcal{C} \leftarrow \mathcal{C} \cup \{m\}$
11: learn π_m^*
12: **end if**
13: **end for**

FETA's task selection process shown in Algorithm 3, builds a training set $\mathcal{C} \subseteq \mathcal{T}$ by filtering tasks that may be redundant. FETA sequentially samples tasks from $\mathcal{T}$ and evaluates the sampled task against the tasks in $\mathcal{C}$ that have been already selected. When $\mathcal{C}$ is empty the first task $m \in \mathcal{T}$ is added to set $\mathcal{C}$ and its optimal policy π_m^* is learnt. From there, a new task $m \in \mathcal{T}$ is sampled and evaluated against all tasks in $\mathcal{C}$(line 4). The evaluation is carried out by function $\mathcal{R}$, transferring the optimal policies $\{\pi_j^*\}_{j=1}^{\mathcal{C}}$ of the all tasks in $\mathcal{C}$ to the new sampled task m. If any of the policies in $\mathcal{C}$ is successfully transferred to task m after a N_e adaptation episodes (line 7), m is discarded and a new task is sampled from $\mathcal{T}$. However, if none of the tasks $\{\pi_j^*\}_{j=1}^{\mathcal{C}}$ successfully transfer to m, m is added to $\mathcal{C}$ and its optimal policy $\{\pi_m^*\}$ is learnt (line 9). The process is completed once all the tasks in $\mathcal{T}$ have been evaluated or a desired number of tasks in $\mathcal{C}$ has been reached.

5.4 Experimental Evaluation

In this experimental evaluation, we seek to answer four key questions: (1) How does FETA compare to ITTS? (2) Being a sequential process, does the order of the tasks impact the method's outcome? (3) Does FETA choose the best selection of tasks from all available tasks? (4) How much does FETa cost computationally as compared to utilizing ITTS or training with all available tasks?

For the evaluations, we use MAML [6] as meta-learning method to run alongside FETA. We utilize the same fixed set of available tasks and test tasks from domains that were used in the ITTS experimental evaluation (section 4.3). In all the plots the shaded area are 95% confidence intervals. Like in ITTS for all the experiments we show the average performance over their respective runs. For all our experiments, 10 adaptation steps were used to evaluate successful transfer with function $\mathcal{R}$ $(l = 10)$.

In the rest of the section, we answer the previously stated questions.

5.5 Transfer Results

We start by evaluating the performance of FETA against two baselines: ITTS and MAML training in all the available tasks (MAML on $\mathcal{T}$). All the trained agents were evaluated over 5 tests tasks per run, with results averaged over 5 runs. We used the same order of the original training tasks used in the ITTS experimental evaluation. The results for Cheetah are shown in Figure 5.6, for Ant in Figure 5.7, for MGEnv in Figure 5.8, for KrazyWorld in Figure 5.9. The number of tasks selected by FETA in each domain are shown in Table 5.1. From the results it can be seen that FETA's outperforms MAML on $\mathcal{T}$ and matches or outperforms

KrazyWorld	Ant	Cheetah	MGEnv
21	22	21	10

Table 5.1: Number of tasks selected by FETA.

ITTS in all domains. Even though FETA decreases the computational costs of task selection when compared to ITTS, by reducing the number of policies that are required to be learnt, it still produces a high-quality training set.

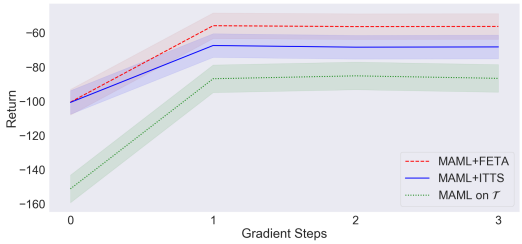


Figure 5.6: Results on Cheetah.

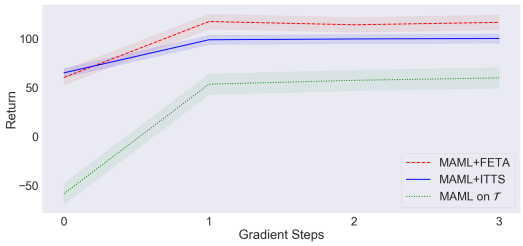


Figure 5.7: Results on Ant domain.

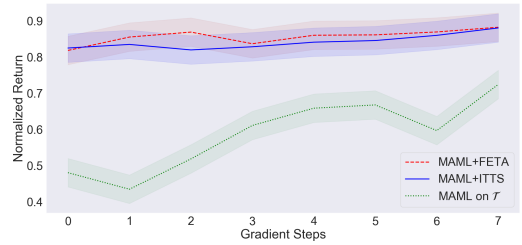


Figure 5.8: Results on MGEnv. Returns are normalized.

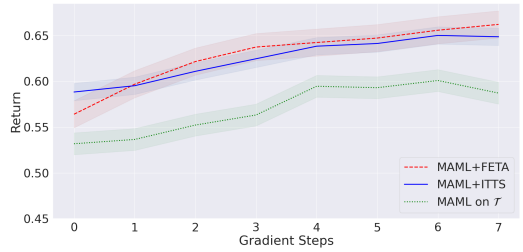


Figure 5.9: Results on Krazyworld domain.

5.6 Task Selection Sequence

A property of a sequential method like FETA is that the order in which the tasks are fed to the algorithm affects the output. Once a task m_i is added to the set of selected tasks $\mathcal{C}$, any subsequent task that is similar to m_i will be discarded. When the order in which tasks $\mathcal{T}$ are fed to the algorithm changes, the output $\mathcal{C}$ may also vary.

To evaluate the impact that task ordering might have over the FETA task selection process, we trained a series of MAML agents on many different set of selected tasks $\mathcal{C}$. Iteratively we randomly changed the order in which tasks $\mathcal{T}$ were fed to FETA, and a series of unique sets $\mathcal{C}$ were obtained. An agent was trained with each unique set and evaluated in the same set of test tasks.

For this evaluation, we used Gridworld, which we introduced in Section 5.2, and Ant as experimental domains. For Gridworld, we ran FETA over 100 unique orders of $\mathcal{T}$, resulting in 18 unique agents. For Ant, we limited the number of runs so that task selection and meta-training would not would not exceed 960 hours of computation time, which led to 20 unique agents. Figure 5.10 for Gridworld and Figure 5.11 for Ant show the results obtained. Despite the fact that each agent was trained on a unique set of tasks, the performance of all agents in both domains is closely related, with higher variance presented in the Ant domain but with an average performance similar to the one obtained in the experiments in Section 5.5. Although in different domains order might affect FETA's performance, in the results obtained in this evaluation, task ordering did not had any observable impact on its performance.

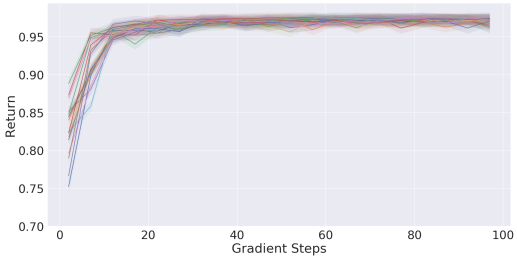


Figure 5.10: Task selection changing order of tasks $\mathcal{T}$ on the Gridworld domain. Each line represents the performance of an agent trained on unique set of tasks selected by FETA.

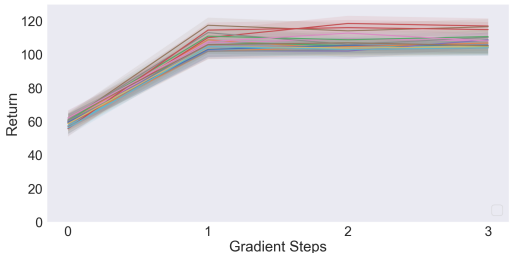


Figure 5.11: Task selection changing order of tasks $\mathcal{T}$ on the Ant domain. Each line represents the performance of an agent trained on unique set of tasks selected by FETA.

5.7 Optimal Task Selection

An important question with a task selection method such as FETA is whether the method actually selects the best set of tasks from all available task combinations. To answer this question and prove global optimality we could use an exhaustive approach, enumerating all possible unique set of tasks and evaluate all of them against FETA. However, the time and resources cost to carry out such experiment would be prohibitively expensive; therefore we take inspiration from local optimization and adapt it to our combinatorial problem, comparing a set of tasks selected by FETA against its "neighbour" sets. We take a set of selected tasks $\mathcal{C}$ and remove/add tasks to it, creating a series of different training sets, to helps us evaluate the local optimality of FETA.

As in the previous section, Gridworld and Ant were used. For Gridworld the set $\mathcal{C}$ selected by FETA has three tasks. We added or removed one task at a time, creating 10 unique training sets from all possible combinations of length 2 and 4 starting from $\mathcal{C}$.

For Ant, the set $\mathcal{C}$ contains 22 tasks. We added or removed 5 tasks at a time, creating unique training sets of length 17 and 27 starting from $\mathcal{C}$. We limited the number of runs to 960 hours of computation time as in the previous experiment, obtaining 30 different training sets, 15 of length $|\mathcal{C}| + 5$ and 15 of length $|\mathcal{C}| - 5$.

The results of the experiment are shown in Figure 5.12 for Gridworld and Figure 5.13 for Ant. FETA- refers to the agents trained with the resultant set of tasks produced by removing tasks from $\mathcal{C}$ and FETA+ refers to the agents trained with the sets obtained by adding tasks to $\mathcal{C}$. The displayed results are the average of all the trained agents for each category. We can see that when tasks are removed from $\mathcal{C}$ the agent's performance deteriorates, but adding tasks to $\mathcal{C}$ does not improve its

performance.

The results obtained from this evaluation show that FETA is able to achieve local optimality, creating sets of tasks that are minimal, obtaining good performance with the least possible number of tasks. Although this evaluation was conducted in two domains, we expect that this results can be replicated in many different domains if the successful transfer function $\mathcal{R}$ is properly defined.

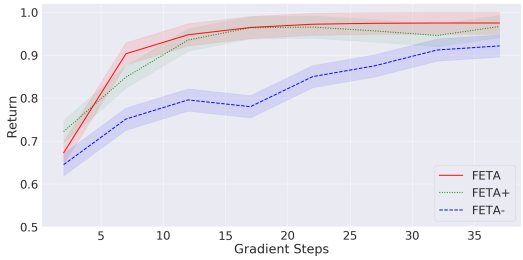


Figure 5.12: Results of FETA's local optimality evaluation on the Gridworld domain.

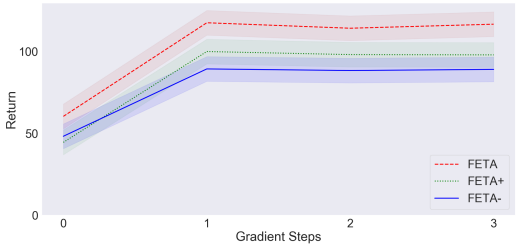


Figure 5.13: Results of FETA's local optimality evaluation on the Ant domain.

5.8 Computational Costs

Lastly, is important to assess how much using FETA costs in comparison to ITTS and the standard training approach of using all the available training tasks. To carry out this evaluation, we measured the wall-clock time required to complete one full loop for each approach. For ITTS, this means learning the optimal policies of all available tasks, filtering tasks and meta-training a model with the selected tasks, whereas for FETA, it entails learning the optimal policies of the tasks that are selected, doing the evaluations to filter tasks and meta-training a model with the output set of selected tasks. We employ two experimental domains, KrazyWorld and Ant, one simple and one complex domain, to analyze how much the costs differ

with different task complexity. Figure 5.14 shows the results for KrazyWorld while 5.15 shows the results for Ant.

From the results, we can see that FETA is the more efficient of the two task selection approaches, requiring about half the time that ITTS does in the Ant domain. However, although both methods are more efficient than training with all tasks in KrazyWorld, in a complex domain such as Ant, the more efficient method, FETA, requires more than double the time. Complex tasks require a considerable amount of time to learn their optimal policies, and since we require to learn the optimal policy of the tasks that are being selected by FETA, each new selected tasks adds up to the costs of the task selection process. On the other hard, in a simpler domain such as KrazyWorld, the greatest cost comes from learning the meta-policy, since learning the optimal policy of each individual task is not hard. Although FETA has a lower computational cost than training with all available tasks in a simple domain, in a more difficult domain such as Ant, FETA's cost is significantly higher, which can be a clear disadvantage if we do not have access to the optimal policies ahead of time and the main interest is to reduce computation costs rather than improving the final performance of the meta-RL agent.

5.9 Summary

We investigated the learned behaviours of a gradient-based meta-RL method and have shown how certain tasks may offer redundant information to the learning model, demonstrating that not all tasks are required for a meta-RL agent to acquire the intended behavior. Moreover, we introduced FETA, a task selection method for meta-RL, that takes advantage of the policy transfer properties of RL to improve the performance of meta-RL agents. We experimentally showed that

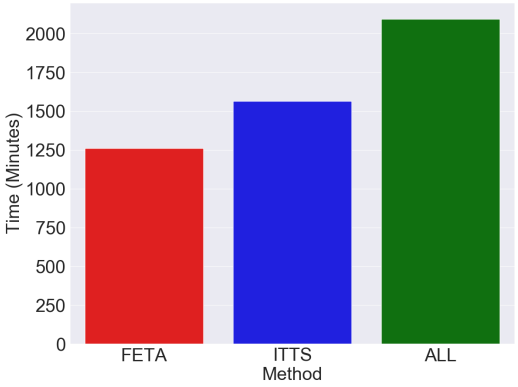


Figure 5.14: Results of the wall-clock time required for one full loop of FETA, ITTS and training with all available tasks (ALL) on the Krazy-World domain. Y axis show the time required in minutes

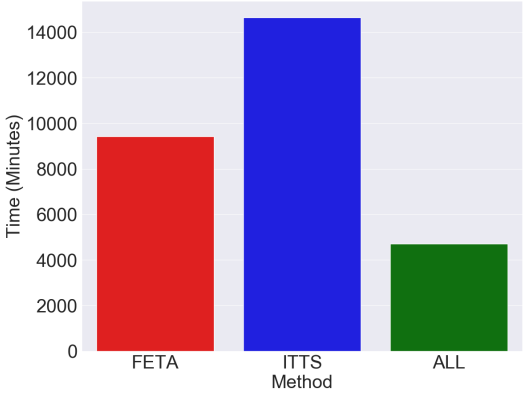


Figure 5.15: Results of the wall-clock time required for one full loop of FETA, ITTS and training with all available tasks (ALL) on the Ant domain. Y axis show the time required in minutes.

FETA outperforms agents that were trained using all the available training tasks, and is able to perform equally or better than ITTS while being more cost efficient. We investigated the effect that task order has on FETA's performance as well as the optimality of FETA's task selection process in a local setting and show that the generated sets are minimal. Finally, we evaluated the computational cost of FETA, showing that the FETA cost is highly related the complexity of the tasks in hand.

Chapter 6

Meta-Reinforcement Learning for Heuristic Planning

6.1 Introduction

In previous chapters we introduced task selection methods and showed the positive impact they have over meta-RL performance. One application where we aim to achieve great performance with a low training cost and using a small pool of training tasks is heuristic planning.

Developing planning heuristics that allow forward search algorithms to generate high quality plans is a constant focus in AI planning research. As discussed in Chapter 2, a central component of these planning algorithms is the heuristic function, which estimates the cost of reaching the goal from any state. This estimate guides the planner to low cost states discarding unpromising regions of the state space. When using Deep Learning to learn a heuristic function, the goal is to obtain a heuristic that expands the fewest possible number of states to find a solution and is as sample efficient as feasible during training.

In this chapter, we define the process to learn heuristic functions for a given planning domain through meta-reinforcement learning and ITTS, learning a heuristic

function that is able to generalize across different instances of a planning domain. Task selection through ITTS allows us to learn a good heuristic using a limited set of training tasks, reducing the training costs, an important feature in planning. We show that the learned heuristic generalizes effectively in a collection of domains from the International Planning Competition (IPC) and the FF Domain Collection.

6.2 Learning Planning Heuristics

We take advantage of the generalization properties of RL^2 [7, 9], a context-based meta-RL method, and apply it to classical planning. Since RL^2 does not require learning during deployment, it is better suited for planning than gradient-based methods. Our aim is to learn a heuristic function that gets as close as possible to an optimal value function. The methodology is summarized in algorithm 4 and consists of the following steps.

6.2.1 Learning Problem Definition

A numerical representation of the state space and the reward function must be defined, so that they can be used in the input and objective function of the neural networks encoding the policy and the value function. We define a numerical vector $s_i = \langle s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(n)} \rangle$, representing the state at time step i, from a domain description in PDDL 3.1. This vector is created by concatenating Boolean predicates and numeric variables that compose the problem, such as object positions, goal positions, binary state of an object, goals achieved, and so on. Each predicate is translated into a value in $\{0, 1\}$, while numeric variables are directly added to the state representation.

The cost function of the planning problem maps onto the reward function of the learning problem, so that $P(s', r|s, a) = 1$ if $r = -c(s, a, s')$, and $P(s', r|s, a) = 0$ otherwise. For shortest planning problems, we used a reward of -1 per action. We also used a positive reward $r_{\mathcal{G}}$ for reaching a goal state, to distinguish this condition from other possible ways to end an episode (for instance, the agent reaching a dead end, or maximum number of actions executed).

Algorithm 4 Meta-Reinforcement Learning for Heuristic Planning

1: **Input:** $\mathcal{N}$ number of tasks to generate
2: **Output:** Meta-trained value function v_θ
3: $\mathcal{T} \leftarrow$ Generate $\mathcal{N}$ tasks
4: Learn the optimal policies for all the tasks in $\mathcal{T}$ using PPO
5: Run ITTS on $\mathcal{T}$ to obtain a subset of optimal tasks $\mathcal{C}$
6: Meta train on parameters θ with RL^2 using tasks $\mathcal{C}$ to obtain v_θ.

6.2.2 Training Task Generation and Selection

The first step in the methodology consists of generating a number of training task candidates, forming the initial set of tasks $\mathcal{T}$. This step is in general domain dependent. A common way relies on a parametrized description of the domain, so that a distribution can be defined over the parameters' range. Examples of this method for task generation are discussed in the experimental section.

Once the set of initial tasks has been $\mathcal{T}$ generated, task selection is used to select a subset $\mathcal{C}$ from $\mathcal{T}$ and improve the quality of the training set. Since FETA was derived out of the insight we got from gradient-based meta-RL, which might not work with context-based approaches, we use ITTS as the task selection method.

6.2.3 Model Training

The meta-RL model is trained using $\mathcal{C}$ as a training set and Proximal Policy Optimization (PPO) [30] as the learning algorithm.

The meta-trained value function v_θ, parametrized by θ, is an adaptive estimate of the value v^* for every task in the domain. The adaptation takes place over the initial transitions, when the memory of the LSTM network fills up (cf. Section 3.2.1), providing the context of the new task. When learning converges on the training tasks, the learned value function can be used to define the planning heuristic, so that $h^{MRL}(s) = -v_\theta(s)$. The optimal value function for a given task is an admissible heuristic for that task. The meta-learned heuristic, however, is subject to function approximation and generalization across tasks; therefore it may not be admissible.

6.3 Experimental Evaluation

The experimental evaluation aims at: (1) showing that the meta-learned heuristic leads to the expansion of fewer states than both popular domain-dependent heuristics and a supervised-learned heuristic with the same state representation; (2) evaluating the quality of the generated plans, since the heuristic is not admissible in general, and may return suboptimal plans; (3) determining how many more tasks a supervised learning approach needs to reach a comparable performance.

As baselines, we used no heuristic (Blind), h^{max}, h^{add} and LM-cut, from the popular FastDownward system [44]. We also used a supervised learning approach, h^{SUPER}, derived from existing approaches (cf. Chapter 3), but using the same PDDL-based state representation as the meta-RL heuristic. This allows us to compare meta-RL with supervised learning without any specific representation tuning for either

method.

For the supervised-learned heuristic, we followed the popular approach of using optimal plans to obtain the real heuristic $h^*(s)$ of each state s of the plan. To obtain these optimal plans we used FastDownward. The network is trained using s as input and $h^*(s)$ as the target. We tuned the hyperparameters and architecture of the model, and selected the best performing combination.

6.3.1 Domains

To evaluate and compare h^{MRL} against classical planning heuristics from the Fast-Downward system, domains must be defined in a PDDL format. For that reason, we introduce a set of new domains. The domains we introduce are benchmarks of the classical planning track of IPC and the FF Domain Collection [43, 81], some of which have also been used to evaluate deep learning approaches in planning. Tasks were generated by drawing their parameters uniformly at random as reported for each domain below. In addition, we also randomly sampled 5 validation tasks for ITTS, and a number of test tasks. The number of training and test tasks varies for each domain, as some allow more parameter combinations than others. At least 10 test tasks were used for every domain. The learning process was repeated 5 times for both meta-RL and supervised learning, and all graphs show mean and standard deviation.

6.3.1.1 Snake

Snake was introduced in IPC-2018. In this domain, a snake navigates a grid with the goal of eating apples that are spawned in the grid. Each time an apple is eaten, the snake extends its length by one and a new apple spawns until there are no more apples left. The grid has a fixed size of 6x6 with 15 apples in all instances.

The grid is initialized with 5 apples, while the others spawn as the snake eats the available ones. The position of the apples and the initial position of the head and tail of the snake are randomly selected. A total of 20 training tasks were generated, of which ITTS selected 12. A total of 15 instances were used for testing.

6.3.1.2 Sokoban

Sokoban is a game from IPC-2008, also used to evaluate previous Deep Learning approaches [28, 26]. The agent must push objects around a grid with the goal of moving them to specific locations. The grid has size 5x5 with 2 objects and 3 obstacles in all instances. The initial positions of the agent, objects, and obstacles, and the position of the goals (one per object) are randomly selected. A total of 20 training tasks were generated, of which ITTS selected 11. A total of 12 instances were used for testing.

6.3.1.3 Gripper

Gripper is a modification of the domain used in IPC-1998, as used in previous work for learning heuristic functions [28]. In this domain there is a robot with two grippers that can carry an object each. The goal is to move a certain number of objects from one room to another. All instances have 2 rooms and 4 objects. The initial location of the objects and the robot, as well as the target room, are randomly selected. A total of 15 training tasks were generated, of which ITTS selected 10. A total of 10 instances were used for testing.

6.3.1.4 Blocksworld

Blocksworld is a popular planning domain from IPC-2001, also used to evaluate previous approaches [28]. A set of blocks lie on a table. The goal is to build stacks

of blocks. Only one block can be moved at a time. All instances had 4 blocks. Each block can be free, under another block, and/or over another block. The initial configuration of the blocks as well as the goal configuration are randomly selected. 15 training tasks were generated, of which ITTS selected 10. A total of 10 instances were used for testing.

6.3.1.5 Ferry

This domain was extracted from the FF Domain Collection, because it has been used to evaluate previous approaches [28, 82]. In this domain, a ferry must transport a certain number of cars from their start location to a goal location. The ferry can only carry one car at a time. In all instances there are 4 cars and 4 locations. The initial positions of the cars and the ferry, as well as the goal position of the cars are randomly selected. A total of 15 training tasks were generated, of which ITTS selected 12. A total of 10 instances were used for testing.

6.3.1.6 Nurikabe

Nurikabe was introduced in IPC-2018. In Nurikabe a robot must paint a certain pattern in a grid. The robot cannot move into locations that have been painted or have already been assigned to a non-painted block. All instances have a grid of size 5x5, 2 colours (black or white), and each colour has 3 or 4 assigned cells. The initial position of the robot, of the sources (where the robot can start painting), and the position of the cells that must be painted are randomly selected. A total of 15 training tasks were generated, of which ITTS selected 10. A total of 10 instances were used for testing.

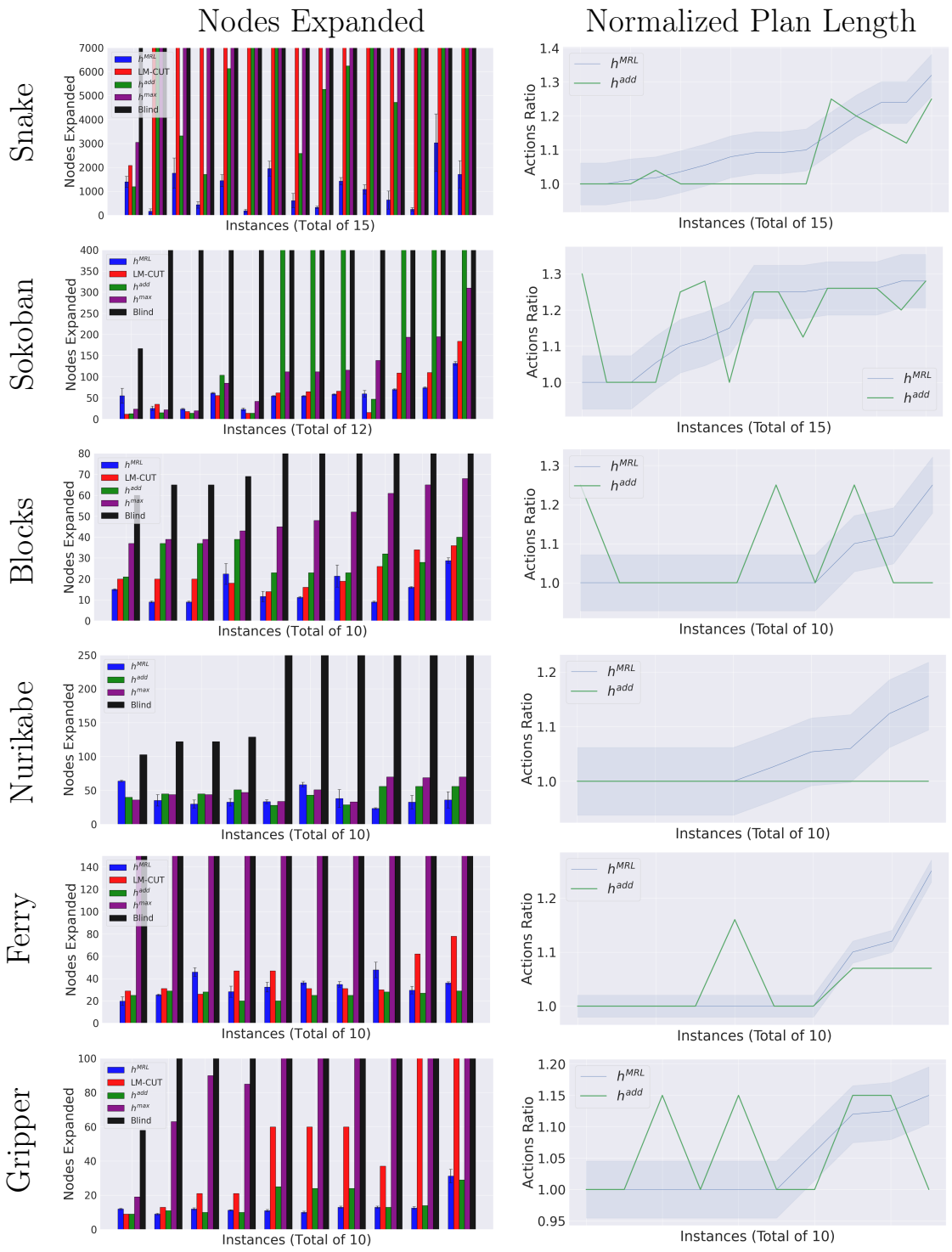


Figure 6.1: Comparison of h^{MRL} against domain-independent planning heuristics. Error bars in the bar plot, and shaded areas in the line plot, show the standard deviation of the learning method. The Y axis shows the number of nodes expanded to find a plan, the lower the better.

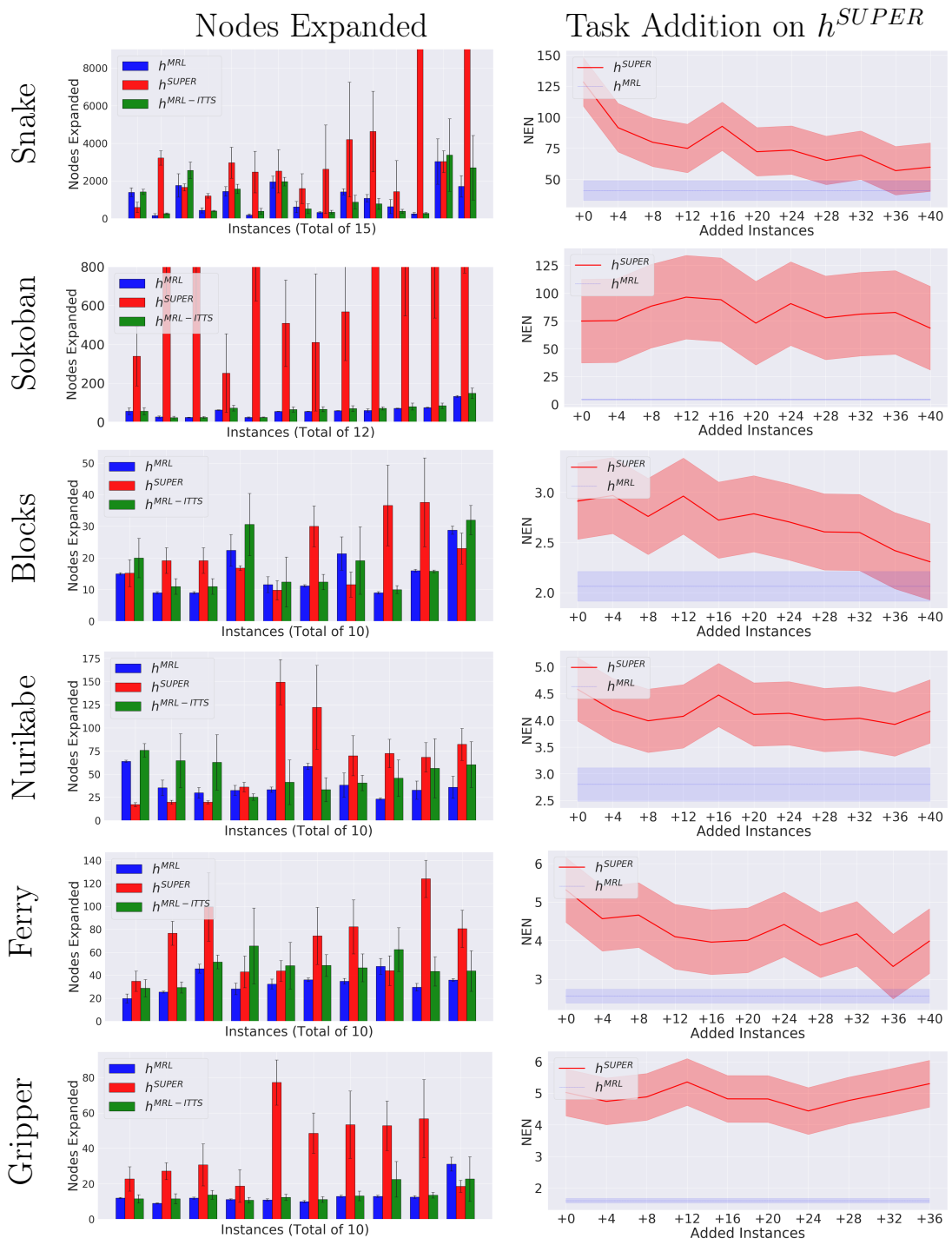


Figure 6.2: Comparison of the learning methods. Y axis shows the number of nodes expanded to find a plan, the lower the better. Error bars in the bar plot represent standard deviation over 5 repetitions. The shaded area in the Normalized Expanded Nodes (NEN) shows the 95% confidence interval, since this is the mean over all instances. The number after the + sign refers to the number of tasks added to the original training set for h^{SUPER}.

6.3.2 Results

Figure 6.1 shows the results of the comparison between h^{MRL} and the domain-independent heuristics in all domains. Since LM-CUT does not support conditional effects it is not possible to use it in Nurikabe. The left-most column of plots shows the nodes expanded by A* for each test instance in each domain, where the blind heuristic gives an indication of the difficulty of the instance. The instances, on the x axis, are sorted in increasing number of states expanded using the blind heuristic. While no heuristic outperforms all the other ones in every instance, h^{MRL} outperforms the baselines in most of them, and on average overall as shown in Table 6.1. The advantage is particularly evident on hard instances, where the blind heuristic leads to the expansion of the highest number of states. On the other hand, instances that are solved by short plans benefit the least, probably because the internal memory of the function approximator requires a few transitions to generate the context, and stabilize the estimate of the costs. The meta-learned heuristic is therefore less reliable in the first few states. The right-most column of the plots shows the length of the computed plans against h^{add}, which is also not admissible but quite commonly used, normalized over the length of the optimal plan. The meta-learned heuristic achieves comparable, if not better, plan length overall. Furthermore, all domain-independent heuristics show a performance that is highly dependent on the single instance, while h^{MRL} has quite consistent performance within each domain.

In the left-hand column of Figure 6.2, we show the results on the same instances against the supervised learning approach, and against a meta-learned heuristic without task selection, indicated as $h^{MRL-ITTS}$. The supervised heuristic h^{SUPER} has been trained on the same instances as h^{MRL}. ITTS improved the meta-learned

Heuristic	Expanded States
h^{MRL}	0.0749112
$h^{MRL-ITTS}$	0.0894698
h^{add}	0.1341158
LM-CUT	0.1473007
h^{SUPER}	0.1991076
$h^{\max}$	0.3566618
Blind	1

Table 6.1: Average number of expanded states over all domains, normalized with respect to the Blind heuristic.

heuristic in almost every instance, while at the same time greatly reducing the variance. The meta-learned heuristic outperforms h^{SUPER} in almost every instance, often by a large margin, with the exception of a few instances with short plans, as previously noted. Overall, h^{MRL} expands less than half the states as h^{SUPER}.

The last set of plots, in the right-most column of Figure 6.2, shows the performance of the supervised heuristic as the number of randomly generated training tasks increases. The "+0" value in these plots corresponds to the $\mathcal{C}$ training set for each domain, and reported in the domain description above. The y axis reports Normalized Expanded Nodes (NEN), which are the number of expanded states divided by the length of the optimal plan. The normalization allows us to average the results over all instances within the domain. In Snake and Block it takes 40 additional training tasks for the performance of the supervised heuristic to be comparable. In all other domains even with 40 additional tasks (which makes a total number of training tasks about 5 times larger) the supervised heuristic still expands more nodes than h^{MRL} trained over the "+0" set.

6.4 Summary

We introduced h^{MRL}, a meta-RL heuristic able to generalise in a wide range of planning domains. We show that h^{MRL} outperforms on average both popular domain-independent heuristics, and a supervised learning one. The meta-learning approach appears to be particularly advantageous on hard tasks, while it is less competitive on easier tasks, solved by short plans. A simple solution for this problem would be to expand states according to a domain-independent heuristic while the LSTM memory is filled, and let the meta-RL heuristic take over from there. These results were obtained without any particular encoding of the state space, which instead has received a fair amount of attention in the supervised learning literature. We expect that the results can be further improved with ad-hoc state encodings. Most importantly, this chapter shows that meta-RL heuristics are viable in planning, and thus creates a new line of heuristic learning.

Chapter 7

Future Work and Conclusions

7.1 Results Summary

In this thesis, we introduced the few-task framework for Meta-Reinforcement Learning, in which agents must be able to achieve good performance even when they have access to small set of training tasks. The proposed framework can help meta-RL to be practical in real-world and sensible scenarios where standard dense task sampling practices are not ideal.

With this framework in mind, we developed an Information-theoretic Task Selection which filters a set of available training tasks employing two concepts from information theory: KL-Divergence and Entropy. The outcome of ITTS is a smaller training set, which can be learned more quickly and results in better performance than the original set. ITTS demonstrated that proper task-selection can have a significant impact over the performance of meta-RL agents and shown that the quality of the training set, a property has been lacking attention in meta-RL, is directly related to the performance of meta-RL agents.

We studied and built insight into the behaviours learned by a gradient-based meta-RL method in a simple domain which allowed to visualize and analyse the learned policy. Moreover, we developed FETA, an improvement over ITTS, that takes

advantage of the policy transfer of properties of RL to filter tasks and build an optimal training set for meta-RL. FETA, a simpler and more efficient task selection process, showed to be able to achieve equal or better performance than ITTS in the experimental evaluation.

FETA helped us gain a better understanding of the task selection process and showed us that not all tasks are necessary for a meta-RL policy to learn a close to optimal exploration strategy that allows for fast adaption. Moreover, FETA showed that a simple task evaluation can be enough to significantly improve meta-RL's performance.

Finally, we proposed the use of meta-RL and ITTS to learn domain-dependent heuristic functions for classical planning. We evaluated our approach, which we call h^{MRL}, in many different domains, outperforming popular domain-independent heuristics and a standard supervising learning approach. With h^{MRL} we demonstrated the effectiveness of meta-RL and task-selection in an interesting application such as heuristic planning.

7.2 Limitations

In this work, we investigated the few-task framework, in which the agent has access to a limited number of tasks to train on, and demonstrated how task selection may help us enhance meta-RL performance in those settings. Although the work presented in this thesis bring many benefits to meta-RL, our methods have some shortcomings.

For instance, ITTS requires to learn the optimal policies of all the available tasks, which can be computationally expensive. Moreover, for ITTS to work optimally,

we need to fine tune the difference threshold, which requires training various meta-RL models until we find a good threshold value to utilize. Each new meta-RL model we train adds significant cost to the process which does not allow for an exhaustive parameter search. Furthermore, to create the validation tasks we assume that we have access to the target or test task distribution, which may limit the method's applicability in some domains.

On the other hand, FETA has a lower computational cost than ITTS and even lower cost than training with all available tasks in a simple domain. However, it is still quite expensive for complex domains, since it requires learning the optimal policies of the tasks that are selected, and the cost of learning the optimal policy of a task is proportional to its complexity. Moreover, FETA requires a successful transfer function to be defined, which assumes knowledge of the domain, knowledge that is often but not always accessible.

Finally, although h^{MRL} showed to be able to learn a efficient heuristic that expands a small number of states to find a solution, it requires an RL environment to interact with, and since most of the planning problems of interest in classical planning are defined in PDDL format, we require to translate the different PDDL parameters to an RL environment that can be used by h^{MRL}.

7.3 Future Work

Meta-Reinforcement Learning is a relatively new topic of research, in which many aspects are unexplored and require more attention. We believe that our work form a solid foundation for future research that can build and improve on our ideas.

For instance, since both of the proposed task selection methods are heuristic in nature, theoretical understanding of the task selection process in the context of

meta-RL is an interesting topic of research going forward. Understanding the effect that each individual task might have over the meta-learning process could help develop more efficient meta-learning methods and increase its efficacy in the real world.

Moreover, both of our proposed methods require the optimal policy of at least the tasks that have been selected. The information gained during the task selection process may be utilized during meta-training, reducing the added cost of task selection. Approaches such as offline meta-RL [83, 84] or guided meta-policy search [85] could be employed for that purpose, developing offline meta-RL methods that reuse the data collected during the tasks selection process or use imitation learning to combine the learned policies.

In this work, task selection was used to exclude tasks that could offer redundant data to the meta-RL agent. Future work to balance the task distribution in such a manner that similar tasks are sampled less frequently than tasks that are less present in the training set, without discarding any tasks, might be an alternative strategy to improve performance of meta-RL in a few-tasks setting.

Finally, in Chapter 6, we showed the benefits that meta-RL in our framework can have in an field such as heuristic planning. It would be interesting to identify other fields that might benefit from this framework and evaluate it in real-world applications such as robotics.

7.4 Conclusions

Meta-RL is an exciting research topic which has the potential for strong impact in the real world. It has been exciting to observe an increasing interest in the developing methods and tools that leverage training tasks in meta-learning such

as task scheduling [62, 63, 64, 65] and task generation [66, 67] to improve the performance of meta-learning models.

In a continually changing environment like the real world, meta-RL might be an ideal solution to many of today's research challenges, building autonomous agents that can quickly adapt to perform optimally in complex domains, such as robotics, autonomous driving and energy management. We hope that the work presented in this thesis serves as inspiration for future research in this area.

We conclude with the following:

- The use of task selection to improve the quality of the training set can considerably improve the performance of meta-RL agents, particularly in a few-task context.
- Learning a proper meta-RL policy which learns a proper exploration strategy that enables an agent to quickly adapt to new unseen tasks is possible with a limited number of training tasks.
- Meta-RL and task selection can considerably aid research areas such as heuristic planning that are interested in getting strong performance with a cheap training cost and a small pool of training tasks.

Bibliography

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. "A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play". In: *Science* (2018).

[2] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`. 2019.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever,

Jie Tang, Filip Wolski, and Susan Zhang. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

[4] D.K. Naik and R.J. Mammone. "Meta-neural networks that learn by learning". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 1. 1992, 437–442 vol.1. DOI: 10.1109/IJCNN.1992.287172.

[5] Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Springer Science & Business Media, 1998.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70 (ICML)*. 2017, pp. 1126–1135.

[7] JX Wang, Z Kurth-Nelson, D Tirumala, H Soyer, JZ Leibo, R Munos, C Blundell, D Kumaran, and M Botvinick. *Learning to reinforcement learn*. arXiv preprint arXiv:1611.05763. 2017.

[8] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. "Efficient off-policy meta-reinforcement learning via probabilistic context variables". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019.

[9] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. *RL2: Fast reinforcement learning via slow reinforcement learning*. arXiv preprint arXiv:1611.02779. 2016.

[10] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning". In: *International Conference on Learning Representations (ICLR)*. 2019.

[11] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. *Learning dexterous in-hand manipulation*. arXiv preprint arXiv:1808.00177. 2018.

[12] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. "An automated measure of MDP similarity for transfer in reinforcement learning". In: *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*. 2014.

[13] David Isele, Eric Eaton, Mark Roberts, and David W Aha. "Modeling Consecutive Task Learning with Task Graph Agendas". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018, pp. 1965–1967.

[14] Thommen George Karimpanal and Roland Bouffanais. "Self-organizing maps as a storage and transfer mechanism in reinforcement learning". In: *Adaptive Learning Agents (ALA) Workshop*. 2018.

[15] Jinhua Song, Yang Gao, Hao Wang, and Bo An. "Measuring the distance between finite markov decision processes". In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*. 2016, pp. 468–476.

[16] James L Carroll and Kevin Seppi. "Task similarity measures for transfer in reinforcement learning task libraries". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks (IJCNN)*. Vol. 2. 2005, pp. 803–808.

[17] M. M. Hassan Mahmud, Majd Hawasly, Benjamin Rosman, and Subramanian Ramamoorthy. *Clustering Markov Decision Processes For Continual Transfer*. arXiv:1311.3959. 2016.

[18] Felipe Leno Da Silva and Anna Helena Reali Costa. "Object-oriented curriculum generation for reinforcement learning". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018, pp. 1026–1034.

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

[20] Maxwell Svetlik, Matteo Leonetti, Jivko Sinapov, Rishi Shah, Nick Walker, and Peter Stone. "Automatic curriculum graph generation for reinforcement learning agents". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[21] Francesco Foglino, Christiano Coletto Christakou, Ricardo Luna Gutierrez, and Matteo Leonetti. "Curriculum Learning for Cumulative Return Maximization". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 2019.

[22] Francesco Foglino, Christiano Coletto Christakou, and Matteo Leonetti. "An Optimization Framework for Task Sequencing in Curriculum Learning". In: *Proceedings of 9th Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EpiRob)*. 2019.

[23] Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Learning to rank for synthesizing planning heuristics". In: *International Joint Conferences on Artificial Intelligence (IJCAI)*. 2016.

[24] Pawel Gomoluch, Dalal Alrajeh, Alessandra Russo, and Antonio Bucchiarone. *Towards learning domain-independent planning heuristics*. IJCAI-17 Workshop on Architectures for Generality and Autonomy. 2017.

[25] Mehdi Samadi, Ariel Felner, and Jonathan Schaeffer. "Learning from Multiple Heuristics". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2008.

[26] Edward Groshev, Aviv Tamar, Maxwell Goldstein, Siddharth Srivastava, and Pieter Abbeel. "Learning Generalized Reactive Policies Using Deep Neural Networks". In: *AAAI Conference on Artificial Intelligence*. 2018.

[27] Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. "Action schema networks: Generalised policies with deep learning". In: *AAAI Conference on Artificial Intelligence*. 2018.

[28] William Shen, Felipe Trevizan, and Sylvie Thiébaux. "Learning domain-independent planning heuristics with hypergraph networks". In: *International Conference on Automated Planning and Scheduling (ICAPS)*. 2020.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. The MIT press, 2018.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347. 2017.

[31] Tianbing Xu, Qiang Liu, Liang Zhao, and Jian Peng. "Learning to Explore via Meta-Policy Gradient". In: *International Conference on MachineLearning (ICML)*. 2018.

[32] Zhongwen Xu, Hado van Hassel, and David Silver. "Meta-Gradient Reinforcement Learning". In: *Conference on Neural Information Processing System (NeurIPS)*. 2018.

[33] Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. "Evolved Policy Gradients". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2018.

[34] Flood Sung, Li Zhang, Tao Xiang, and Timothy Hospedales. *Learning to Learn: Meta-Critic Networks for Sample Efficient Learning*. arXiv preprint arXiv:1706.09529v1. 2017.

[35] Bradly Stadie, Ge Yang, Rein Houthooft, Peter Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. "The importance of sampling in meta-reinforcement learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, pp. 9280–9290.

[36] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. "PROMP: Proximal Meta-Policy Search". In: *International Conference on Learning Representations (ICLR)*. 2019.

[37] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: *International Conference on Learning Representations (ICLR)*. 2020.

[38] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. "A simple neural attentive meta-learner". In: *In International Conference on Learning Representations (ICLR)*. 2018.

[39] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. *Meta-Learning in Neural Networks: A Survey*. 2020. arXiv: `2004.05439 [cs.LG]`.

[40] Malte Helmert and Carmel Domshlak. "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?" In: *Graph Search Engineering*. Ed. by Lubos Brim, Stefan Edelkamp, Erik A. Hansen, and Peter Sanders.

Dagstuhl Seminar Proceedings 09491. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010. URL: http://drops.dagstuhl.de/opus/volltexte/2010/2432.

[41] M. Fox and D. Long. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20 (Dec. 2003), pp. 61–124. ISSN: 1076-9757. DOI: 10.1613/jair.1129. URL: http://dx.doi.org/10.1613/jair.1129.

[42] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[43] Jörg Hoffmann and Benhard Nebel. "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* (2001).

[44] Malte Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* (2006).

[45] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. "Learning to generalize: Meta-learning for domain generalization". In: *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*. 2018.

[46] Nicolas Schweighofer and Kenji Doya. "Meta-learning in reinforcement learning". In: *Neural Networks* 16.1 (2003), pp. 5–9.

[47] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. "Meta-reinforcement learning of structured exploration strategies". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, pp. 5302–5311.

[48] Joshua Achiam Alex Nichol and and John Schulman. *On First-Order Meta-Learning Algorithms*. arXiv preprint arXiv:1803.02999. 2018. arXiv: 1803.02999.

[49] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*. arXiv:1703.03864. 2017.

[50] Sepp Hochreiter and JÃ¼rgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[51] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[53] Mike Huisman, Jan N. van Rijn, and Aske Plaat. "A Survey of Deep Meta-Learning". In: *CoRR* abs/2010.03522 (2020). arXiv: 2010.03522. URL: https://arxiv.org/abs/2010.03522.

[54] Malcolm J A Strens. "A Bayesian Framework for Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2000.

[55] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning

with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)*. 2018.

[56] Matthew E. Taylor and Peter Stone. "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *Journal of Machine Learning Research* 10.56 (2009), pp. 1633–1685. URL: http://jmlr.org/papers/v10/taylor09a.html.

[57] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. *Transfer Learning in Deep Reinforcement Learning: A Survey*. 2021. arXiv: 2009.07888 [cs.LG].

[58] Jivko Sinapova, Sanmit Narvekar, Matteo Leonetti, and Peter Stone. "Learning Inter-Task Transferability in the Absence of Target Task Samples". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2015.

[59] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. *A study on overfitting in deep reinforcement learning*. arXiv preprint arXiv:1804.06893. 2018.

[60] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey". In: *Journal of Machine Learning Research* (2020).

[61] Francesco Foglino, Matteo Leonetti, Simone Sagratella, and Ruggiero Seccia. "A Gray-Box Approach for Curriculum Learning". In: *World Congress on Global Optimization*. 2019.

[62] Jean Kaddour, Steindór Sæmundsson, and Marc Peter Deisenroth. "Probabilistic Active Meta-Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2020.

[63] Huaxiu Yao, Yu Wang, Ying Wei, Peilin Zhao, Mehrdad Mahdavi, Defu Lian, and Chelsea Finn. "Meta-learning with an Adaptive Task Scheduler". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2021.

[64] Chenghao Liu, Zhihao Wang, Doyen Sahoo, Yuan Fang, Kun Zhang, and Steven C.H. Hoi. "Adaptive Task Sampling for Meta-Learning". In: *European Conference on Computer Vision (ECCV)*. 2020.

[65] Su Lu, Han-Jia Ye, Le Gan, and De-Chuan Zhan. "Towards Enabling Meta-Learning from Target Models". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2021.

[66] Kyle Hsu, Sergey Levine, and Chelsea Finn. *Unsupervised Learning via Meta-Learning*. 2019. arXiv: `1810.02334 [cs.LG]`.

[67] Siavash Khodadadeh, Ladislau Bölöni, and Mubarak Shah. "Unsupervised Meta-Learning for Few-Shot Image Classification". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2019.

[68] Huaxiu Yao, Linjun Zhang, and Chelsea Finn. *Meta-Learning with Fewer Tasks through Task Interpolation*. arXiv preprint arXiv:2106.02695v1. 2021.

[69] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. *Unsupervised Meta-Learning for Reinforcement Learning*. 2020. arXiv: `1806.04640 [cs.LG]`.

[70] Allan Jabri, Kyle Hsu, Benjamin Eysenbach, Abhishek Gupta, Sergey Levine, and Chelsea Finn. "Unsupervised Curriculafor Visual Meta-Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2019.

[71] Stefan Edelkamp and Stefan Schrödl. *Heuristic Search: Theory and Applications*. Morgan Kaufmann, 2012.

[72] Santiago Franco, Levi H. S. Lelis, Mike Barley, Stefan Edelkamp, Moises Martinez, and Ionut Moraru. "The Complementary Planner in the IPC 2018". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2018.

[73] Shahab Jabbari Arfaee, Sandra Zilles, and Robert C. Holte. "Learning heuristic functions for large state spaces". In: *Artificial Intelligence* 175 (2011), pp. 2075–2098.

[74] Jordan T. Thayer, Austin Dionne, and Wheeler Ruml. "Learning Inadmissible Heuristic During Search". In: *International Conference on Automated Planning and Scheduling (ICAPS)*. 2011.

[75] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinıcius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. "Relational inductive biases, deep learning, and graph networks". In: *CoRR* abs/1806.01261 (2018). arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

[76] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)*. 2020.

[77] M.L. MenÃ©ndez, J.A. Pardo, L. Pardo, and M.C. Pardo. "The Jensen-Shannon divergence". In: *Journal of the Franklin Institute* 334.2 (1997), pp. 307–318. ISSN: 0016-0032. DOI: https://doi.org/10.1016/S0016-

0032(96)00063-4. URL: https://www.sciencedirect.com/science/article/pii/S0016003296000634.

[78] John D. Co-Reyes, Abhishek Gupta, Suvansh Sanjee, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. "Guiding Policies With Language Via Meta-Learning". In: *International Conference on Learning Representations (ICLR)*. 2019.

[79] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. arXiv preprint arXiv:1606.01540. 2016.

[80] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for OpenAI Gym*. https://github.com/maximecb/gym-minigrid. 2018.

[81] Jörg Hoffmann, Stefan Edelkamp, Sylvie Thiébaux, Roman Englert, Frederico dos Santos Liporace, and Sebastian Trüg. "Engineering Benchmarks for Planning: the Domains Used in the Deterministic Part of IPC-4". In: *Journal of Artificial Intelligence Research (JAIR)* (2006).

[82] Silvan Sievers, Michael Katz, Shirin Sohrabi, Horst Samulowitz, and Patrick Ferber. "Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019.

[83] Vitchyr H. Pong, Ashvin Nair, Laura Smith, Catherine Huang, and Sergey Levine. "Offline Meta-Reinforcement Learning with Online Self-Supervision". In: *International Conference on Machine Learning (ICML)*. 2021.

[84] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. "Offline Meta-Reinforcement Learning with Advantage Weighting". In: *International Conference on Machine Learning (ICML)*. 2021.

[85] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Sergey Levine Pieter Abbeel, and Chelsea Finn. "Guided Meta-Policy Search". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[86] The garage contributors. *Garage: A toolkit for reproducible reinforcement learning research*. `https://github.com/rlworkgroup/garage`. 2019.

Appendix A

Experimental Details

For ITTS and FETA experiments the same model architecture and hyperparameters were used, table A.1 show the parameters for the models trained with MAML while table A.2 show the parameters for the models trained with RL^2. For all the models trained with RL^2 a LSTM of 64 hidden nodes is used as the first layer, which is shared for the Actor and Critic. The first two rows show the number of layers and hidden nodes for the Actor and the Critic after the LSTM. The code used for the experiments can be here `https://github.com/RicardoLunaG/ITTS` for ITTS and here `https://github.com/RicardoLunaG/FETA` for FETA. For training the individual tasks and the meta-RL agents, as well as testing the agents garage was used [86].

	Ant	HC	KrazyWorld	MGEnv
Actor arch.	[64,64]	[64,64]	[64,64]	[100, 100]
Critic arch.	[32,32]	[32,32]	[64,64]	[64,64]
Gamma	0.99	0.99	0.99	0.99
Gae Lambda	1.0	1.0	0.95	0.95
Inner LR	0.1	0.1	0.1	0.1
Outer LR	0.0001	0.0001	0.0001	0.0001

Table A.1: Architecture and hyperparameters of the models trained with MAML for ITTS and FETA experiments. LR refers to learning rate.

	CartPole	MiniGrid	KrazyWorld	MGEnv
Actor arch.	[64]	[64]	[64]	[200]
Critic arch.	[64]	[64]	[64]	[200]
Gamma	0.99	0.99	0.99	0.99
Gae Lambda	0.95	0.95	0.95	0.95
LR	0.0001	0.0001	0.0001	0.0001

Table A.2: Architecture and hyperparameters of the models trained with RL^2 for the ITTS and FETA experiments. LR refers to learning rate.

For the h^{MRL} experiments two LSTMs of 64 hidden nodes each, one for the Actor and one for the Critic, were used as first layer. Table A.3 show the parameters used to train the models in each domain. The first two rows show the number of layers and hidden nodes for the Actor and the Critic after the LSTMs. The code used for the h^{MRL} experiments can be found in the following link: `https://github.com/RicardoLunaG/Meta-Reinforcement-Learning-for-Heuristic-Planning`

	Blocks	Ferry	Gripper	Nurikabe	Snake	Sokoban
Actor arch.	[64]	[64]	[64]	[126]	[64]	[126]
Critic arch.	[64]	[64]	[64]	[126]	[64]	[126]
Gamma	0.99	0.99	0.99	0.99	0.99	0.99
Gae Lambda	0.95	0.95	0.95	0.95	0.95	0.95
LR	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001

Table A.3: Architecture and hyperparameters of the models trained with RL^2 for the h^{MRL} experiments. LR refers to learning rate.

For the h^{SUPER} heuristic, the network was composed by standard dense layers. The parameters and architecture of the models used in each domain is shown in table A.4

	Blocks/Ferry/Gripper	Nurikabe/Snake/Sokoban
Architecture	[100,64,64]	[100,100,64]
LR	0.001	0.001

Table A.4: Architecture and hyperparameters of the models used to learn the h^{SUPER} heuristic. LR refers to learning rate.