Abstraction-Guided Modular Reinforcement Learning

Ponnambalam, C.T.

DOI
10.4233/uuid:0071a2f9-c56f-4f75-b6eb-63ebadadc918

Publication date
2023

Document Version
Final published version

Citation (APA)
Ponnambalam, C. T. (2023). *Abstraction-Guided Modular Reinforcement Learning*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:0071a2f9-c56f-4f75-b6eb-63ebadadc918

ABSTRACTION-GUIDED MODULAR REINFORCEMENT LEARNING

ABSTRACTION-GUIDED MODULAR REINFORCEMENT LEARNING

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by authority of the Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on Monday 26 June 2023 at 12:30 o'clock

by

Canmanie Teresa PONNAMBALAM

Master of Applied Science, University of Toronto, Canada,
born in Kitchener, Canada.

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus	chairperson
Dr. M. T. J. Spaan	Delft University of Technology, promotor
Dr. F. A. Oliehoek	Delft University of Technology, promotor

Independent members:

Prof. dr. F. M. Brazier	Delft University of Technology
Prof. dr. M. A. Neerincx	Delft University of Technology
Prof. dr. A. Nowé	Vrije Universiteit Brussel, Belgium
Dr.-Ing. J. Kober	Delft University of Technology
Dr. B. Peng	University of Liverpool, United Kingdom

This research is part of the research program Physical Sciences TOP-2 with project number 612.001.602, financed by the Dutch Research Council (NWO).

Cover by Carmel Freeman.

ISBN 978-94-6366-713-5

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

எண்பொருள வாகச் செலச்சொல்லித் தான்பிறர்வாய்
நுண்பொருள் காண்ப தறிவு.

- திருக்குறள் 424

Wisdom lies in keeping it simple for your audience,
and grasping all intricacies when others speak.

- Tirukkuṟaḷ 424

Translation for all Tirukkuṟaḷ text by Kannan, available at: https://thirukkural133.wordpress.com/.

CONTENTS

Summary xi

Samenvatting xiii

1 Introduction 1
- 1.1 Reinforcement Learning: The Future of AI . . . 2
  - 1.1.1 Challenges in RL . . . 4
- 1.2 Learning from Humans . . . 5
  - 1.2.1 The Power of Abstraction . . . 6
  - 1.2.2 Modular Learning . . . 7
- 1.3 The Abstraction-Guided Modular Approach: Learning to Learn . . . 8
  - 1.3.1 Research Questions . . . 8
  - 1.3.2 Thesis Outline . . . 10

References . . . 10

2 Background 15
- 2.1 Markov Decision Processes . . . 15
  - 2.1.1 Exact Algorithms . . . 16
  - 2.1.2 Challenges . . . 17
- 2.2 Reinforcement Learning . . . 17
  - 2.2.1 Algorithms . . . 18
  - 2.2.2 Offline Reinforcement Learning . . . 20
- 2.3 Abstraction in MDPs . . . 20
  - 2.3.1 State Abstraction . . . 21

References . . . 22

3 Parallel Learning in Abstractions 25
- 3.1 Formalisms . . . 27
  - 3.1.1 State-Conditioned Abstraction . . . 27
  - 3.1.2 Optimal-Policy-Preserving Abstractions . . . 28
- 3.2 Parallel Q-Learning in Abstractions . . . 29
  - 3.2.1 Case 1: Incomplete Abstractions . . . 30
  - 3.2.2 Case 2: Fully-Specified Abstractions . . . 32
- 3.3 Experimental Evaluation . . . 34
  - 3.3.1 Hallway Environment . . . 35
  - 3.3.2 Taxi . . . 38

3.4 Theoretical Discussion . . . 38
3.4.1 Non-Stationarity . . . 40
3.4.2 Conditions of Module Optimality . . . 40
3.4.3 Conditions of General Optimality . . . 42
3.4.4 Sample Complexity and Memory . . . 43
3.4.5 Fully-Specified Abstractions . . . 44
3.5 Transfer Learning . . . 45
3.5.1 Empirical Evaluation . . . 46
3.6 Related Work . . . 46
3.7 Limitations . . . 48
3.8 Conclusion . . . 49
References . . . 50

4 Uncertainty-Aware Predictive Reinforcement Learning 53
4.1 History Markov Decision Processes . . . 55
4.2 Uncertainty-Aware Predicted State Reinforcement Learning (PRESTO) . . . 55
4.2.1 Learning from History . . . 56
4.2.2 The Predictive Model . . . 57
4.2.3 Algorithm . . . 59
4.2.4 Mitigating Uncertainty . . . 61
4.2.5 Transferring the Predictor and the Policy . . . 61
4.3 Empirical Demonstration . . . 62
4.3.1 Sim2Real - Cartpole . . . 62
4.3.2 Hindsight - Tiger . . . 65
4.4 Related Work . . . 69
4.5 Conclusion . . . 71
References . . . 71

5 Abstraction-Guided Policy Recovery from Expert Demonstrations 75
5.1 RECO: Abstraction-Guided Policy Recovery . . . 76
5.1.1 Problem Formalization . . . 77
5.1.2 RECO in Tabular Domains . . . 78
5.1.3 Abstraction Selection . . . 81
5.1.4 RECO in Continuous Domains . . . 83
5.2 Empirical Evaluation . . . 85
5.2.1 Tabular Experiments . . . 85
5.2.2 Continuous Experiment . . . 87
5.3 Related Work . . . 89
5.4 Conclusion . . . 89
References . . . 90

6 Conclusion 93
6.1 Thesis Contributions . . . 94
6.1.1 Transfer . . . 94
6.1.2 Efficiency . . . 95
6.1.3 Transparency . . . 95
6.1.4 Research Questions . . . 95

6.2 Where to Next? . . . 96
6.2.1 Deep RL . . . 97
6.2.2 Model-based RL . . . 97
6.2.3 Lifelong RL . . . 98
References . . . 98

Acknowledgements **101**

Curriculum Vitæ **103**

List of Publications **105**

SUMMARY

Reinforcement learning (RL) models the learning process of humans, but as exciting advances are made that use increasingly deep neural networks, some of the fundamental strengths of human learning are still underutilized by RL agents. One of the most exciting properties of RL is that it appears to be incredibly flexible, requiring no model or knowledge of the task to be solved. However, this thesis argues that RL is inherently inflexible for two main reasons: 1. If there is existing knowledge, incorporating this without compromising the optimality of the solution is highly non-trivial, and 2. RL solutions can not be easily transferred between tasks, and generally require complete retraining to guarantee that a solution will work in a new task.

Humans, on the other hand, are very flexible learners. We easily transfer knowledge from one task to another, and can learn from knowledge that we learned in other tasks or that other people share with us. Humans are exceptionally good at abstraction, or developing conceptual understandings that allow us to extend knowledge to never-before-seen experiences. No artificial agent nor neural network has displayed the abstraction and generalization capabilities of humans in such varied tasks and environments. Despite this, utilizing the human as a tool for abstraction is commonly done only at the stage of defining the model. In general, this means making choices about what to include in the state space that will make the problem solvable without adding unnecessary complexity. While necessary, this step is not explicitly referred to as abstraction, and it is generally not considered relevant to how RL is applied. Much of the research in RL is less focused on how the problem is modelled, and instead centers the development and application of computational advances that allow for solving bigger and bigger problems.

Applying abstraction explicitly is highly non-trivial, as confirming that an abstract problem preserves the necessary information of the true problem can generally only be done if a full solution is already found, which may defeat the purpose of finding an abstraction if such a solution cannot be found. When such a confirmation can be made, the abstraction can be the result of a very complex function that would be difficult for a human to define. In this work, human-defined abstractions are used in a way that goes beyond the initial definition of the problem.

The first approach, presented in Chapter 3, breaks a problem into several abstract problems, and uses the same experience to solve each at the same time. A meta-agent learns how to compose the learned policies together to find the optimal policy. In Chapter 4, a method is introduced that uses supervised learning to train a model on partially-observable experience which is labelled with hindsight. The agent then learns a policy on predicted states, trading off information gathering with reward maximization. The last method presented in Chapter 5 is a modular approach to offline RL, where even with expert data, the method can become ineffective if the given data does not cover the entire problem space. This method introduces a second problem of recovering the agent to a state where it can safely follow the expert's action. The method applies abstraction to

multiply the given data and safely plan recovery policies. Combining the recovery policies with the imitation policy maintains high performance even when the expert data provided is limited.

In the methods developed in this research, a learning-to-learn component enables the agent to relax the usually strict requirements of abstraction, the parallel processing allows the agent to learn more from fewer samples, and the modularity means that the agent can transfer its knowledge to other related tasks.

SAMENVATTING

Reinforcement learning (RL) modelleert het leerproces van mensen, maar ondanks dat er indrukwekkende vooruitgangen worden gemaakt waarbij steeds diepere neurale netwerken worden gebruikt, worden sommige van de fundamentele sterke punten van menselijk leren nog steeds onvoldoende benut door RL-agenten. Een van de meest fascinerende eigenschappen van RL is dat het ongelooflijk flexibel lijkt te zijn, zonder een model of kennis van de te verrichten taak nodig te hebben. Dit proefschrift betoogt echter dat RL inherent niet flexibel is om twee belangrijke redenen: 1. Als er voorkennis is, is het niet triviaal om dat te integreren zonder de optimaliteit van de oplossing aan te tasten, en 2. RL-agenten kunnen niet makkelijk worden omgeschakeld tussen verschillende taken, en moeten in het algemeen opnieuw getraind worden om te garanderen dat een oplossing in een nieuwe taak zal werken.

Mensen daarentegen kunnen zeer flexibel leren. Wij kunnen gemakkelijk kennis overdragen van de ene taak naar de andere, en kunnen leren van inzichten die wij in andere taken hebben opgedaan of die andere mensen met ons delen. Mensen hebben een uitzonderlijk hoog abstractievermogen, zoals het ontwikkelen van conceptuele begrippen waarmee we kennis kunnen overdragen naar volledig nieuwe ervaringen. Geen enkele kunstmatige agent of neuraal netwerk heeft het abstractie- en generalisatievermogen van de mens evenaart in zulke uiteenlopende taken en omgevingen. Desondanks wordt het abstractievermogen van de mens gewoonlijk alleen benut bij het definiëren van het model. Over het algemeen betekent dit dat keuzes worden gemaakt over wat in de toestandsruimte moet worden opgenomen om het probleem oplosbaar te maken zonder onnodige complexiteit toe te voegen. Hoewel deze stap noodzakelijk is, wordt het niet expliciet abstractie genoemd, en wordt het over het algemeen niet relevant geacht voor de toepassing van RL. Veel van het onderzoek in RL is minder gericht op hoe het probleem wordt gemodelleerd, en concentreert zich in plaats daarvan op de ontwikkeling en toepassing van computationele vooruitgang die het mogelijk maakt steeds grotere problemen op te lossen.

Het expliciet toepassen van een abstractie is zeer moeilijk, omdat het bevestigen dat een abstract probleem de noodzakelijke informatie van het ware probleem behoudt, meestal alleen kan worden gerealiseerd als er al een volledige oplossing is gevonden, wat het doel van de abstractie kan ondermijnen als zo'n oplossing niet gevonden kan worden. Wanneer een dergelijke bevestiging wel mogelijk is, kan de abstractie het resultaat zijn van een zeer complexe functie die voor een mens moeilijk te definiëren zou zijn. In dit werk worden door mensen gedefinieerde abstracties zodanig gebruikt dat ze verder gaan dan alleen de initiële definitie van het probleem.

De eerste methode, gepresenteerd in hoofdstuk 3, splitst een probleem op in verschillende abstracte problemen, en gebruikt dezelfde ervaringen om ze allemaal tegelijk op te lossen. Een meta-agent leert hoe het geleerde gedrag gecombineerd moet worden om het optimale gedrag te vinden. In hoofdstuk 4, wordt een methode geïntroduceerd

die supervised learning gebruikt om een model te trainen op partially-observable ervaringen die met kennis van achteraf worden gelabeld. De agent leert dan een strategie op verwachte states, waarbij het verzamelen van nieuwe informatie wordt afgewogen tegen het maximaliseren van de beloning. De laatste methode die in hoofdstuk 5 wordt gepresenteerd is een modulaire aanpak van offline RL, waarbij zelfs met deskundige data de methode ineffectief kan worden als de gegeven data niet de gehele probleemruimte bestrijkt. Deze methode introduceert een tweede probleem om de agent terug te brengen naar een staat waarin deze veilig de actie van de expert kan volgen. De methode past abstractie toe om de gegeven data te vermenigvuldigen en de strategie voor terugkeren veilig te plannen. Door de strategie voor terugkeren te combineren met het imiteren van de expert blijven de prestaties hoog, zelfs wanneer de verstrekte deskundige data beperkt is.

De in dit onderzoek ontwikkelde methoden maken het mogelijk voor de agent om, dankzij een “leren-leren"-component de gewoonlijk strenge eisen van abstractie te versoepelen, dankzij de parallelle verwerking meer te leren uit minder voorbeelden, en dankzij de modulariteit de geleerde kennis over te dragen naar andere verwante taken.

1

INTRODUCTION

கல்லாத மேற்கொண் டொழுகல் கசடற
வல்லதூஉம் ஐயம் தரும்.
- திருக்குறள் 845

Brazenly acting out what one has not learnt
arouses doubts over that they've flawlessly mastered.
- Tirukkuṟaḷ 845

Artificial intelligence (AI) has been an idea long thought of only hypothetically. For a select few, defining and producing it offered a philosophical and theoretical challenge. In the mainstream, it was a concept contained safely within the walls of science-fiction. Perhaps the most limiting factor has been the definition itself. Artificial Intelligence: The Human Brain in a Machine. It could be a natural property of the human ego to resist the idea that one's great intelligence can be written to a microchip. This idea of capturing human intelligence in a computer is a frequently sought-after goal that has taken many forms in past decades. In 2011, IBM's supercomputer Watson defeated human contestants in a game of *Jeopardy!*, armed with 4 terabytes of stored data including the entire contents of Wikipedia [1]. This idea of intelligence formed by vast deposits of knowledge has now pivoted toward the concept of *learning* as intelligence. Turing himself had postulated that it would be easier to model a child's mind than that of an adult [2], alluding to the development of learning systems that would develop conceptual understandings rather than simply store information for retrieval.

Machine learning models are a departure from the view of intelligence as a deposit of knowledge and toward systems that are fed data and learn an implicit understanding, represented as a predictive (black-box) model. These days, the definition of artificial intelligence has softened to encompass an umbrella of analytical and decision-making methods, with the term often being used synonymously with machine learning and other long-established statistical tools. This change in terminology, and the recent advances in the latter, means that AI is no longer a fantasy, but both a necessity and

a potential threat. Practical applications of machine learning are now commonplace and what was once referred to only in science-fiction is now a fact of every day life. As more people are accepting the benefits that computational analysis of vast deposits of data brings, the potential downsides and opportunities for abuse are also entering the conversation. Collecting exhaustive amounts of data results in growing privacy violation concerns. Data is known to contain the same biases as the systems that created it, and increasingly complex algorithms are making decisions that can greatly affect day-to-day life, with little explanation as to how or why. Data collection, storage, analytics and retrieval has enormous environmental consequences in terms of energy usage. The emergence of more powerful AI is not necessarily welcomed by all, and certainly there remains a need for improvement to current methods.

1.1. REINFORCEMENT LEARNING: THE FUTURE OF AI

Machine learning methods are often divided simplistically into two categories: supervised and unsupervised. In supervised learning, a computational model is trained by feeding the model labeled data so that it can predict the labels on data for which the label is unknown. This encompasses classification and recognition tasks, such as isolating and identifying faces in a video. In unsupervised learning, a computational model is fed unlabeled data and tasked with labelling the data in some meaningful way. This could, for example, refer to a recommender system that groups users into different types in order to suggest films they are likely to enjoy. Both of these categories of machine learning approaches have proven to be highly applicable to real-world problems [3].

Another line of research lies in modelling and solving problems that require decisions that take into account long-term effects. Termed sequential decision-making problems, these encompass planning problems such as choosing shipping routes, controlling power plant operations, or designing the optimal layout of a warehouse. These problems come with their own set of challenges, and researchers developing methods for automated planning algorithms have been tackling this class of problems for decades. These approaches generally require defining a precise model of the problem dynamics which can be very difficult to craft, making it attractive to use alternative methods that are capable of learning such a model through experience or data. The machine learning methods described above, while powerful and effective, are not designed for long-term sequential decision-making.

This thesis concerns itself with one of the most promising approaches to artificially-intelligent systems today – sometimes considered a third category of machine learning – reinforcement learning. Reinforcement learning (RL) embeds the predictive capabilities of machine learning into the mathematical framework of sequential decision-making used in classical planning algorithms. Rather than learning to optimize immediate reward signals, as is the general case in supervised learning, RL requires learning multi-step plans that optimize reward over a longer period of time. In RL, unlike in the classical planning problem, it is assumed that little or nothing about the environment dynamics or optimization function is known. The agent learns only with direct access to the environment rather than to data or models. Put in the most optimistic terms, RL algorithms form artificially-intelligent agents that learn to take optimal sequences of actions without any prior knowledge. An RL agent explores its environment, which provides feed-

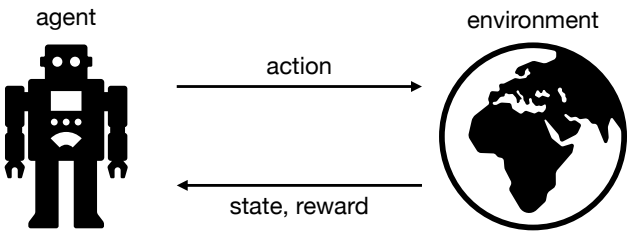

Figure 1.1: Reinforcement learning involves an agent taking actions and learning from how the environment responds.

back in the form of rewards (or penalties), and learns only through experience how to optimize its long-term returns.

Reinforcement learning was formed from an understanding of how animals (including humans) learn, and its principles were rooting in psychology decades before any mathematical framework was attached to it [4]. In early literature, the idea emerged that learning occurs only when the knowledge one has or assumes in the form of a model is proven to be wrong [5]. This is further elaborated on in the Rescorla-Wagner model, which speaks of learning as what occurs only when events violate expectations [6]. This suggests that one can only learn if ones prediction or model of the world is proven wrong. This type of thinking inspired the temporal difference learning paradigm, which is where a formulation resembling the current mathematical framework for RL first emerged [7]. The basic idea of temporal difference learning is that the agent maintains an estimate of how rewarding taking any action is (in the long-term) by trying it out and seeing what results. From the environmental feedback, it continuously updates its estimates.

At first, the agent can only arbitrarily guess how good any action is, but over time as it continues to update based on the error between its estimate and its true experience, the agent learns the true value of all possible actions. What sets this apart from other types of machine learning is the feedback loop incurred by interacting with the environment (depicted in Figure 1.1), as well as the temporal aspect. The agent learns how the environment reacts to the actions taken (either directly or indirectly), and finds the best sequence of actions to maximize the total of rewards over a period of time.

Temporal difference learning forms the backbone of reinforcement learning as it is known today. While RL continued to mature and develop a strong theoretical basis [8], there was little mainstream attention. Even though there was notable progress, such as the development of TD-Gammon (an RL algorithm that learned to play Backgammon [9]), RL was still unable to match human performance. This has, in many ways, all changed in the last decade. The field of reinforcement learning has enjoyed immense success as it benefits from the exponentially growing computational power, and power of the tools to analyze it, available today. There has been mainstream news coverage

of RL algorithms surpassing human performance in classical board games [10], Atari games [11], and complex video games such as Starcraft [12]. However, the reality, which is explored in this thesis, is that RL agents are limited by a number of factors and the boom in reinforcement learning is still limited to either theoretical or highly-crafted scenarios. Ultimately, this thesis proposes that reinforcement learning needs further development that goes back to its roots in human learning, both by incorporating more understanding of human cognitive processes and by enabling human input.

1.1.1. CHALLENGES IN RL

Reinforcement learning makes a big promise: artificial agents that learn by themselves without any prior knowledge. However, the framework is attractive mostly for its mathematical rigor and strong theoretical basis. Real-world problems, on the other hand, offer several challenges that limit the applicability of RL in real systems [13]. Some of the challenges that have been highlighted in literature and are relevant to this thesis are summarized below.

CURSE OF DIMENSIONALITY

Real-world problems are generally very large and complex. This offers a challenge for any computational method, as approaches for high-dimensional problems are subject to the *curse of dimensionality* [14]. This *curse* describes the issue of the exponential increase in the problem space with the linear increase of dimensions in the problem, which quickly renders even the most effective methods computationally infeasible for large problems.

SAMPLE COMPLEXITY

One of the benefits of reinforcement learning is also its downfall. In the ideal case, reinforcement learning algorithms do not assume anything about the task, making them promising learners of complex problems that are difficult to formally specify. This requires RL agents to exhaustively explore the environment to find the best policy, requiring huge amounts of data collected by interacting with the environment. To guarantee convergence to the optimal solution in discrete Q-learning, a popular RL algorithm, the agent must visit every part of the state-action space an infinite amount of times. This is, of course, not possible in practice. Whether problems have a tabular (discrete) state space or continuous state space, the key issue remains that the state-action space must be sufficiently explored; sub-optimal states and actions must also be experienced thoroughly in order to learn an optimal policy. When knowledge is embedded into RL algorithms to improve sample complexity, it might put hard constraints on RL agents that keep them from exploring sufficiently to find the true optimal policy. This downside can occur when applying action elimination [15], temporal abstraction [16], state abstraction [17], and safety constraints [18], to name a few examples.

EXPLAINABILITY

A reinforcement learning agent learns a policy, either directly or indirectly in conjuction with a value function. During learning, both are updated iteratively and (in the ideal case) come closer and closer to the true value function and optimal policy. However,

the error used to steer learning may not correspond to the true error [19]. In addition, the agent generally explores around its current idea of the optimal policy, reducing the number of random actions it takes over time. This means that the value function will not reflect the entirety of the problem and does not provide much insight into what the agent has learned. In practice, learning has to stop or be used at some point, and at this point it may not be clear how close the learned policy and value function are to the ground truth. The policy will provide an action for every state, but with no indication of how close to optimal the action (or overall policy) is, and why this action was selected.

In general, an RL agent does not explain what features of a current state lead to its proposed action, nor how certain it is in its proposal, nor the potential risk of taking the action. A human employing the RL agent may find such insights crucial. The need for more explainable policies has been highlighted in several works of literature [20].

OFFLINE LEARNING

Reinforcement learning is generally thought as an online algorithm that steers data collection, building up the data set it is trained on as it learns iteratively. Another approach is offline reinforcement learing, in which a dataset of experience is provided and a policy must be learned on this exisiting data without any further exploration. When all the training data is available beforehand, applying reinforcement learning naïvely has been known to fail for a number of proposed reasons [21], most cited of which is the lack of coverage of the true problem space available in the data set. This leads to poor estimations of the value of actions that are not represented sufficiently in the data, which are known to tend toward overestimation [22]. There is great interest in the development of offline reinforcement learning algorithms designed for this case, as learning from a static data set would alleviate many of the inherent issues facing RL, some of them listed above.

1.2. LEARNING FROM HUMANS

While reinforcement learning emerged as a mathematical framework of human learning, it offers an inflexible and rather poor mimicry of human decision-making power. Interestingly, attempting to formalize more concepts of human learning into the framework of RL has also led to insights (and sometimes evidence) back into the understanding of human decision-making [23]. Even so, there is still a major gap between the capabilities of an RL agent (or any machine learning method) and of even a child.

Humans are masters of efficiency. The sensory input to the human brain includes high-resolution visual, auditory and tactile cues, to name only a few. Considering our incredible ability to convert high-dimensional information into long-term plans, one can quickly determine that there are a number of tools available to humans to manage the processing of information. Neural networks, the powerhouse tool of most machine learning algorithms today, are modeled after neurons in the brain and the synapses they use to communicate information between them. Even so, and after continuous development in computational power, humans are much more efficient than methods that employ neural networks at innumerable tasks, such as understanding speech or language [24]. Imagine how easily a child can distinguish between a pineapple and a hairbrush, even after seeing only one example of each, whereas a neural network may need

thousands of examples [24]. This efficiency is apparent not only within a single skill, but between skills. Humans do not learn from scratch or need to be reconfigured every time things in the environment change [24].

The current trends in RL show that researchers are aware of its limitations and drawbacks and are actively seeking to address them. Meta-RL [25], lifelong RL [26], combinations of model-based and model-free RL [27], all of these branches of research address the enormous gap between RL and human learning. Two of the capabilities available to humans form the basis of this manuscript: abstraction and modularity. By embedding these tools into RL learning systems, we can lower the barrier to applying RL in real-world problems.

1.2.1. THE POWER OF ABSTRACTION

One of the keys of efficient information processing is inarguably *abstraction*. While abstraction fittingly has no single definition, one that applies broadly is that of Burgoon, which states that abstraction is the "process of identifying a set of invariant central characteristics of a thing" [28]. Humans, and all naturally occuring intelligent decision-makers, make great use of abstraction. We are remarkably capable at navigating high-dimensional spaces, with estimates that our brains are sent 11 million pieces of information per second through our many sensory systems [29]. Abstraction allows us to selectively focus on small subsets of information at a time as needed. It is the tool by which we turn vast amounts of complex information into manageable and meaningful concepts on which to base our decisions [30]. This conceptual knowledge allows us to assign significance to certain features or attributes and use these to induce information about something not explicitly seen before. For example, we may understand a car to have four wheels and an engine; we can apply this conceptual understanding to a vehicle seen for the first time, even if it is perceptually quite different from what we have seen before, and derive some notion of its function. Conceptual understanding not only aids efficient information processing in a single task, but also is the key to transferring information between tasks, finding a common axis they share. The ability to extend prior knowledge in new scenarios is a pillar of human intelligence [31]. This is yet another downfall of RL agents: their inability to generalize to even only slightly new environments or tasks.

As sample complexity has been mentioned above as one of the key issues facing reinforcement learning algorithms, the ability to apply abstraction is cherished as a direction in the field [32]. Researchers have been interested in defining abstraction [33], applying given abstractions [34], and learning abstractions [17, 35]. Abstraction has been applied to improve sample efficiency [36], to transfer knowledge between tasks [37], devise explainable policies [38], and learn safely [39].

It is easy to sell the benefits of abstraction, but much harder to apply it. The cost of abstracting away details is that you can lose important information that may be necessary for optimal (or good) planning. As deep reinforcement learning continues to achieve impressive results in high-dimensional environments, the need for explicitly encoding abstraction abilities into RL agents has been moved to the sidelines. Neural networks are implicit tools of abstraction, iteratively adjusting the weights of their many nodes during training to combine inputs at each layer in some way that minimizes

loss. Their power has seemingly removed the need for learning and applying abstraction in RL. However, there are many drawbacks to this implicit approach. In complex high-dimensional state spaces, planning in the true space is only possible for short time horizons [40]. Further, deep RL methods are not achieving the sample efficiency of humans [23]. Neural networks will also overfit to their training task data, exhibiting very weak inductive bias that degrades performance in transfer [24]. Finally, it is extremely difficult to glean any meaning from what neural networks have learned, or whether the features hidden in their many layers can provide any useful insights.

As the field moves toward hiding abstraction abilities in neural networks, there is still much to benefit from the natural human ability to devise meaningful subsets from vast amounts of data, using abstraction to learn efficiently, and extrapolating what they've learned across a lifetime of tasks and experiences.

1.2.2. MODULAR LEARNING

Another concept frequently exercised in natural intelligence and closely tied to abstraction is that of *modularity*, or breaking tasks down into separable problems [41]. Any truly intelligent being should have separable components and capabilities and should not rely on only a single representation of their world [42]. By learning in a modular way, insights gained (and packaged into modules) can be distributed across new tasks and shared with others. With this skill of compositionality, humans show yet again their ability to be incredibly data-efficient [24]. An example shows up early in children who quickly exhibit understanding of models such as "intuitive physics" [24]; children do not have to learn the basic laws of gravity every time they play with a new toy. Reinforcement learning agents generally assume nothing about the environment, which is both the attractive part of RL and its downfall. It allows them to find an optimal policy purely by exploring, but renders them inefficient and unable to make use of information provided in other forms.

Taking a modular approach also allows humans to expand and compress their local models seamlessly, adding or removing information input as needed [43]. We can incorporate knowledge learned from other tasks or given to us by other humans. In reinforcement learning, the state space is specified and remains fixed either for the entire life of the agent, or for a specified task with known boundaries. This means excessive information must be included in the state that may not be relevant at all times; essentially the agent must always plan over the union set of information that may be required at any point in their learning life.

The first emergence of modular methods (by that name) in reinforcement learning came in 2003, with Modular SARSA(O) [44] and Q-Decomposition [45]. These works highlighted the potential benefits of modular approaches to RL, such as efficiency, transfer and facilitating abstraction. However, they also showed that these modular approaches required fine hand-tuning or expert knowledge (in the form of local reward functions and appropriate arbitration functions), and even then could not be provably optimal. Since then, the term *modular RL* has fallen out of use, never forming a clear definition [23]. Instead, *hierarchical RL*, which falls under the umbrella of modular methods, has continued to flourish. Both modular and hierarchical methods share the goal of breaking up a problem into sub-problems (often to facilitate abstraction and trans-

fer), but in the latter a hierarchical structure is explicitly imposed. Because of this, the solution to a sub-task is often closely coupled to the solution of the parent tasks. Both modular and hierarchical RL have strong ties to the idea of abstraction, as forming sub-problems provides an opportunity to abstract away details which are irrelevant to local goals.

A particularly powerful consequence of modular learning is the ability to learn from knowledge provided in many forms, and that are either learned by oneself in other tasks or given by another being. Incorporating knowledge into RL algorithms is not trivial, and as mentioned above can often remove the flexibility RL agents require for optimality. This means that despite the tools available to humans that allow them to excel over machine learning, learning to make use of human input is still a missing link in RL today.

1.3. THE ABSTRACTION-GUIDED MODULAR APPROACH: LEARNING TO LEARN

Reinforcement learning has its roots in human learning, but fails to empower the unique properties in which humans excel. While RL agents can effortlessly run millions of simulated steps, no artificial agent nor neural network has displayed the abstraction and generalization capabilities of humans in such varied tasks and environments. In addition, humans can build on knowledge acquired from many sources and in many ways, while research in RL often ignores the option to allow for input from other sources, whether they be human experts or other pre-trained agents. The goal of this thesis is to derive RL algorithms that empower reinforcement learning agents to learn more like humans, while at the same time learning from humans by making use of human-defined abstractions.

In reinforcement learning, abstraction often occurs in the form of a mapping between a state space and a compact state space that maintains some crucial properties of the original state space. Exact abstractions for problems modelled as Markov decision processes (MDPs) are extremely difficult to calculate. This limits the ability to use human experts as a starting point for providing useful abstractions. Even approximate abstractions for MDPs can be extremely complex to calculate [46] and provide hardly any guarantees. This work explores the use of partial abstractions, that is, abstractions that do not model the entire problem, but can still prove useful to an RL agent in solving the entire problem. The methods in this thesis take as input human-defined abstractions and incorporate them as modules into methods which can be used to learn efficiently and to facilitate the transfer of knowledge between different tasks. While it may sound trivial to learn from existing knowledge, applying any sort of rigidity or structure into an RL system can mean the optimal (or even a good) solution is never found. The key goal of the methods presented here is for RL agents to be able to take in human-definable knowledge and benefit from it, rather than compromise their own desirable properties.

1.3.1. RESEARCH QUESTIONS

The methods introduced in this thesis have several things in common. They all assume some human knowledge is given, mainly in the form of a state abstraction mapping, and aim to mitigate the influence of error in this abstraction on the final policy. They

all involve multiple learning processes conducted in parallel, and employ the idea of learning to learn. Learning to learn (or meta-learning) is another straightforward human ability (also exhibited by monkeys [23]) that is essential to true AI, considered one of the keys to sample efficient learning in humans [24].

Learning to use abstractions to boost reward while performing policy optimization is particularly difficult because optimality of an abstraction can only be confirmed when the correct Q-values, model and/or optimal policy is known. In the general online case, there is no access to these values, thus the suitability of an abstraction must be either learned or assumed. This thesis explores these and the issues mentioned previously, by exploring the following research question:

- How can abstraction and modularization facilitate flexibility in reinforcement learning?

This question is broken down into the following sub-questions, listed here:

- Is there value to abstraction mappings that are intuitive and human-definable, even if they form poor abstractions of the MDP? (Chapters 3 and 5)
  - This question concerns the use of approximate or conditional abstraction, in which an abstraction can be useful for certain sub-tasks but is not suitable as an abstraction of the MDP. Such an abstraction, it is argued, is much easier for a human to define. In Chapter 3, it is seen whether such abstractions can be used without compromising the ability to find an optimal policy, and under which conditions. In Chapter 5, a sub-task is explicitly defined as a separate MDP formed by a given abstraction, and used only to bring the agent back to a state where it can continue to carry out its known policy.
- Is it worth the additional computation requirements that learning in several parallel processes incurs? (Chapters 3, 4 and 5)
  - Learning in a modular way can often introduce additional computation during training. This thesis is motivated by the idea that modularity provides benefits in ways which dwarf the negative effects of such initial computation. These benefits are explored in each chapter in this thesis.
- How can modular approaches contribute to the goals of lifelong reinforcement learning, such as robustness, transfer and learning from offline data? (Chapters 4 and 5)
  - Lifelong RL seeks to produce agents with the ability to adapt to continuously changing tasks, with the goal of being robust to fluctuations in the goals or environment and thus more efficient over a long period of time. This thesis argues that modularity and abstraction are key to achieving these goals. In Chapter 4, transfer is explored by using supervised learning to explicitly incorporate uncertainty into the policy. In Chapter 5, abstraction is used to boost learning in the offline setting when only an expert policy is provided.

1

1.3.2. THESIS OUTLINE

This thesis explores the use of abstraction and modularity by introducing three novel approaches to unique but relevant problems. Each of these approaches takes human-definable abstractions as input, and uses feedback from the environment to learn how to effectively use them.

First, Chapter 2 introduces necessary background material and notation that will be referenced throughout the thesis. In Chapter 3, an algorithm is presented that takes human-devised abstractions as input and uses discounted reward as feedback to learn how to use these abstractions effectively. The method in Chapter 4 involves an agent that uses supervised learning to predict states and then learns to take actions that trade off improving its predictions with maximization of reward. In Chapter 5, the agent uses abstraction to navigate in areas where limited data is available in order to return to parts of the state space where it can follow a given behaviour policy. In all these works, the agent maintains several learning processes at once, securely maintaining its ground-truth representation but learning to benefit from abstraction when possible. The thesis concludes with Chapter 6, which summarizes the findings and contributions and poses future directions to extend this work.

REFERENCES

[1] J. Jackson, *IBM Watson Vanquishes Human Jeopardy Foes,* PCWorld (2011).

[2] A. M. Turing, *Computing Machinery and Intelligence,* MIND: A Quarterly Review of Psychology and Philosophy **236**, 433 (1950).

[3] I. H. Sarker, *Machine Learning: Algorithms, Real-World Applications and Research Directions,* SN Computer Science **2**, 1 (2021).

[4] A. Dickinson, *Contemporary Animal Learning Theory* (Cambridge University Press, 1980).

[5] L. J. Kamin, *Predictability, Surprise, Attention and Conditioning,* in *Symposium on Punishment* (Hamilton, Ontario, 1967).

[6] R. A. Rescorla and A. Wagner, *A Theory of Pavlovian Conditioning: Variations in the Effectiveness of Reinforcement and Nonreinforcement,* (Appleton- Century-Crofts, 1972).

[7] R. S. Sutton, *Learning to Predict by the Methods of Temporal Differences,* Machine Learning **3**, 9 (1988).

[8] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction,* IEEE Transactions on Neural Networks **9**, 1054 (1998).

[9] G. Tesauro, *Temporal Difference Learning of Backgammon Strategy,* in *Machine Learning Proceedings 1992*, edited by D. Sleeman and P. Edwards (Morgan Kaufmann, San Francisco (CA), 1992) pp. 451–457.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the Game of Go with Deep Neural Networks and Tree Search,* Nature **529**, 484 (2016).

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing Atari with Deep Reinforcement Learning,* arXiv (2013), arXiv:1312.5602.

[12] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, *Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning,* Nature **575**, 350 (2019).

[13] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, *Challenges of Real-World Reinforcement Learning: Definitions, Benchmarks and Analysis,* Machine Learning **110**, 2419 (2021).

[14] R. Bellman, *A Markovian Decision Process,* Indiana University Mathematics Journal **6**, 679 (1957).

[15] G. Neustroev, C. T. Ponnambalam, M. de Weerdt, and M. T. J. Spaan, *Interval Q-Learning: Balancing Deep and Wide Exploration,* Adaptive and Learning Agents Workshop at the 19th International Conference on Autonomous Agents and MultiAgent Systems (2020).

[16] N. K. Jong, T. Hester, and P. Stone, *The Utility of Temporal Abstraction in Reinforcement Learning,* in *Proceedings of the 7th International Conference on Autonomous Agents and MultiAgent Systems*, May (2008) pp. 294–301.

[17] N. K. Jong and P. Stone, *State Abstraction Discovery from Irrelevant State Variables,* in *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005) p. 752–757.

[18] A. Wachi and Y. Sui, *Safe Reinforcement Learning in Constrained Markov Decision Processes,* in *Proceedings of the 37th International Conference on Machine Learning* (2020) pp. 9797–9806.

[19] S. Fujimoto, D. Meger, D. Precup, O. Nachum, and S. S. Gu, *Why Should I Trust You, Bellman? The Bellman Error is a Poor Replacement for Value Error,* in *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162, edited by K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato (PMLR, 2022) pp. 6918–6943.

[20] L. Wells and T. Bednarz, *Explainable AI and Reinforcement Learning—A Systematic Review of Current Approaches and Trends,* Frontiers in Artificial Intelligence **4** (2021).

[21] S. Levine, A. Kumar, G. Tucker, and J. Fu, *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems,* (2020).

[22] J. Buckman, C. Gelada, and M. G. Bellemare, *The Importance of Pessimism in Fixed-Dataset Policy Optimization,* in *International Conference on Learning Representations* (2021).

[23] A. Subramanian, S. Chitlangia, and V. Baths, *Reinforcement Learning and its Connections with Neuroscience and Psychology,* Neural Networks **145**, 271 (2022), arXiv:2007.01099 .

[24] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, *Building Machines that Learn and Think like People,* Behavioral and Brain Sciences **40** (2017).

[25] I. Clavera, A. Nagabandi, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, *Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning,* in *International Conference on Learning Representations* (2019).

[26] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman, *Policy and Value Transfer in Lifelong Reinforcement Learning,* in *Proceedings of the 35th International Conference on Machine Learning,* Vol. 80, edited by J. Dy and A. Krause (PMLR, 2018) pp. 20–29.

[27] V. Pong, S. Gu, M. Dalal, and S. Levine, *Temporal Difference Models: Model-Free Deep RL for Model-Based Control,* in *International Conference on Learning Representations* (2018).

[28] E. M. Burgoon, M. D. Henderson, and A. B. Markman, *There Are Many Ways to See the Forest for the Trees: A Tour Guide for Abstraction,* Perspectives on Psychological Science **8**, 501 (2013).

[29] J. Fan, *An Information Theory Account of Cognitive Control,* Frontiers in Human Neuroscience **8**, 1 (2014).

[30] R. Lachman, J. L. Lachman, and E. C. Butterfield, *Cognitive Psychology and Information Processing: An Introduction* (Psychology Press, 1979).

[31] D. Kumaran, J. J. Summerfield, D. Hassabis, and E. A. Maguire, *Tracking the Emergence of Conceptual Knowledge during Human Decision Making,* Neuron **63**, 889 (2009).

[32] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, *From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning, Journal of Artificial Intelligence Research,* **61**, 215 (2018).

[33] D. Abel, D. E. Hershkowitz, and M. L. Littman, *Near Optimal Behavior via Approximate State Abstraction,* in *Proceedings of the 33rd International Conference on Machine Learning*, Vol. 48 (PMLR, New York, NY, USA, 2016) pp. 1–18.

[34] T. Dietterich, *State Abstraction in MAXQ Hierarchical Reinforcement Learning,* in *Advances in Neural Information Processing Systems*, Vol. 12, edited by S. Solla, T. Leen, and K. Müller (MIT Press, 1999).

[35] L. C. Cobo, K. Subramanian, C. L. Isbell, A. D. Lanterman, and A. L. Thomaz, *Abstraction from Demonstration for Efficient Reinforcement Learning in High-Dimensional Domains,* Artificial Intelligence **216**, 103 (2014).

[36] D. Abel, D. Arumugam, L. Lehnert, and M. Littman, *State Abstractions for Lifelong Reinforcement Learning,* in *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, edited by J. Dy and A. Krause (PMLR, 2018) pp. 10–19.

[37] K. Asadi, D. Abel, and M. L. Littman, *Learning State Abstractions for Transfer in Continuous Control,* CoRR (2020), 2002.05518 .

[38] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, *Conservative Q-Improvement: Reinforcement Learning for an Interpretable Decision-Tree Policy,* CoRR (2019), 1907.01180 .

[39] T. D. Simão, N. Jansen, and M. T. J. Spaan, *AlwaysSafe: Reinforcement Learning Without Safety Constraint Violations During Training,* in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (IFAAMAS, 2021) pp. 1226–1235.

[40] M. K. Ho, D. Abel, T. L. Griffiths, and M. L. Littman, *The Value of Abstraction,* Current Opinion in Behavioral Sciences **29**, 111 (2019).

[41] A. Solway, C. Diuk, N. Córdova, D. Yee, A. Barto, Y. Niv, and M. Botvinick, *Optimal Behavioral Hierarchy,* PLoS Computational Biology **10** (2014).

[42] R. A. Brooks, *Intelligence Without Representation,* Artificial Intelligence **47**, 139 (1991).

[43] W. A. Johnston and V. J. Dark, *Selective Attention,* Annual Review of Psychology **37**, 43 (1986).

[44] N. Sprague and D. Ballard, *Multiple-Goal Reinforcement Learning with Modular Sarsa(O),* in *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003) p. 1445–1447.

[45] S. Russell and A. L. Zimdars, *Q-Decomposition for Reinforcement Learning Agents,* in *Proceedings of the 20th International Conference on Machine Learning* (AAAI Press, 2003) p. 656–663.

[46] E. Even-Dar and Y. Mansour, *Approximate Equivalence of Markov Decision Processes,* in *Learning Theory and Kernel Machines*, edited by B. Schölkopf and M. K. Warmuth (Springer Berlin Heidelberg, Berlin, Heidelberg, 2003) pp. 581–594.

2

BACKGROUND

தொட்டனைத் தூறும் மணற்கேணி மாந்தர்க்குக்
கற்றனைத் தூறும் அறிவு.

- திருக்குறள் 396

The deeper a well is dug, the more the water that springs;
the more one learns, the more the wisdom it brings.

- Tirukkuṟaḷ 396

In this chapter, the foundational concepts upon which this work is built are introduced.

2.1. MARKOV DECISION PROCESSES

This work considers sequential decision making problems that can be modeled as Markov decision processes (MDPs) [1]. An MDP is a formal framework for solving sequential decision making problems that describes a discrete-time stochastic control problem.

An MDP is a tuple $M = \langle S, A, T, R, \gamma \rangle$ of the following variables:

S – the state space

A – the action space

T – the state transition probability function

R – the reward function

$\gamma \in [0, 1)$ – a discounting factor

A Markov decision process can define a problem with either discrete or continuous state and/or action spaces. At any time step t, the MDP M is in some state $s_t \in S$. Upon taking action $a_t \in A$, the MDP transitions to state s_{t+1} according to the transition function, which maps state-action pairs to a probability distribution over possible next states, $T : S \times A \times S \mapsto [0, 1]$. The reward function takes a transition (state, action, next state) and returns a real-valued number R_t, where $R : S \times A \times S \mapsto \mathbb{R}$. These functions

dictate the dynamics of a task, and the discounting factor γ is a property of the task that represents the value of taking an action now rather than later in the future. An MDP must retain the *Markov property* which declares that the future state is a function of only the present state and action and not any previous history. Formally, the following must be true:

$$T(s_{t+1}|s_t, a_t) = T(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...)$$

A solution to an MDP constitutes a *policy* $\pi(s)$ that defines, for each state, a probability distribution over actions. The expected return (called the *value*) of a policy is given by

$$V(\pi) = \mathbb{E}[\Sigma_{t=0}^{\infty} \gamma R_t | s_0, \pi],$$

where s_0 is an initial starting state. Often, one refers to the value of a particular state. This is then the sum of discounted rewards expected from following policy π beginning in state s_t:

$$V_\pi(s_t) = \sum_{s_{t+1}} T(s_{t+1}|s_t, \pi(s_t))(R(s_t, \pi(s_t), s_{t+1}) + \gamma V(s_{t+1})),$$

where actions are selected according to the policy. An optimal policy is one that maximizes this value for all possible states in S.

FACTORED MDPS

A factored MDP [2] is an MDP in which structure has been encoded through the use of a dynamic Bayesian network [3]. It offers a principled approach to abstraction in the form of compressing the transition function by breaking the state space into components. The result of approaching an MDP as a factored problem can be an exponential reduction in the problem space over which an optimal solution must be found [2].

In a factored MDP, the state is factored into state variables, thus the state space factored into k variables can be broken down into its k components:

$$S = S_1 \times S_2 \times ... \times S_k.$$

Each individual state s in the state space is defined by the value of its k state variables. The advantage comes in defining the transition function as a dynamic Bayesian network which describes the influence of taking an action on each state variable separately. The transition of a particular state variable may only depend on a subset of the state variables before it. This abstracts away components from transitions where they have no influence, allowing the transition function to be compactly represented. For solutions to solve factored MDPs efficiently, the reader is referred to [2, 4].

2.1.1. EXACT ALGORITHMS

Two popular approaches for optimally solving Markov decision processes are described here, suitable when both the state and action spaces are discrete, finite and of "reasonable" size (this is elaborated on further in the section). Several adaptations of these foundational methods have been developed, and continue to be, to solve MDPs that do not fit into these terms. In general, finding a solution to an MDP falls into the study of *planning* and is commonly approached with dynamic programming, in which solving the problem is broken into sub-steps and solved iteratively [5].

POLICY ITERATION

Policy iteration involves continuously evaluating and improving a policy until it converges to the optimal policy. The initial policy and value function are randomly initialized, and the policy is first evaluated by calculating its value. This is referred to as the *policy evaluation* step and is computed for every state $s \in S$, repeatedly iterating over the state space until convergence:

$$V_\pi(s) = \sum_{s'} T(s'|s, \pi(s))(R(s'|s, \pi(s)) + \gamma V(s')), \tag{2.1}$$

where $\pi(s)$ is the action suggested by the current policy in state s. Equation 2.1 is an example of what is known as a Bellman equation.

The *policy improvement* step then computes a new policy that selects the best action according to the newly calculated value function, again over all states $s \in S$:

$$\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s'|s, a)(R(s'|s, a) + \gamma V(s')) \tag{2.2}$$

The policy evaluation and improvement steps are continuously performed until convergence, at which point the policy and value function do not change. For a finite MDP, policy iteration is guaranteed to converge to an optimal policy [6].

VALUE ITERATION

In value iteration [7], the process of policy evaluation which generally requires multiple sweeps over the state space is truncated. The following computations are performed on an arbitrarily initialized value function that combine this truncated step with policy improvement:

$$V_{t+1}(s) = \max_a \sum_{s'} T(s'|s, a)(R(s'|s, a) + \gamma V_t(s')), \tag{2.3}$$

where this is performed over all $s \in S$ only once in each iteration step t. While this is theoretically guaranteed to converge to the optimal value function only after an infinite number of steps, in practice, the algorithm is stopped when the value function changes only a small degree between iterations [6]. The optimal policy is then the one which maximizes this value in every state.

2.1.2. CHALLENGES

Both of the algorithms described above are tabular approaches that maintain the value function in tabular form. This limits them to discrete MDPs of manageable size, as the memory requirements for this table can be high and quickly become infeasible. The oft-mentioned *curse of dimensionality* describes the exponential increase in the state space with increase in the number of state variables [7]. This, combined with the multiplicative effect of the size of the action space can make it very computationally expensive (both in memory and time) to solve large MDPs optimally.

2.2. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a technique for approaching problems that can be modeled as MDPs, but where the transition and reward functions are unknown [6]. Through

experiences collected by an agent while interacting with the environment, some representation of an environment model is learned and used to choose actions that maximize long-term rewards. In *model-based RL* techniques, a model is built (in the form of estimations of the transition function T and reward function R) according to the experience gathered by an agent when interacting with the environment and used to predict the value of future states when making decisions. In *model-free RL*, instead of estimating the model and using offline planning techniques to solve the MDP, experience is used directly to estimate the value of taking any action from any state.

2.2.1. ALGORITHMS

The field of algorithms for reinforcement learning is a rich and dynamic one, to which this thesis aims to contribute. Again, choosing the most suitable algorithm depends on a number of factors such as whether the state or action spaces are discrete or continuous, how large they are, how sparse the reward function is, whether there are constraints on the accumulated reward, and so on. Only some of the most popular and general algorithms are detailed here as they are most relevant to later chapters.

Q-LEARNING

While early methods such as TD-learning centered around calculating state values, Q-learning isolates calculating the value of each state-action pair [8]. The state-action value is referred to as the Q-value $Q(s,a)$. At time t, upon taking an action a_t according to some policy in state s_t, the agent receives a reward r_{t+1} and enters the next state s_{t+1}. In Q-Learning, these values are used to update $Q(s_t,a_t)$ through the update:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1},a)) \tag{2.4}$$

where α is the learning rate.

Under the following conditions, Q-learning is guaranteed to converge to the optimal values [9, 10]:

- Every state-action pair is visited infinitely often.
- The learning rate α approaches zero at an appropriate rate.

Practically, there becomes some point at which it may be favourable to take actions which can already be considered good. The question of when to execute actions in order to gain information (and get closer to the true Q-value) and when to execute actions to maximize reward according to the information you already have is called the *exploration-exploitation* dilemma and occurs persistently in reinforcement learning. A common approach to this dilemma is to take a random action with probability ϵ and otherwise take the action with the highest Q-value. This is called an ϵ-greedy policy. The parameter ϵ is often set according to a schedule that decreases it over timesteps, approaching zero.

Q-learning uses a tabular representation of the Q-value function, thus is extremely susceptible to the curse of dimensionality detailed above. As with other tabular methods, it also has no generalization capability and cannot infer anything about state-action pairs it has never visited. For problems with very large state-action spaces, it can quickly

be infeasible to use tabular methods such as Q-learning. In these cases, it becomes necessary to use methods that employ function approximation.

DEEP Q-LEARNING

The use of deep Q-networks uses a neural network representation of the Q-value function, enabling reinforcement learning to be applied to rich and complex state spaces such as that of Atari games [11]. In deep Q-learning (DQN), the Q-table is replaced with a neural network that predicts action-values from state inputs. The problem becomes a regression problem where the target value is:

$$y = R(s,a) + \gamma \max_{a'} Q(s', a' : \theta') \tag{2.5}$$

and θ' refers to the weights of the neural network Q. The regression problem is to minimize the difference between this and the predicted value given as $\hat{y} = Q(s, a : \theta)$. Notice that the parameters (weights) of the estimated $\hat{y}$ are not the same as the network that gives the Q-values of the next state-action (called the target network). In practice, using two different neural networks for the target and prediction Q-values is done to reduce the effects of non-stationarity on the stability of the method. The non-stationarity arises as the regression task is done on a iteratively updated target. By freezing the parameters of the target network and updating them only periodically, the method becomes more stable. Another technique to improve stability is the use of a replay buffer, which stores experience and is randomly sampled from periodically in order to obtain a batch of samples for the Q-network update. Deep Q-learning has many variants and iterations, and remains a popular approach to discrete-action problems with large state spaces.

POLICY GRADIENT METHODS

In policy gradient methods, the policy is directly represented as a function parameterized by some parameter ω and referred to as π_ω. The reward function according to the policy is defined as:

$$J(\pi_\omega) = \sum_{s \in S} \rho^{\pi_\omega}(s) V(s), \tag{2.6}$$

where $\rho^{\pi_\omega}(s)$ is the stationary distribution induced by the Markov chain resulting by following the policy. This can also be represented as $\rho^{\pi_\omega}(s) = \lim_{t \to \infty} P(s = s_t | s_0, \pi_\omega)$, or the probability that you end up in state s after following policy π_ω from initial state s_0 for an infinite number of time steps. Equation 2.6 can be expanded into:

$$J(\pi_\omega) = \sum_{s \in S} \rho^{\pi_\omega}(s) \pi_\omega(a|s) Q^{\pi_\omega}(s,a). \tag{2.7}$$

The general idea of all policy gradient methods is to compute the gradient $\Delta_{\pi_\omega} J(\omega)$, and update the parameter ω to move it in the direction suggested by the gradient that produces the highest reward.

One popular policy gradient method is REINFORCE, also called Monte-Carlo policy gradient [12]. REINFORCE works by simulating trajectories in order to calculate an accurate gradient of the policy. The gradient is the expectation of the sample gradient:

$$\Delta_{\pi_\omega} J(\pi_\omega) = \mathbb{E}_{\pi_\omega}[Q^{\pi_\omega}(s_t, a_t) \Delta_{\pi_\omega} \ln \pi_\omega(a_t | s_t)], \tag{2.8}$$

where $Q^{\pi_\omega}(s_t, a_t)$ is the expected return G_t received by following the policy from state s_t (starting with action a_t) for k time steps (a hyperparameter). By simulating the trajectory of k steps, one can calculate G_t, then update ω according to:

$$\omega \leftarrow \omega + \alpha\gamma G_t \Delta_{\pi_\omega} \ln \pi_\omega(a_t|s_t), \tag{2.9}$$

where α is the learning rate and γ the discount factor.

Another family of policy gradient methods is Proximal Policy Optimization (PPO) [13], which improved on methods such as REINFORCE. PPO consists of methods that are considerably easier to tune than other policy gradient methods while maintaining high performance. There are several versions and modifications of PPO, but one of the first implementations was notable for applying clipping, which encourages the gradient update to produce a policy that is close to the previous policy, limiting how much variance can occur in training thus being quite stable.

For an in-depth look at policy gradient methods, the reader may refer to [6].

2.2.2. OFFLINE REINFORCEMENT LEARNING

The reinforcement learning approaches described above are methods of *online reinforcement learning*. In this setting, the agent interacts with the environment at the same time as improving its policy (or model, in the case of model-based learning). One of the main issues with RL is the need to interact exhaustively with the environment. Offline reinforcement learning aims to address mainly this issue. In the offline setting, the agent does not have access to a simulator or generative model, instead only to a data set D generated through interactions with an MDP. Without access to the MDP itself, the agent cannot use further interactions to improve its policy. The MDP that emerges from the given data is referred to as $\hat{M}$, where:

$$\hat{M} = \langle \hat{S}, \hat{A}, \hat{R}, \hat{T}, \gamma \rangle.$$

The goal in offline RL is to learn a policy π_D on $\hat{M}$ that maximizes the discounted rewards of the true MDP M. A policy learned by applying a naïve online RL algorithm to D is likely to over-estimate the value of state-action pairs that are not well-represented in the data set [14]. To address this, recent methods aim to steer the learned policy away from such areas.

2.3. ABSTRACTION IN MDPS

In the loosest sense, abstraction suggests the re-representation of information in a way that compacts it but maintains (enough) meaning. While there is no single accepted definition, one that is aligned with the ideas and context presented here is given below [15].

Definition 1 (Abstraction)**.** An abstraction operation changes the representation of an object by hiding or removing less critical details while preserving desirable properties. By definition, this implies loss of information.

Abstraction in reinforcement learning is often considered in one of two categories: state abstraction and temporal abstraction [16]. By concept, abstraction is a general term

that can be applied to many different techniques outside of these two categories, but for simplicity only the two are described in some further detail. State abstraction refers to how the state space itself is represented in the problem; it generally involves applying an abstraction mapping to the state space that maps it to a more compact space, then solving the smaller resulting problem induced by this mapping. Temporal abstraction refers to temporal extensions of actions, wherein taking an action triggers a sub-routine that executes subsequent lower-level actions. This is the approach taken in the development of skills [17] and in other hierarchical approaches. With regards to particular MDPs, abstraction is generally considered as one of two kinds. In *state abstraction*, the abstraction operation is applied to the state space. In *action abstraction*, the abstraction is in the form of *temporally-extended actions* [18] that potentially consist of multiple actions joined together in a sequence [19]. Of the two types, this work focuses solely on state abstraction.

2.3.1. STATE ABSTRACTION

The goal of state abstraction is to compresses a finite state space into a smaller state space. This results in a smaller problem that allows faster or more efficient computation of a policy which can then be applied to the original problem [20]. A state abstraction ϕ applied to an MDP M is a mapping from the original state space to an abstract state space, $\phi : S \mapsto \bar{S}$. This mapping takes a state s and returns the corresponding state in the abstract state space $\phi(s) = \bar{s}$. The inverse mapping $\phi^{-1}(\bar{s})$ returns all the ground states that map to a particular abstract state.

The result of applying the abstraction mapping ϕ to the original MDP is the abstract MDP $\bar{M} = \langle \bar{S}, A, \bar{R}, \bar{T}, \gamma \rangle$ where the original action space and discount factor are maintained. The original MDP M is commonly referred to as the *ground MDP*. In the abstract MDP, the reward and transition functions are a weighted sum of those pertaining to the ground states that map to a corresponding abstract state.

TYPES OF STATE ABSTRACTION

There are several types of properties of the original MDP that can be preserved by an abstract MDP that have theoretical implications on the quality of the solution when it is applied to the original MDP. The referred-to definition of a state-abstraction type is as follows ([19]):

Definition 2 (State-abstraction type)**.** A state-abstraction type is a collection of functions $\phi : S \mapsto S_\varphi$ associated with a fixed predicate on state pairs:

$$p : S \times S \mapsto \{0, 1\},$$

such that when φ clusters state pairs in MDP M, the predicate must be true for that state pair:

$$\phi(s_1) = \phi(s_2) \implies p(s_1, s_2).$$

A predicate is a condition on ground states which is preserved by the abstraction. From [19], an example of a predicate is:

$$p_Q(s_1, s_2) \triangleq \max_a |Q^*_M(s_1, a) - Q^*_M(s_2, a)| = 0.$$

This requires that all states merged to the same abstract state share optimal Q-values. This can be called a Q^*-value preserving or Q^*-irrelevance abstraction.

Commonly found predicates in literature are the following, listed in order of increasing coarseness (coarser abstractions have looser requirements and merge, in general, more states) [20]:

- Model-preserving - merges states which share a one-step model (one-step transition and reward).
- Q-value-preserving - merges states which share the same values for all actions.
- Q^*-value-preserving - merges states which share the same value for optimal actions.
- a^*-value-preserving - merges states which share an optimal action and its value.
- π^*-preserving - merges states which share an optimal action (but not necessarily its value).

Note that guaranteeing that a predicate is met requires access to at least part of the solved value function of the MDP.

Theoretical Implications As mentioned, abstraction implies a loss of information. Ideally, this is a loss of irrelevant information, preserving the minimum required detail to sufficiently solve a particular task. Depending on the predicate preserved in a state abstraction, the ability to recover an optimal policy (optimal in the ground MDP) from the abstract MDP may suffer. From the list of state abstractions above, only the model, Q-value, and Q^*-value preserving abstractions are guaranteed to produce an optimal policy (in the ground MDP) under normal conditions [20].

REFERENCES

[1] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Series in Probability and Statistics (John Wiley & Sons, Inc., Hoboken, NJ, USA, 1994).

[2] C. Boutilier, R. Dearden, and M. Goldszmidt, *Stochastic Dynamic Programming with Factored Representations,* Artificial Intelligence **121**, 49 (2000).

[3] T. Dean and K. Kanazawa, *A Model for Reasoning about Persistence and Causation,* Computational Intelligence **5**, 142–150 (1989).

[4] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, *Efficient Solution Algorithms for Factored MDPs,* Journal of Artificial Intelligence Research **19**, 399–468 (2003).

[5] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. (Athena Scientific, 2018).

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. (The MIT Press, Cambridge, MA, USA, 2018).

[7] R. Bellman, *A Markovian Decision Process,* Indiana University Mathematics Journal **6**, 679 (1957).

[8] C. J. C. H. Watkins and P. Dayan, *Q-Learning,* Machine Learning **8**, 279 (1992).

[9] C. J. C. H. Watkins, *Learning From Delayed Rewards*, Ph.D. thesis, King's College, Cambridge, United Kingdom (1989).

[10] T. Jaakkola, M. I. Jordan, and S. P. Singh, *On the Convergence of Stochastic Iterative Dynamic Programming Algorithms,* Neural Computation **6**, 1185 (1994).

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing Atari with Deep Reinforcement Learning,* arXiv (2013), arXiv:1312.5602.

[12] R. J. Williams, *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,* Machine Learning **8**, 229 (1992).

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms,* arXiv (2017), arXiv:1707.06347, 1707.06347 [cs.LG] .

[14] J. Buckman, C. Gelada, and M. G. Bellemare, *The Importance of Pessimism in Fixed-Dataset Policy Optimization,* in *International Conference on Learning Representations* (2021).

[15] M. Ponsen, M. E. Taylor, and K. Tuyls, *Abstraction and Generalization in Reinforcement Learning: A Summary and Framework,* in *Adaptive and Learning Agents*, edited by M. E. Taylor and K. Tuyls (Springer-Verlag, Berlin, Heidelberg, 2010) pp. 1–32.

[16] M. K. Ho, D. Abel, T. L. Griffiths, and M. L. Littman, *The Value of Abstraction,* Current Opinion in Behavioral Sciences **29**, 111 (2019).

[17] G. Konidaris and A. Barto, *Efficient Skill Learning Using Abstraction Selection,* in *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009) p. 1107–1112.

[18] R. S. Sutton, D. Precup, and S. Singh, *Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning,* Artificial Intelligence **112**, 181 (1999).

[19] D. Abel, D. Arumugam, L. Lehnert, and M. Littman, *State Abstractions for Lifelong Reinforcement Learning,* in *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, edited by J. Dy and A. Krause (PMLR, 2018) pp. 10–19.

[20] L. Li, T. J. Walsh, and M. L. Littman, *Towards a Unified Theory of State Abstraction for MDPs,* in *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics* (2006) pp. 531–539.

3

Parallel Learning in Abstractions

செய்வினை செய்வான் செயன்முறை அவ்வினை
உள்ளறிவான் உள்ளம் கொளல்.
- திருக்குறள் 677

One who is excellent at execution, executes after
grasping the subtleties of all tasks from the experts.
- Tirukkuṟaḷ 677

There are several drawbacks to reinforcement learning (RL) that act as barriers to its wide-spread implementation. These issues are particularly relevant in real-world problems which are often high-dimensional and have complex dynamics. One of the downfalls of RL agents is that they require numerous interactions with the environment, sometimes making fatal errors, in order to learn to make optimal or near-optimal decisions from any state. This limits the applicability of RL agents in real-world problems where safety and cost are paramount and random or bad actions must be minimized. In discrete domains in particular, where learning is done in tabular form, reinforcement learning agents do not have the ability to generalize over features in the state space. This results in learning processes where all states and actions need to be exhaustively visited.

State abstraction has been posited as a way to reduce the size of a the problem, providing exponential benefits in terms of samples and computation required to finding a good solution [1]. However, this generally means learning a single abstracted representation of the MDP. Humans, on the other hand, are capable of selectively abstracting information as needed during decision-making [2]. One way to apply a similar ability to reinforcement learning would be to break down the MDP into several MDPs relating to each task and abstraction, however doing this requires the reward function to be decomposed into local reward functions. This is difficult to do by hand and requires extensive expert knowledge, whereas it may be the case that only a global reward func-

tion is available. In addition, devising abstractions often requires the optimal solution to the MDP [3]. In reinforcement learning, data is generally collected during the learning process and not available beforehand, defeating this possibility.

There are many scenarios where different information in an observation is relevant at different times in an overarching task. Consider an agent that must navigate to a key, pick it up, and then navigate to a door and unlock it. While looking for the key, the door location is not needed for decision-making and unnecessarily increases the state space. Similarly, after the agent has picked up the key, it is no longer relevant where the key was before it was picked up. Traditional reinforcement learning requires the agent to use the joint space of all potentially relevant information in its planning. Hierarchical and other modular reinforcement learning methods aim to break up problems into sub-tasks, speeding up learning via abstraction and facilitating transfer of knowledge. However, such methods typically require detailed specification of the sub-tasks, including their entry and termination states, decomposition of the reward function and careful hand-tuning of parameters in order to be used effectively. It is difficult to rely on humans to perfectly define these properties, especially for large and complex problems.

The parallel learning approach introduced here provides the benefits of hierarchical and modular learning without the need for highly-crafted deployment. State abstraction is applied in a novel way, where, rather than seeking a single abstraction of the environment, it is assumed that relevant state information varies throughout the task, thus several abstractions can be composed. By reducing many of the usual constraints on abstraction definition, the method allows for input of an imperfect human-defined abstraction mapping, consisting only of a mapping of states to abstract states. Learning in the abstractions and learning how to use the abstract policies is done using only the global reward from the environment as feedback. The aim is to exploit both given human knowledge and the modular structure inherent in the problem to address issues such as high sample requirements and inflexibility. In tasks with certain properties (discussed in the theoretical analysis), this method can make efficient use of experience and reach a total reward threshold in fewer learning episodes than traditional methods, as well as facilitate the transfer of modules between related tasks.

One of the benefits of taking a modular approach is the potential for transferring knowledge in order to bootstrap learning on a new task. Transfer learning forms a large field of research populated mainly with heuristic methods with limited formal guarantees [4]. The idea is to transfer learning from one task (the source task) to another related, but different task (the target task). The aim can be to improve any of several different facets of learning in the target task, such as performance of the initial policy or number of samples required to reach an optimal policy. The problem of transfer learning is closely related to that of abstraction as the goal is to identify which parts of the policy or model generalize to new problems and which should be overwritten with new knowledge. However, even if these parts are known, the problem of re-learning parts of the new task without strong assumptions on the value function or transition function is non-trivial. Simply transferring the policy or value function from one task to another can result in *negative transfer*, where learning in the new task is hindered by the constraints or inaccuracies imposed by this information, as the agent does not freely explore parts of the problem space that may be necessary to learn optimally in the new task. The par-

allel learning approach avoids the possibility of negative transfer through the use of a meta-agent that learns where to use the transferred policy (if at all).

In this chapter, a novel approach to state abstraction is defined that facilitates the use of simple human-definable abstraction mappings in a reinforcement learning algorithm. The method that exploits abstractions of this type is then introduced that allows for the input of several abstraction mappings. This method can also be used to learn in which states the abstractions apply (if this information is not given) in conjuction with solving the MDP. A theoretical analysis follows that explores the optimality conditions of the method and the conditions of improvement over a flat RL approach. In experiments, it is shown that not only can gains be made when the states in which the abstraction mappings apply are known, gains can also be made when no previous knowledge about when to use the given abstractions is provided. Finally, using this approach to conduct transfer learning is demonstrated in experiments.

3.1. FORMALISMS

The task or problem to be solved is modeled as a Markov decision process (MDP) $M = \{S, A, R, T, \gamma\}$. This will also be referred to as the original or ground MDP. Recall that, as in any reinforcement learning problem, the transition T and reward R functions are unknown. The goal is to learn a policy, a mapping from states to actions, that maximizes long-term reward.

3.1.1. STATE-CONDITIONED ABSTRACTION

Traditionally, finding a state abstraction mapping requires defining the predicate which it preserves; for example, an abstraction formed by merging states that share the same Q-values. This generally requires a series of comparative tests to the known value function of the ground MDP. This is a computationally expensive procedure; even finding an approximate abstraction for a particular predicate is an NP-Hard problem [5]. Further, in the reinforcement learning setting, the agent only has access to a generative model (or simulator), and not to the full MDP itself. These difficulties imply that hand-crafting such a precise mapping would be nearly impossible. This thesis is interested in making use of abstraction mappings that *can* be defined by a human, where they carry semantic meaning to an expert with knowledge of the problem.

To be able to make use of human-definable abstractions, state abstraction is reframed in a way that decouples the predicate and the mapping. Rather than seek a mapping that preserves a predicate over the entire state space, the mapping is given and the states where the predicate holds are sought. This is a new concept introduced here as a state-conditioned state abstraction, which consists of a mapping, a predicate, and the set of states in which the applied mapping preserves the predicate. The term state-conditioned abstraction is used synonomously with state-conditioned state abstraction from here forward for conciseness and to remain general to the case of action abstraction, though only abstracting the state space is considered in this work.

Definition 3 (State-conditioned abstraction)**.** A state-conditioned abstraction $\langle \phi, p, S_p \rangle$ is defined in relation to a particular MDP M. It contains a single abstraction mapping ϕ, a predicate p, and a set of states $S_p \in S$ in which the mapping preserves the predicate.

When ϕ clusters state pairs in MDP M, the predicate must be true for each state pair in S_p:

$$\phi(s_1) = \phi(s_2) \implies p(s_1, s_2) \forall s_1, s_2 \in S_p \,.$$

Consider a discrete MDP with state space S. When considering state-conditioned abstractions, the problem of finding an abstraction changes from computing the abstraction mapping to learning where a given mapping applies. We first look at the former case where the aim is to find an abstraction mapping that, applied to the entired state space, preserves the Q-values of the optimal action in the states it merges. Assuming that we have the optimal value function, making a comparison requires checking the optimal Q-value of two states (or clusters, as they are formed) to see if they can be merged. In the scenario where no two states share optimal Q-values, the number of comparisons that must be made is $\frac{|S|!}{2!(|S|-2)!}$, as every state must be compared to every other state (except itself) once (the equivalent of $\binom{|S|}{2}$).

In the alternate case where state-conditioned abstraction is applied, a candidate abstraction mapping ϕ is given that maps the state space S to abstract state space $\bar{S}$, where $|\bar{S}| < |S|$. Here the goal is to find the states where this mapping merges states that share optimal Q-values. Comparisons are made between states merged into abstract states. Only a single contradiction is needed before all the merged states can be dismissed. Thus, in the same scenario where no two states satisfy the predicate, only $|\bar{S}|$ comparisons are required. In the other extreme where all states satisfy the predicate, finding an exact abstraction requires $|S|-1$ comparisons, while finding the states where a proposed mapping is satisfied requires at most $|S|-1$ comparisons. This is not surprising, as in the case of state-conditioned abstraction, it is unlikely to end up with an abstraction that accomplishes maximum compression of the state space. However, there is still merit to taking this approach, as it may be infeasible to find an optimal abstraction due to the high amounts of computation and knowledge required.

If the set of states S_p is known, the state-conditioned abstraction mapping can be put into the terms of an abstraction mapping definition ϕ' applied over the entire state space (all $s \in S$), given by:

$$\phi'(s) \equiv \begin{cases} \phi(s) & \text{if } s \in S_p \,, \\ s & \text{otherwise} \,. \end{cases}$$

This produces the abstract MDP from the ground MDP and state-condition abstraction.

3.1.2. OPTIMAL-POLICY-PRESERVING ABSTRACTIONS

The coarsest form of state abstraction is considered to be optimal-policy-preserving abstraction (which merges states that share an optimal policy), particularly because it is agnostic of both the model and value function of an MDP [1]. All finer state abstraction types are optimal-policy-preserving, making it particularly attractive to devise methods that maintain optimality even in the case of only optimal-policy-preservation. However, in theoretical literature, the interest using state abstractions which do not have strict guarantees, such as these, has been limited. This is likely because when no guarantees

can be made, the performance of a policy learned on such an abstraction of the problem could be arbitrarily poor [6]. Even so, finer (and more exact) abstraction types require extensive knowledge of the problem dynamics or data resulting from exhaustive interactions with the MDP. For many problems, this can be simply impractical or even impossible to do.

In the case of an optimal-policy-preserving abstraction, the predicate is:

$$\pi^*(s_1) = \pi^*(s_2)$$

When the policy π^* is deterministic, as assumed in this chapter, this implies that $\pi^*(s_1) = \pi^*(s_2) = a^*$, i.e., both states share an optimal action (but not necessarily the value of the action). In the previously mentioned door and key problem, if the agent needs to pick up the key before going to the door, then regardless of whether the door is at point A or B, the best action is to go toward the key. This is intuitive and fairly easy for a human to define. In contrast, knowing that going toward the key will have the exact same long-term reward no matter the location of the door is not so intuitive. Optimal-policy-preserving abstractions do not require detailed knowledge of the value function or model.

3.2. PARALLEL Q-LEARNING IN ABSTRACTIONS

In the parallel learning in abstractions approach, the aim is to take advantage of abstractions that are *at least* optimal-policy preserving. The method takes a set of abstraction mappings and uses them to solve an MDP in a way that is both sample efficient and modular. The agent conducts learning in several abstract problems in parallel, efficiently using experience to update multiple local Q-value functions at once, each one representing a piece of the original (ground) MDP. Learning in abstract MDPs is known to introduce a number of potential problems that make recovering an optimal policy impossible in many cases [7]. In this work, two key things mitigate these issues: 1) learning in the ground MDP is done in parallel at all times, preserving the ground truth, and 2) the use of a novel Q-update function guarantees that the abstract modules will never overestimate the true value of an action, ensuring that they are used only to boost learning when possible.

The proposed parallel learning approach is built on top of the Q-learning framework in a method referred to as Parallel Q-Learning in Abstractions (QLiA). Under certain conditions, Q-learning is guaranteed to converge to the optimal policy, providing a solid foundation for this approach. The method requires a set of J candidate abstraction mapping functions $\Phi = \{\phi_1, \phi_2, ..., \phi_J\}$ provided by an expert. In the states where the given abstraction mappings are optimal-policy-preserving, the optimal policy can be learned efficiently in the abstract MDP. When the relevant states S_p for each abstraction mapping are unknown, a meta-agent uses reinforcement learning to learn how to compose the policies of each abstraction agent using global reward as feedback. This is the main case considered for this method, as flexibility is maintained and the optimal policy will be recovered even when the given abstractions are not optimal-policy-preserving. Later sections discuss how the approach works if these relevant states are known, and how sample complexity can be improved further in this case (with some caveats).

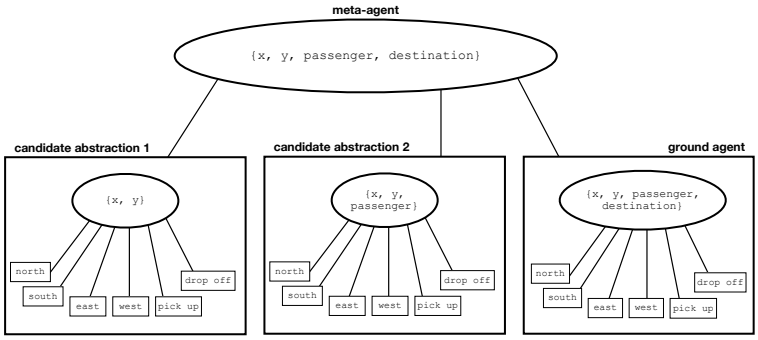

Figure 3.1: Parallel learning in the taxi domain. Though this approach could also provide an opportunity to abstract the actions available to each abstraction module, this work considers that they share the same actions available to the ground truth module.

3.2.1. CASE 1: INCOMPLETE ABSTRACTIONS

When the relevant states S_p for each abstraction mapping are unknown, a reinforcement learning agent (the meta-agent) is tasked with learning which abstract policy to execute at every state in the state space. The meta-agent models this task MDP where the state space is the ground state space and the action space is the set of abstraction choices. The number of abstraction choices is equal to the number of candidate abstractions J plus 1; this additional candidate is a trivial mapping from each state to itself (so it is *not* an abstraction). This trivial mapping, the ground truth, is referred to as ϕ_M. The action space of the meta-agent is $\mathrm{A}_\phi = \{A_{\phi_1}, A_{\phi_2}, ..., A_{\phi_J}, A_{\phi_M}\}$. Each action, including the ground truth, corresponds to another RL learning problem which has a local state space formed by applying its abstraction mapping. In each of these problems (also referred to as modules), the set of primitive actions A are those that are able to be executed in the environment.

A meta-agent's local MDP is defined as:

$$\hat{M} = \{S, \mathrm{A}_\phi, \hat{\mathrm{R}}, \hat{\mathrm{T}}, \hat{\gamma}\},$$

where $\hat{\mathrm{R}}$ maps state-abstraction-action pairs to real-valued rewards ($\hat{\mathrm{R}} : S \times \mathrm{A}_\phi \mapsto \mathbb{R}$) and $\hat{\mathrm{T}}$ maps them to a probability distribution over next states ($\hat{\mathrm{T}} : S \times \mathrm{A}_\phi \times S \mapsto [0,1]$). Each abstraction agent i exists in an abstract MDP:

$$\bar{M}_i = \{\bar{S}_i, A, \bar{R}_i, \bar{T}_i, \bar{\gamma}_i\}.$$

The abstract MDP $\bar{M}_i$ is the result of applying the candidate abstraction mapping ϕ_i to the ground MDP for all states.

Example The classical taxi problem was initially introduced in the context of hierarchical RL and continues to be used to demonstrate methods that exploit structure [8]. In

this problem, an agent in a grid world is tasked with picking up a passenger from a specified location and dropping them off at a specified destination. The state space spans the variables {`x, y, passenger location, destination location`}, with one of the passenger location values indicating that the passenger is in the vehicle. The actions available are: {`up, down, left, right, pick up, drop off`}. The agent gets a +20 reward for successfully dropping the passenger off at the correct destination and a -1 reward for each action taken; illegal `pick up` or `drop off` actions incur a -10 reward penalty. In Figure 3.1, the structure of the parallel learning approach to the taxi problem is shown. The meta-agent has three action choices; each action corresponds to an independent learning problem. Two candidate abstractions have been provided which see only a subset of the ground state, one that ignores both the passenger location and destination and another which ignores only the destination.

The QLiA system requires the meta-agent to learn which abstract policy to execute in each state, without knowing what the relevant states are. In Q-learning, at every time step, new experience is used to update Q-values corresponding to state-action pairs in a Q-table. In QLiA, multiple learning processes occur in parallel, thus multiple Q-tables are used. The entire algorithm, summarized in Algorithm 1, is described below.

The meta-agent Q-table $\hat{\mathrm{Q}}$ is of size $|S \times \mathrm{A}_\phi|$ and stores the value of executing a particular abstract policy in every state. An additional Q-table for each abstraction agent $\bar{\mathrm{Q}}_i$ ($i = 1, 2, ..., J$) has size $|\bar{S}_i \times A|$ and stores the value of executive primitive actions from each abstract state $\phi_i(s)$. The ground Q-table, which maintains the ground truth, is referred to as $\mathbb{Q}$. An ϵ-greedy policy π is used to choose actions; it takes a Q-table, state, action set and parameter ϵ and, with probability ϵ, returns a random action (drawn from a uniform distribution), or else returns the action with the highest Q-value for that state. The two-step action selection process, performed at every time step, is:

1. Given the state s_t and Q-table $\hat{\mathrm{Q}}$ the meta-agent makes an abstraction choice $a_t^\phi = \phi_k$ that indicates whether an abstract Q-table $\bar{\mathrm{Q}}_k$ or the ground Q-table $\mathbb{Q}$ (if k is equal to $J+1$) will be used to select the primitive action.

2. If an abstract Q-table will be used, s_t is mapped to abstract state $\bar{s}_t = \phi_k(s_t)$ and the Q-values for action selection are given by $\bar{\mathrm{Q}}_k(\bar{s}_t)$. If the ground Q-table will be used, $\mathbb{Q}(s_t)$ is used for action selection. Upon executing a_t in the environment, the agent receives a reward r_t and next state s_{t+1}.

The execution of an action and receipt of a reward and subsequent next state triggers a multi-step update process. The Q-tables of all abstraction candidates are updated after every action taken, regardless of which abstraction was used to choose the regular action, allowing experience to be shared amongst each module.

The novel update The most important distinction of the QLiA approach over other modular methods is the update used by the abstraction agents. Rather than performing a traditional off-policy update that uses the value of the next abstract state, the abstraction agents update their values according to the value of the *ground* next state. This is the key difference in the QLiA approach that mitigates the issues that otherwise occur if

each abstract problem is solved as an independent MDP. The implications of doing this are discussed further in the theoretical analysis in Section 3.4.2.

The two-step update process is:

1. For each candidate abstraction i, the experience $(\phi_i(s_t), a_t, r_t, s_{t+1})$ is used to update $\bar{\mathrm{Q}}_i$:

$$\bar{\mathrm{Q}}_i(\phi_i(s_t), a_t) \leftarrow \bar{\mathrm{Q}}_i(\phi_i(s_t), a_t) + \alpha(r + \gamma \max_{\phi} \hat{\mathrm{Q}}(s_{t+1}, \phi) - \bar{\mathrm{Q}}_i(\phi_i(s_t), a_t)).$$

   For the ground agent, the experience (s_t, a_t, r_t, s_{t+1}) is used to update $\mathbb{Q}$:

$$\mathbb{Q} \leftarrow \mathbb{Q}(s_t, a_t) + \alpha(r + \gamma \max_{a} \mathbb{Q}\, M(s_{t+1}, a) - \mathbb{Q}(s_t, a_t)).$$

   The two updates are not the same; the ground agent does a traditional off-policy update but the abstraction agents update their values according to the value of the ground next state.

2. The reward received is also used to update $\hat{\mathrm{Q}}$, using the state, next state and the abstraction action chosen. Thus, $\hat{\mathrm{Q}}$ is updated with the experience $(s_t, a_\phi, r_t, s_{t+1})$:

$$\hat{\mathrm{Q}}(s_t, \mu) \leftarrow \hat{\mathrm{Q}}(s_t, \mu) + \alpha(r + \gamma \max_{\phi} \hat{\mathrm{Q}}(s_{t+1}, \phi) - \hat{\mathrm{Q}}(s_t, \mu))$$

This method relies on a human expert providing abstraction mappings that maintain the predicate in parts of the problem, without guaranteeing that this will be the case as the agent will converge to an optimal policy even if it is not. If the provided mappings can be applied to parts of the state space, the abstraction agent will learn the optimal policy faster in these areas and the meta-agent will learn which abstraction agent chooses actions that lead to higher long-term reward. By this mechanism, applying QLiA can result in efficient learning. A key benefit of this approach is that the abstractions are coarse and do not need to hold for the entire MDP, being much easier for a human to define. This aligns more with the goals of hierarchical approaches which aim to break up the problem into sub-problems.

3.2.2. CASE 2: FULLY-SPECIFIED ABSTRACTIONS

In the previous section, the relevant states S_p for each potential optimal-policy-preserving abstraction mapping are not known. Instead, the meta-agent must learn to choose the abstractions that are optimal-policy-preserving in each state. In the case where the expert does provide the full specifications of a set of state-conditioned abstractions, each with a relevant state set $S_p = \{S_p^1, S_p^2, ...\}$ paired with an abstraction mapping $\Phi = \{\phi_1, \phi_2, ...\}$, there is no longer a need for the meta-learning layer. The trade-off is that this results in a more rigid approach that will suffer from any errors in the given specification and in which optimality is not guaranteed.

The fully-specified state-conditioned abstractions inform a fixed policy Π which selects the appropriate abstraction (from a set Φ) for every state:

$$\Pi(s) = \{\phi_i \in \Phi \,|\, s \in S_p^i\}.$$

Algorithm 1: Parallel Q-Learning in Abstractions (QLiA)

Input: Set of $J+1$ candidate abstraction mappings (including the full state agent) $\Phi = \{\phi_1, \phi_2, ..., \phi_J, \phi_M\}$
Input: Primitive actions A
Input: Abstraction-choice actions $A_\phi = \{\mu_1, \mu_2, ..., \mu_{J+1}\}$
Initialize parameters α, γ, ϵ, δ_α, δ_ϵ, H;
Initialize $\hat{Q}, \bar{Q}_1, ..., \bar{Q}_J, \mathbb{Q}$;
Set $t = 0$;
while $t < H$ **do**
  Observe state s_t;
  $\mu_t = \pi(\hat{Q}, s_t, A_\phi)$;
  $a_t = \pi(\bar{Q}_{\mu_t}, \phi_{\mu_t}(s_t), A)$;
  Take action a_t;
  Receive reward r and next state s_{t+1};
  for ϕ_i *where* $i \in \{1, ..., J\}$ **do**
    $\bar{Q}_i(\phi_i(s_t), a_t) \leftarrow \bar{Q}_i(\phi_i(s_t), a_t) + \alpha(r + \gamma \max_\phi \hat{Q}(s_{t+1}, \phi) - \bar{Q}_i(\phi_i(s_t), a_t))$
  end
  $\mathbb{Q} \leftarrow \mathbb{Q}(s_t, a_t) + \alpha(r + \gamma \max_a \mathbb{Q} M(s_{t+1}, a) - \mathbb{Q}(s_t, a_t))$;
  $\hat{Q}(s_t, \mu) \leftarrow \hat{Q}(s_t, \mu) + \alpha(r + \gamma \max_\phi \hat{Q}(s_{t+1}, \phi) - \hat{Q}(s_t, \mu))$;
  $t = t + 1$;
  $\alpha \leftarrow \delta_\alpha \alpha$;
  $\epsilon \leftarrow \delta_\epsilon \alpha$;
end

There is an assumption made that each state is uniquely assigned to the relevant state set of a single abstraction mapping. If it were the case that a state belongs to several mappings, an arbitration criterion can be used to choose which abstraction to favour (presumably the one that merges more states together, providing a more compact representation). The final result acts as a partition, where every state is represented in a set of relevant states, no state falls into more than one set of relevant states, and there are no empty sets of relevant states.

With fully-specified state-conditioned abstractions, it is no longer a requirement to include the ground agent in the abstraction agent candidate set. However, in the likely case that there exist states which are not a member of any abstraction-relevant state set, it is necessary to include a trivial abstraction which is a one-to-one mapping of the state space to an identical state space and whose relevant state set includes all these states. In the parallel updates, the max term is replaced with the maximum value of the next abstraction agent (that will be chosen by the fixed policy for the *next state*), in its corresponding abstract next state. The implications of no longer maintaining a ground truth Q-table are discussed in the next section.

The entire method of QLiA when abstractions are fully-specified is summarized in Algorithm 2.

Algorithm 2: QLiA with Fully-Specified Abstractions

Input: Set of $J+1$ candidate abstraction mappings (including the full state agent) $\Phi = \{\phi_1, \phi_2, ..., \phi_J, \phi_M\}$
Input: Abstraction selection policy Π
Input: Primitive actions A
Input: Abstraction-choice actions $A_\phi = \{\mu_1, \mu_2, ..., \mu_{J+1}\}$
Initialize parameters α, γ, ϵ, δ_α, δ_ϵ, H;
Initialize $\bar{Q}_1, ..., \bar{Q}_J$;
Set $t = 0$;
while $t < H$ **do**
  Get state s_t;
  Choose μ_t using $\Pi(s_t)$;
  Choose a_t using $\pi(\bar{Q}_k, \phi_t(s_t), A)$;
  Take action a_t;
  Receive reward r and next state s_{t+1};
  Get μ_{t+1} using $\Pi(s_{t+1})$
  for ϕ_i *where* $i \in \{1, ..., J\}$ **do**
    $\bar{Q}_i(\phi_i(s_t), a_t) \leftarrow \bar{Q}_i(\phi_i(s_t), a_t) + \alpha(r + \gamma \max_a \bar{Q}_{\mu_{t+1}}(\phi_i(s_{t+1}), a) - \bar{Q}_i(\phi_i(s_t), a_t))$
  end
  $t = t + 1$;
  $\alpha \leftarrow \delta_\alpha \alpha$;
  $\epsilon \leftarrow \delta_\epsilon \epsilon$;
end

3.3. EXPERIMENTAL EVALUATION

In this section, experiments are performed to assess QLiA both with unknown and fully-specified state-conditioned abstractions in two different domains. All environments are simple discrete grid world domains where the agent navigates using the four cardinal directions: up, down, left, right.

QLiA requires a set of candidate optimal-policy-preserving abstraction mappings provided by a human. In theory any type of abstraction mapping between two discrete state spaces can be applied, however, when a task is represented as a factored MDP there is an opportunity to define abstraction mappings based on subsets of state variables; this is more interpretable as the variables carry semantic meaning. This is the type of abstraction mapping provided in all the following experimental set-ups. In a factored MDP, an abstraction mapping can be defined as a subset of state variables.

Baselines QLiA is compared to a flat Q-learning agent in order to empirically explore the potential gains (or losses) in sample efficiency. Another baseline is a Q-learning agent learning conventionally in the abstract MDP of a single abstraction mapping. In the taxi domain, MAXQ is introduced as an additional baseline. MAXQ is a hierarchical RL approach that uses full specifications of both the sub-procedures and their relevant

(a) No inner walls impede left and right movement.

(b) An inner wall impedes some movement to the left and right.

Figure 3.2: Hallway environment where the agent gets reward for reaching a star.

state abstractions [8]. The learning parameters α, ϵ, and linear decay schedules for parameters α and ϵ were chosen by running each algorithm with all combinations from a discrete set of expert-chosen values and taking the values that resulted in the highest cumulative reward for each algorithm individually.

3.3.1. HALLWAY ENVIRONMENT

The domain in the first two evaluations is referred to as the hallway environment (pictured in Figure 3.2), wherein an agent must go all the way to the right of the hallway and take a final right action to receive a +10 reward and end the episode. All actions incur a penalty of -1 reward to execute. In the first version of this domain, the agent must remain on the grid but is otherwise not impeded. In the second version, there is a wall that impedes the left and right motion of the agent except in the center row. These two environments demonstrate how QLiA compares when the size of the relevant state sets for the abstraction candidate differ.

The hallway environment can be modeled as a factored MDP where each square on the grid is represented by its corresponding `row` and `column` coordinate. The abstraction provided to QLiA is a single candidate mapping which ignores the `row` variable, meaning it maps all states that differ only by their `row` value to the same abstract state (Case 1). The QLiA_specified agent is given the same abstraction mapping as well as the set of states in which it preserves the optimal policy (Case 2). In the domain shown in Figure 3.2a, all states share an optimal action (go right). However, in the domain in Figure 3.2b, the states directly to the left of the wall do not share this optimal action. Finally, the same abstraction is also applied to the MDP solved by the naïve Abstract Q agent. The initial domain is small, with a size of 5 rows and 10 columns. To demonstrate the value of the parallel learning approach in problems of larger sizes, an extra dimension of varying size is added to the state and/or the action space. This noisy dimension has no actual impact on the reward or dynamics of the environment but, because a tabular approach is taken, it makes the problem of learning the value function proportionally bigger and therefore the exploration requirements higher.

RESULTS

The results of applying QLiA to the hallway domain without an inner wall are pictured in the plots in Figure 3.3. The plots show regret against the number of learning episodes, where regret is the difference in total episode reward between an agent that behaves optimally and the indicated algorithm – lower regret signifies better performance. The naïve Abstract agent performs as well as the QLiA_specified agent, reaching optimal re-

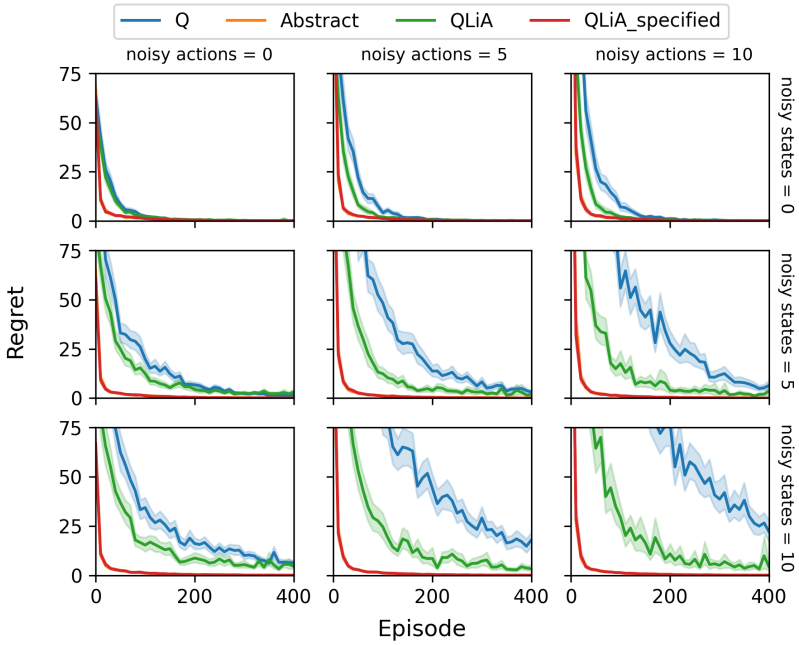

Figure 3.3: Results of running experiments on the hallway domain with no inner wall. 200 trials are averaged with the 90% confidence interval depicted as the shaded region. Note that the orange line (Abstract agent) is entirely behind the red line (QLiA_specified agent).

wards in few episodes. This is expected as the given abstraction is optimal according to the strict Q-value preservation criteria, where each merged state shares not only an optimal action but also the Q-values of all actions, and is therefore guaranteed to converge to an optimal policy [1]. This neat property does not hold in the other experiments. In the smallest state-action space, the Q agent performs on par with the QLiA agent. However, as the problem space grows, the benefits of using QLiA become more and more obvious, as the theoretical analysis revealed.

The hallway environment with an inner wall highlights the benefits that are possible even when the abstraction candidates provided are approximate or only hold in some states. The results of the experiments are shown in Figure 3.4. Here it is obvious that the naïve Abstract agent performs very poorly, and does not converge. Both cases of QLiA outperform the Q agent, particularly as the state-action space increases.

The policies learned by the meta-agent are plotted in Figure 3.5, for versions of the hallway domain where the wall is placed in different positions. The states directly to the left of the wall do not share an optimal action, as the agent must move to the gap in the wall in order to get to the reward. The black squares indicate the positions where the meta-agent has greater or equal preference for the abstraction agent over the ground agent. This visual demonstrates how the meta-agent learns in which states the abstraction is not suitable for choosing an optimal action (which states do not maintain the predicate).

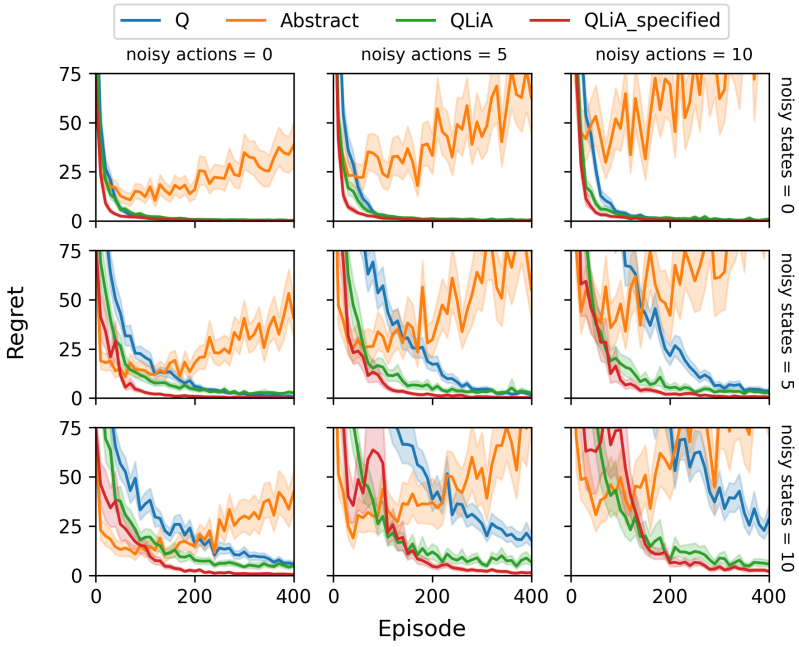

Figure 3.4: Results of running experiments on the hallway domain with an inner wall placed between the 3rd and 4th columns. 200 trials are averaged with the 90% confidence interval depicted as the shaded region.

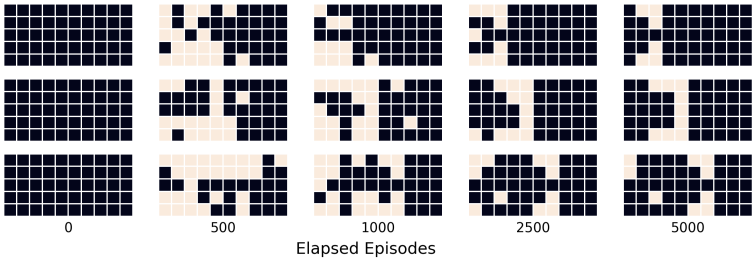

Figure 3.5: In the top row, the wall is between the 3rd and 4th columns, in the middle row between the 5th and 6th columns, and in the bottom row between the 7th and 8th columns. Squares depict the policy of the meta-agent where black indicates equal or greater preference of the abstract agent and white indicates preference of the ground agent.

3.3.2. TAXI

QLiA was applied to a stochastic version of the taxi problem, where there is a 20% chance of moving in a random perpendicular direction each time a directional action is taken before the passenger has been picked up and a 30% chance of the destination randomly changing when the passenger is picked up [9]. The QLiA agent is given a single abstraction mapping corresponding to the subset of state variables `{x, y, passenger}`, ignoring the destination. It is compared against a Q agent (with no abstraction), and MAXQ, which requires the full definition of the sub-procedures and their pertaining state abstractions by an expert. The algorithms were both tuned to bring out their best respective performance.

RESULTS

The experimental results are plotted in Figure 3.6. The biggest advantage of MAXQ is seen in the earliest few episodes, as it has a significantly higher initial reward. This huge gain is due to the limited action set available in each subtask, which keeps the agent from exploring the illegal penalty-inducing actions `pick up` or `drop off`. With the same knowledge, this could easily be incorporated by limiting the actions available to each module in QLiA. Even with extensive parameter tuning, it was not possible to have MAXQ reach the same top performance as the other algorithms in the same number of episodes.

There is no benefit seen to using QLiA with unspecified relevant states over standard Q-learning. The reduction in problem size under the QLiA architecture can be looked at analytically. The original problem is size 3000 (500 states $\times$ 6 actions) and QLiA has a problem size of 2200. This reduction appears to be too small to produce a visible benefit when applying QLiA if the relevant states are not known. However, it is worth noting that the performance did not falter even with the added learning layers of QLiA and the other benefits of the approach still stand, most notably that it empowers the reuse of a learned abstract module in a new related problem, as is discussed in the next section.

3.4. THEORETICAL DISCUSSION

Q-learning is guaranteed to converge under the assumption that each state-action pair is visited infinitely often and that the learning rate α approaches zero [10]. However, in the proposed method, the application of abstraction and the layered approach taken compromise the properties of traditional Q-learning. In this section, the theoretical implications of the parallel learning in abstractions approach are evaluated by investigating the following questions:

1. Does this approach maintain assumptions of stationarity?
2. Under which conditions is it guaranteed that the abstraction modules will learn the (or an) optimal policy?
3. Under which conditions is convergence to the optimal policy guaranteed?
4. Under which conditions is decreased regret over standard Q-learning expected?
5. How do these theoretical results change in the case of fully-specified subtasks (Case 2)?

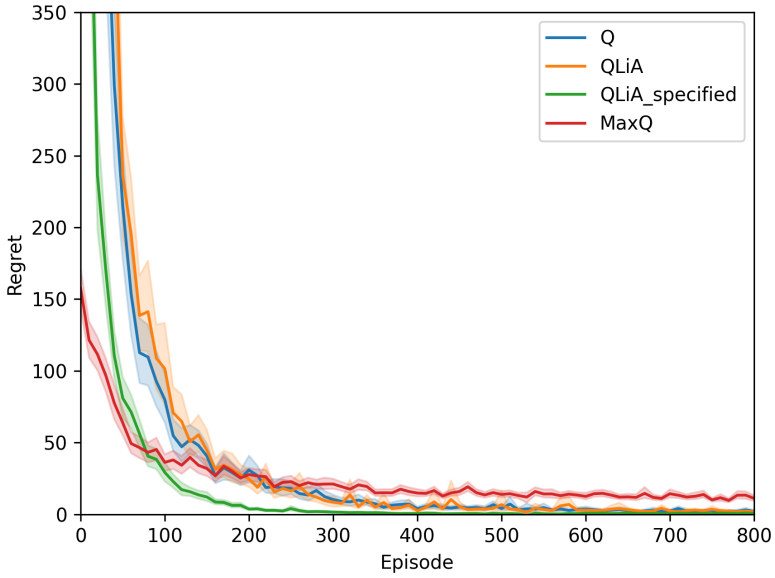

Figure 3.6: Results depicting the average of 200 trials on the stochastic taxi domain. Shaded region indicates 90% confidence interval.

3.4.1. NON-STATIONARITY

When the relevant states for each abstraction mapping are not fully-specified, QLiA involves two layers of simultaneous reinforcement learning. A meta-agent learns to optimize its choice of abstraction at the same time as the abstraction agents learn to optimize their choice of action *subject to* the distribution induced by the policy of the meta-agent. This breaks the assumptions of stationarity required for convergence of the Bellman equation to the optimal policy. However, the included ground agent is not subject to the same non-stationarity as it observes the full state description and is updated with a regular off-policy Q-learning update. In this section, it is shown that the meta-agent is guaranteed to converge to a policy where choosing the ground agent is the (or an) optimal action and that the ground agent itself is guaranteed to converge to the optimal policy. Thus, this source of non-stationarity in the abstraction agents is tolerated, knowing that with enough experience the system will converge to an optimal solution. In the case where subtasks are fully-specified, there remains a source of non-stationarity that occurs in abstract MDPs due to the clustering of states with different dynamics [11].

3.4.2. CONDITIONS OF MODULE OPTIMALITY

To investigate the optimality of the method, it is first considered whether the abstraction agents will learn meaningful policies both in merged states where the predicate holds and in merged states where it does not hold. If each abstract space is treated as an abstraction of the entire MDP, a typical Bellman update for an abstract state would be the following:

$$\bar{Q}(\phi_i(s)_t, a_t) \leftarrow (1-\alpha)\,\bar{Q}(\phi_i(s)_t, a_t) + \alpha(r(s_t, a_t) + \gamma \max_a \bar{Q}(\phi_i(s)_{t+1}, a))$$

There is a glaring issue here, which is that this assumes that the state space in the abstract MDP contains all the necessary information to produce a Markovian problem for which the optimal policy can be learned with Q-learning. However, the abstractions considered are not expected to maintain optimal-policy-preservation for the full problem, thus this update attempts optimize a partially observable problem – naïvely applying Q-learning to a partially observable MDP can result in an arbitrarily bad solution [6]. The loss of stationarity due to merging states with different state distributions has been cited as a major contributor to this issue [11].

This issue results in a potential to over-estimate actions due to the *max* operator which performs a max over the action values of the next abstract state. Therefore, in the parallel learning scheme, the max is applied according to the entire system, considering both the actions of the meta-agent as well as the possible actions of the other modules. This means the max is performed on the next state value according to the meta-agent:

$$\bar{Q}_i(\phi_i(s)_t, a_t) \leftarrow (1-\alpha)\,\bar{Q}_i(\phi_i(s)_t, a_t) + \alpha(r(s_t, a_t) + \gamma \max_{\phi} \hat{Q}(s_{t+1}, \phi))$$

This small change (highlighted in red) does several things. First, it means updates are made according to the value assigned to the *true* next state (i.e. the next state according to the flat representation), avoiding the repeated insertion of noise into the feedback loop that can occur if the next abstract state is used, and the resulting problems described above. Second, it means the distribution over next states that occurs as a conse-

quence of the meta-agent policy is embedded into the calculated Q-values. This second point requires further expansion.

Each abstraction module observes only a partial observation, thus each parallel Bellman update is performed on a different local state for each abstraction. The meta-agent policy sees the full observation and decides which abstraction will choose the executed action. The transition to the next (true) state is not only the consequence of the abstraction agent action, but also of the meta-agent action, which is a consequence of the current (true) state that the abstraction agent cannot observe. The definition of the Q-value for the abstraction agent i can be decomposed into:

$$\bar{\mathrm{Q}}_i(\phi_i(s), a) = \sum_{s^* \in \phi_i^{-1}(\phi_i(s))} \sum_{s'} \rho_i(s^* \mid \phi_i(s))\, T(s' \mid s^*, a)[R(s^*, a) + \gamma\, V(s')], \tag{3.1}$$

where $\rho_i(s^* \mid \phi_i(s))$ is the probability that the abstraction agent i is in the true state s^* given its local observation $\phi_i(s)$. This is determined by both the initial state distribution inherent in the environment and the behaviour of the QLiA agent. In contrast to the traditional definition of the Q-value, the sum is over all possible true current states in addition to summing over the next states. Recall that the inverse abstraction function $\phi_i^{-1}(\bar{s})$ returns all ground states that map to the same abstract observation $\bar{s}$:

$$\phi_i^{-1}(\bar{s}) = \{s \in S \mid \phi_i(s) = \bar{s}\}.$$

Putting this in terms of the traditional Q-value definition:

$$\bar{\mathrm{Q}}_i(\phi_i(s), a) = \sum_{s^* \in \phi_i^{-1}(\phi_i(s))} \rho_i(s^* \mid \phi_i(s))\, \mathbb{Q}(s^*, a). \tag{3.2}$$

First consider the states in $\phi_i^{-1}(\phi_i(s))$ that satisfy the optimal-policy-preserving predicate p_{π^*} (i.e., the states in S_p). This means an action a^* which optimizes the Q-value for any state in $\phi_i^{-1}(\phi_i(s))$ also optimizes the Q-value for all other states in $\phi_i^{-1}(\phi_i(s))$. While it is expected that the particular Q-values learned by the abstraction agent are not necessarily meaningful, the question of interest is whether the abstraction agents will indeed learn the optimal action for these states. Notation is simplified from this point by referring to an abstract state as $\bar{s}$, where $\bar{s} = \phi(s)$.

Theorem 1. *Consider any abstraction mapping function $\phi : S \mapsto \bar{S}$. It holds that for states that map to the same abstract state and that share an optimal action a^* in the ground MDP, this action a^* that maximizes the Q-value of state s in the ground MDP $\mathbb{Q}(s)$ also maximizes the Q-value of state $\bar{s}$ in the abstraction agent function $\bar{\mathrm{Q}}(\bar{s})$.*

Proof. By definition, an action a^* is optimal in state s if it maximizes the Q-value of state s:

$$\forall_a [Q(s, a^*) \geq Q(s, a)].$$

Under the conditions described above, each ground state given by the inverse mapping function $s^* \in \phi^{-1}(\bar{s})$ shares an optimal action a^*. Recall two basic properties of the argmax function.

Axiom 1. *If* $\underset{x}{\operatorname{argmax}} f(x) = \underset{x}{\operatorname{argmax}} g(x)$ *then*

$$\underset{x}{\operatorname{argmax}}(f(x)+g(x)) = \underset{x}{\operatorname{argmax}} f(x) = \underset{x}{\operatorname{argmax}} g(x).$$

Axiom 2. *If* $c > 0$, *then* $\underset{x}{\operatorname{argmax}} f(x) = c \underset{x}{\operatorname{argmax}} f(x)$.

From Equation 3.2, the Q-values of the abstraction agent for abstract state $\bar{s}$ are a linear combination of the true Q-values for each ground state s^* and a probability. As long as each next state is visited (thus the probability is non-zero), by the properties of the argmax function above, it holds that the action a^* that maximizes $\mathbb{Q}(s^*, a)$ for all $s^* \in \phi^{-1}(\bar{s})$ also maximizes $\bar{\mathrm{Q}}(\bar{s}, a)$. □

The above shows that the abstraction agent will learn the optimal action for abstract states that fall within the relevant set S_p. The case for states outside of S_p that map to a single abstract state but do not share an optimal action is considered in the next section, which investigates whether the entire system will produce an optimal solution, given that this second case is likely to emerge often.

3.4.3. CONDITIONS OF GENERAL OPTIMALITY

When merged states do not share an optimal policy, the action that maximizes the Q-values learned by the abstraction agents may be arbitrary. However, the set of abstraction agents always includes an agent which observes the full state and performs standard Q-updates. This section describes how with enough experience, the meta-agent will learn to favour this ground agent and that this agent will preserve the true optimal policy.

Theorem 2. *With sufficient experience, given any abstraction mapping function* $\phi : S \mapsto \bar{S}$*, the value assigned by the meta-agent to the action* a_{ϕ_i} *of choosing any abstraction agent* i *will not exceed the value of choosing the ground agent* a_M*. That is:*

$$\forall s \in S, \forall a_{\phi_i} \in \mathrm{A}_\phi [\hat{\mathrm{Q}}(s, a_{\phi_i}) \leq \hat{\mathrm{Q}}(s, a_M)]$$

Proof. The Q-value for a state s according to the meta-agent is, by definition:

$$\hat{\mathrm{Q}}(s, \phi_i) = \sum_{s'} T(s'|s, \phi_i)[R(s, \phi_i) + \gamma V(s')].$$

However, the reward and transition functions are defined over states and actions, not states and abstraction agents. Therefore, in the T and R functions, ϕ_i is rather the action that will be chosen by the abstraction agent, which is the action that optimizes its local Q-value:

$$\underset{a}{\operatorname{argmax}} \bar{\mathrm{Q}}_i(\phi_i(s), a).$$

This results in the revised definition:

$$\hat{\mathrm{Q}}(s, \phi_i) = \sum_{s'} T(s'|s, \underset{a}{\operatorname{argmax}} \bar{\mathrm{Q}}_i(\phi_i(s), a))\, R(s, \underset{a}{\operatorname{argmax}} \bar{\mathrm{Q}}_i(\phi_i(s), a)) + \gamma V(s'). \tag{3.3}$$

This is equivalent to the value abstraction agent i assigns to local state $\phi_i(s)$ when there is a fixed state s, meaning $\rho_i(s|\phi_i(s)) = 1$ (see Equation 3.2). The value is the state-action value of its highest valued action:

$$\hat{\mathrm{Q}}(s, \phi_i) = \max_a \bar{\mathrm{Q}}_i(\phi_i(s), a)$$

This is the case of the ground agent, where $\bar{s} = s = s^* = \phi^{-1}(s)$. By Equation 3.2, when this is true, then $\rho(s^*|\bar{s}) = 1$ and the converged Q-values of the ground agent match the true Q-values in the ground MDP. By the same equation, the maximum value any abstraction agent can assign to an action is the true value of taking the action in ground state of the true MDP. In the case of the ground agent, where $s = \phi_M(s)$ for all $s \in S$:

$$\bar{\mathrm{Q}}_M(\phi_M(s), a) = \mathbb{Q}(s, a).$$

With sufficient experience, the ground agent learns the true values of every (s, a) pair and the value the meta-agent can assign to any abstraction is upper-bounded by this value.

$$\forall\, \phi \in \mathrm{A}_\phi[\hat{\mathrm{Q}}(s, \phi)] \leq \bar{\mathrm{Q}}_M(\phi_M(s), a) = \mathbb{Q}(s, a).$$

□

It is important to note that the above holds regardless of whether any abstraction satisfies optimal-action-preservation for any states it merges. This means that, again with sufficient experience, arbitrary abstraction candidates will never be favoured over the ground truth.

3.4.4. SAMPLE COMPLEXITY AND MEMORY

QLiA will converge to the solution of the ground MDP under the same conditions as Q-learning. However, this says nothing for whether a solution will be reached with fewer interactions with the environment. In fact, by incorporating a second learning problem into the method, it is possible to have increased the learning requirements. Sample complexity scales linearly with the size of the state-action space [12], therefore the size of the problem space is used as a proxy measure of whether sample complexity has been reduced.

In QLiA, what would be a single learning task is transformed into two parallel learning problems: 1) the meta-agent learns which abstractions optimize each state, 2) each abstraction agent learns which actions optimize each abstract state. The size of the abstraction set $|A_\Phi|$ is the number of abstractions available to the agent and $|\bar{S}_i|$ the size of the abstract state space resulting when abstraction ϕ_i is applied to S. The first problem has a size of $|S \times \mathrm{A}_\phi|$ and the second a size of $\sum_i^{J+1} |\bar{S}_i \times A|$ for each abstraction i. The abstraction agents learn in parallel, thus the one with the largest problem space will dominate the problem size. A reduction in state space comes when abstractions can learn the optimal action for a sufficient number of merged states. The problem of learning the optimal action in abstract space reduces to a size of $|1 \times A|$ for each set of states that were merged into a single abstract state for the corresponding abstraction agent. The total size of the problem is dependent both on the number of states merged by a candidate abstraction that maintain the optimal-policy-preserving predicate as well as

the number of candidate abstractions. Therefore, there are situations wherein the problem size can be increased, if the number of candidate abstractions is too high and the number of merged states with appropriate abstractions too few.

The total size of the QLiA problem is the size of the problem of learning which abstraction agent to select for each state ($|S \times \mathrm{A}_\phi|$) plus the problem of learning the correct action for each state that does not get merged by an abstraction where the optimal-policy-preserving predicate is held ($|(S - \sum_{j=1}^{|A_\Phi|} S_p^j) \times A|$), plus the problem of learning the correct action for each state correctly merged under the predicate in the corresponding abstract space, which can be approximated as the problem size of each abstract MDP. This last term will be dominated by the size of the largest abstract MDP, given by: $|\max_{k \in \{1,\ldots,|\mathrm{A}_\phi|\}} \overline{S}_p^k \times A|$. The problem size is summarized in the following:

$$|S \times \mathrm{A}_\phi| + |A||S - \sum_{j=1}^{|A_\Phi|} S_p^j + \max_{k \in \{1,\ldots,|\mathrm{A}_\phi|\}} \overline{S}_p^k| \tag{3.4}$$

In the case where no states that share an optimal action are merged by any of the abstractions, the second term is dominated by the largest abstraction (i.e. the non-abstraction) and becomes $|S \times A|$, showing a clear increase in the total problem size. A reduction in the problem space occurs when the entirety of Equation 3.4 sums to less than $|S \times A|$. This is only possible when the number of abstractions is strictly fewer than the number of actions available in the MDP. Imagine a task with 10 states and 4 actions, where a single abstraction exists that merges 4 states which share an optimal action under an abstraction ϕ. This abstraction plus the full state brings the total number of candidate abstractions to 2. The problem space is then $(10 \times 2) + 4((10 - 4) + 1) = 48$, greater than $|S \times A| = 10 \times 4 = 40$. However, consider the same problem but instead 8 states share an optimal action and are merged into a single abstract state; the problem space becomes $(10 \times 2) + 4((10 - 8) + 1) = 32$ which means that QLiA will solve a smaller problem and is more likely to fewer samples to reach the optimal solution over a ground Q-learning agent. Whether this is actually the case is problem-specific and will depend on the actual transition and reward dynamics of the environment [1]. The memory and computation requirements of applying the QLiA algorithm (when subtasks are not fully-specified) are higher than that of standard Q-learning. Each abstraction agent must have its own Q-table stored in addition to the meta-agent table. Further, the number of Bellman updates is multiplied by a factor of the number of abstraction candidates.

3.4.5. FULLY-SPECIFIED ABSTRACTIONS

When the state-conditioned state abstractions are fully specified and the meta-agent policy is fixed, there is also less flexibility as the agent cannot recover the optimal policy if any relevant state sets are incorrectly specified. This raises the question of whether the correct specification of the relevant-states affects convergence to the optimal policy.

[1] There is an opportunity for additional use of experience that is not investigated in detail in this thesis, but is enabled by the parallel learning approach and worth noting. The policy of each abstraction agent can be queried from any state s to see which action each agent *would* execute. Then, upon taking an action suggested by the chosen abstraction, additional updates using the same experience can be made to the $\hat{\mathrm{Q}}$-table at state s for all abstractions that *would have* taken the same action.

In the trivial scenario where a single state-conditioned abstraction mapping preserves the optimal-action for all states in S, applying QLiA is equivalent to applying regular Q-learning in the resultant abstract MDP. If two states s_1 and s_2 share an optimal action and are merged under the single abstraction, the value of s_1 can be overestimated if the value of s_2 is higher; the merged state will have a value higher than the true value of s_1. This overestimation can cause convergence to a sub-optimal policy [1]. It is therefore not possible to prove that QLiA will converge to the optimal policy under this particular predicate. Even so, taking this approach can greatly improve learning speed over both ground Q-learning and QLiA without specified relevant states, as was demonstrated empirically in experiments. Further, it may be possible to guarantee converged to optimality when stricter predicates are preserved, though this is outside the focus of the current work.

REDUCTION IN PROBLEM SIZE

The fully-specified abstraction case potentially causes a large reduction of the problem space of a task, particularly because the first term of the problem size (given in Equation 3.4) no longer applies. This results in a size of:

$$|A||S - \sum_{j=1}^{|A_\Phi|} S_p^j + \max_{k \in \{1,\ldots,|A_\phi|\}} \overline{S}_p^k|$$

The computation and memory requirements are also lower than when abstractions are not specified, as there is no longer a need to maintain and train the meta-agent.

3.5. TRANSFER LEARNING

A benefit of taking a modular approach is the potential for transfer learning, as each module could be re-used in related tasks which share abstract sub-tasks. The problem is formalized as follows: consider two related tasks as source and target MDPs M_s and M_t. An abstraction mapping function ϕ_s that maps the source state space to a local abstract state space is provided as a candidate abstraction and QLiA is performed in the source MDP, resulting in the learned Q-table $\bar{Q}_s$. A transformation τ is given that, when applied to the abstraction mapping maps states from the **target** state space to the same local abstract state space. In the target task, this candidate abstraction uses the mapping $\tau(\phi_s)$ and the Q-table $\bar{Q}_s$ is directly used to choose actions without further updating. The meta-agent must learn, for every state, the value of choosing this pre-trained module to choose actions.

An issue with many approaches to transfer learning is that the agent can perform worse than an agent that starts from scratch [4]. By taking the parallel learning approach, this issue is mitigated because the meta-agent learns the value of the transferred module in the target task, without assuming anything about how the transition and rewards differ between the target MDP and the source MDP in which the module was trained. A pre-trained module can be re-used without fear of negative transfer, as the meta-agent will learn to avoid the choices of this module if they are sub-optimal.

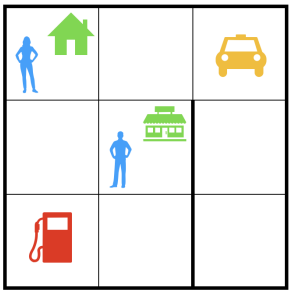

Figure 3.7: The mini taxi with fueling domain, where the taxi must pick up the indicated passenger and drop them off at their specified destination without running out of fuel. The taxi can refuel at the fueling station.

3.5.1. EMPIRICAL EVALUATION

The potential for transfer learning is demonstrated on a modified version of the taxi problem, where the taxi has a limited fuel capacity and faces a -20 reward penalty if it runs out of fuel during an episode (after which the episode ends) [13]. In this domain, the taxi can refuel by visiting the fuel station and executing the `refuel` action, which is appended to the original action space. The state space is expanded to include a state variable that indicates the `fuel level`. In these experiments, the environment acts deterministically and is miniaturized to the 3x3 grid pictured in Figure 3.7.

For the transfer agent, QLiA is first applied to the original problem without the fueling task, meaning the taxi has unlimited fuel and the `fuel level` is not a part of the state space. In this original learning task, the QLiA agent was given a single abstraction module pertaining to the subset of variables `{x, y, passenger location}`. After learning in the task without fueling, the module and its policy were provided to the QLiA_transfer agent. The module uses the same mapping function in the new task, additionally ignoring the fuel level. The transfer agent is compared to Q, QLiA, and QLiA_specified agents that all learned from scratch. The results of learning in the new fueling task are plotted in Figure 3.8.

This experiment demonstrates how QLiA facilitates transfer learning without making assumptions on how the source and target tasks are similar or where the transferred knowledge can be used. The QLiA_transfer agent converges to a near-optimal solution in the fewest number of episodes, even when it has to learn how to use the transferred module using the meta-learning layer. In this environment, the Q agent performs better than the QLiA agent when both are learning from scratch. The QLiA_specified agent outperforms both Q and QLiA.

3.6. RELATED WORK

The proposed method of parallel learning in abstractions is closely related to the concepts of learning state abstractions, hierarchical RL and the more loosely-defined modular reinforcement learning.

Hierarchical approaches focus on identifying sub-goals and trying to reach these in compact ways. The aim is often abstraction via the use of temporally-extended ac-

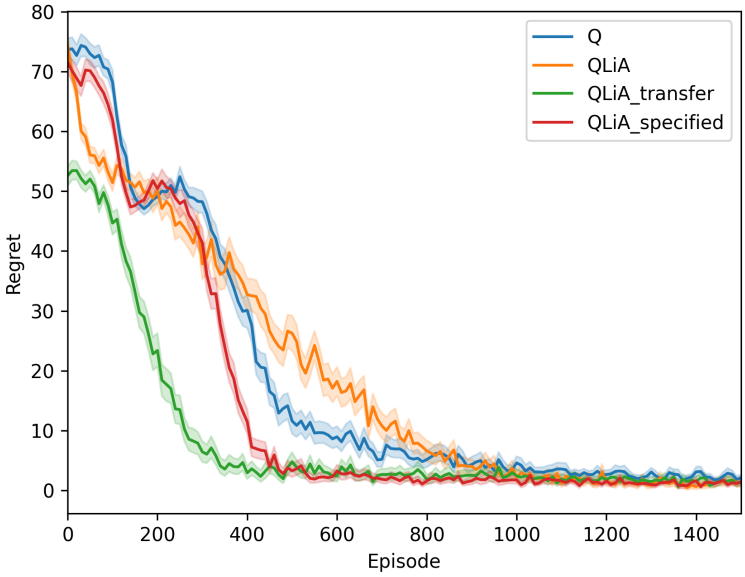

Figure 3.8: Results showing the average of 200 trials of experiments on taxi with fueling domain. The shaded region indicates the 90% confidence interval.

tions, rather than state abstraction. QLiA differs from other hierarchical approaches as it does not employ temporal abstraction, rather a different state abstraction can be activated at every time step. Generally, hierarchical approaches assume perfect knowledge of the structure and the appropriate abstractions. MAXQ, a famous early hierarchical RL method, is one such approach that was specifically adapted to be used in conjuction with state abstraction [14]. In QLiA, the root agent learns to use (or not use) appropriate reward-maximizing abstractions, thus the method is robust to being given poorly-designed sub-spaces. In this way, some structure is proposed, but not assumed, lending more flexibility.

Modular RL has become more of an umbrella term that can emcompass many different types of methods, but the term was popularized by two approaches that were introduced around the same time [15, 16]. Both methods employ a multi-layered approach where an arbitrator (similar to the meta-agent in QLiA) decides which module should choose an action based on its local information. These works are significantly different from the setting described above as the assumption is that each module forms a complete MDP with a local reward function. Even though this still has implications over whether an optimal policy can be learned (generally, no), it has none of the complications introduced in the scenario where each module is given a partial abstraction of a single MDP and only the global reward function can be used.

Existing research on reinforcement learning with state abstractions is often concerned with finding abstractions that preserve qualities of the entire MDP. Generally, these can only be applied when the solution to the MDP is already available [3, 9, 17]. The most relevant literature to the parallel learning in abstractions approach are reinforcement learning algorithms that aim to discover abstractions or that learn how to apply them online. Some have used statistical measures to determine whether an abstraction correctly predicts observations [18] or to find abstractions that contain the minimum state variables relevant to learning the optimal policy [9]. In [19], they learn relevant features of state space at every time step via demonstration from humans. In contrast, the parallel learning approach uses global reward as the feedback signal for learning the quality of a particular abstraction, which is represented in a state-conditioned decomposition, and does this in an online fashion without access to the optimal solution.

The parallel learning in abstractions approach is most closely related to the work of [20] on abstraction selection. In their work, they augment the state space with abstractions and the action space with 'switching' actions. In an episodic MDP version, they demonstrated empirically that their method is promising even when a switching action is taken at every time step. However, in this method candidate abstractions must retain the Markov property in regards to the full problem. In the parallel learning paradigm, several abstractions can be used at once that reduced the state space but, if used on their own, would not allow for the full task to be solved. In arguably more domains, this is likely the case and it requires a more flexible solution.

3.7. LIMITATIONS

While the parallel learning in abstractions paradigm offers a simple approach to incorporating human knowledge that can potentially drastically decrease the size of the problem space, it suffers some drawbacks that are shared by many hierarchical or modular

methods. Because of the layered and parallel learning approach, the number of hyperparameters to be tuned grows with each added module. This can make tuning much more difficult, and burdens the designer with finding good values. It was seen in experiments that the choice of learning rate should not be too high even in deterministic problems where this is not an issue for Q-learning. This approach however, has not been found to be more sensitive to the choice of other hyperparameters than standard Q-learning, and performs robustly.

It is not guaranteed that applying the parallel learning approach will reduce the problem space, which means it will not always improve sample efficiency over a naïve Q-learning agent. It is noted in particular that, if the primitive action space is smaller than the number of candidate abstractions, the problem space has actually been increased (potentially significantly). However, the approach can still facilitate other benefits such as transfer learning and transparency, even if sample efficiency in the source task is not achieved, as was demonstrated in the taxi with fueling domain.

Another limitation mentioned is that in the method when state-conditioned abstractions are fully specified, it is not possible to guarantee convergence to the optimal solution, though empirically it has been seen that great improvements can be made in this case.

3.8. CONCLUSION

Parallel learning in abstractions exploits the existence of optimal-policy-preserving abstractions with minimal knowledge of the problem beforehand. It does this in an online fashion using only global reward signals distinguishing the approach from most modular and hierarchical approaches. By applying state-conditioned state abstraction, much simpler human-definable abstraction mappings can be used which do not necessarily hold for the entire state-space.

This chapter introduces an approach to making use of state abstractions that do not encompass the entire MDP, reminiscent of the goals of hierarchical reinforcement learning. The devised method allows for the use of candidate abstractions provided by an expert that can potentially speed up learning, but where the optimal policy can still be recovered if these abstractions are not sound or otherwise useful. This parallel learning approach allows experience to be used efficiently, updating the local models of each abstraction candidate, which still takes into account the entire layered system that it aims to optimize. By taking a modular approach, the policies learned by the meta-agent can be reviewed to gain insight into which state abstractions are being used at any state. This method also enables transfer learning when a related problem shares an abstract state space with the source environment, but is otherwise different. The notion of state-conditioned state abstraction was also introduced, which is argued to be a much easier approach to encorporating information from an expert over abstractions that preserve a predicate over the entire MDP.

While QLiA was demonstrated to improve sample efficiency, the layered learning approach can potentially insert noise into the layers that take large amounts of experience to unlearn. The additional computation and memory required can also result in a significant load when state-action spaces grow exponentially. Nonetheless, it offers a method that provides insight into which parts of the state space are used to make decisions and

can make use not only of partial abstractions (abstractions that do not apply to the entire MDP), but of the coarsest form of abstraction which is easiest for humans to define, without requiring extensive knowledge about how or where to use it.

REFERENCES

[1] L. Li, T. J. Walsh, and M. L. Littman, *Towards a Unified Theory of State Abstraction for MDPs,* in *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics* (2006) pp. 531–539.

[2] W. A. Johnston and V. J. Dark, *Selective Attention,* Annual Review of Psychology **37**, 43 (1986).

[3] D. Abel, D. Arumugam, L. Lehnert, and M. Littman, *State Abstractions for Lifelong Reinforcement Learning,* in *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, edited by J. Dy and A. Krause (PMLR, 2018) pp. 10–19.

[4] M. E. Taylor and P. Stone, *Transfer Learning for Reinforcement Learning Domains: A Survey,* Journal of Machine Learning Research **10**, 1633 (2009).

[5] E. Even-Dar and Y. Mansour, *Approximate Equivalence of Markov Decision Processes,* in *Learning Theory and Kernel Machines*, edited by B. Schölkopf and M. K. Warmuth (Springer Berlin Heidelberg, Berlin, Heidelberg, 2003) pp. 581–594.

[6] S. P. Singh, T. S. Jaakkola, and M. I. Jordan, *Learning without State-Estimation in Partially Observable Markovian Decision Processes,* in *Proceedings of the 11th International Conference on Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994) p. 284–292.

[7] R. A. N. Starre, M. Loog, and F. A. Oliehoek, *An Analysis of Abstracted Model-Based Reinforcement Learning,* arXiv (2022), arXiv:2208.14407.

[8] T. G. Dietterich, *Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition,* Journal of Artificial Intelligence Research **13**, 227–303 (2000).

[9] N. K. Jong and P. Stone, *State Abstraction Discovery from Irrelevant State Variables,* in *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005) p. 752–757.

[10] C. J. C. H. Watkins and P. Dayan, *Q-Learning,* Machine Learning **8**, 279 (1992).

[11] R. A. N. Starre, M. Loog, and F. A. Oliehoek, *Model-Based Reinforcement Learning with State Abstraction: A Survey,* in *Joint International Scientific Conferences on AI and Machine Learning, BeNeLearn* (2022).

[12] S. M. Kakade, *On the Sample Complexity of Reinforcement Learning,* Ph.D. thesis, University College London - Gatsby Computational Neuroscience Unit (2003).

[13] T. G. Dietterich, *The MAXQ Method for Hierarchical Reinforcement Learning,* in *Proceedings of the 15th International Conference on Machine Learning* (1998) pp. 118–126.

[14] T. Dietterich, *State Abstraction in MAXQ Hierarchical Reinforcement Learning,* in *Advances in Neural Information Processing Systems*, Vol. 12, edited by S. Solla, T. Leen, and K. Müller (MIT Press, 1999).

[15] S. Russell and A. L. Zimdars, *Q-Decomposition for Reinforcement Learning Agents,* in *Proceedings of the 20th International Conference on Machine Learning* (AAAI Press, 2003) p. 656–663.

[16] N. Sprague and D. Ballard, *Multiple-Goal Reinforcement Learning with Modular Sarsa(O),* in *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003) p. 1445–1447.

[17] D. Andre and S. J. Russell, *State Abstraction for Programmable Reinforcement Learning Agents,* in *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (American Association for Artificial Intelligence, USA, 2002) p. 119–125.

[18] G. Konidaris and A. Barto, *Efficient Skill Learning Using Abstraction Selection,* in *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009) p. 1107–1112.

[19] L. C. Cobo, K. Subramanian, C. L. Isbell, A. D. Lanterman, and A. L. Thomaz, *Abstraction from Demonstration for Efficient Reinforcement Learning in High-Dimensional Domains,* Artificial Intelligence **216**, 103 (2014).

[20] H. van Seijen, S. Whiteson, and L. Kester, *Efficient Abstraction Selection in Reinforcement Learning,* Computational Intelligence **30**, 657 (2014).

4

UNCERTAINTY-AWARE PREDICTIVE REINFORCEMENT LEARNING

எதிரதாக் காக்கு மறிவினார்க் கில்லை
அதிர வருவதோர் நோய்.
- திருக்குறள் 429

The wise who are perceptive and take preventive measures,
will not be staggered by any harm.
- Tirukkuṟaḷ 429

In the previous chapter, several benefits to developing modular approaches to reinforcement learning were touted. Among these was the potential to create RL agents that are flexible and can adapt to new scenarios. Humans do not learn from scratch every time they find themselves in a new context, instead they can use knowledge from past experiences and learn to adapt to new information. These adaptations are considered this chapter, in which the goal is to equip reinforcement learning agents with the ability to transfer learning between state-spaces in order to efficiently adapt to the addition of new information.

Humans have the ability to expand and compress their local models seamlessly, adding or removing information input as needed [2]. For reinforcement learning agents, facing a problem where not all the information is available at every time step puts the agent in a partially-observable MDP. Even if additional information is there during training, this cannot be used if it is not available during policy deployment. In a partially-observable MDP, parts of the state space cannot be observed directly, and finding a solution becomes significantly more complicated. Many idealized problems are partially-observable when realized in the real-world, making the ability to learn from information

Elements of this chapter are published in [1].

that is not present at decision-making time something that can be beneficial in many ways.

In this chapter, a modular approach is introduced which geared toward partially-observable problems that fit into a special case; they are fully-observable during training but not during deployment. Two distinct possibilities for encountering this scenario are provided as examples: 1) There is extra information available during training such as *hindsight* knowledge (for example, in fraud mitigation it could be possible to label fraud at some point in the future but this label will not be available when the policy needs to choose actions) or 2) Information can be deliberately limited or removed during training (for example, lab conditions may be noiseless but sensor readings can be purposefully perturbed to simulate the real-world). This allows the reinforcement learning agent to benefit from supervised learning advances, as labeled data is available, reducing some of the burden on policy optimization to implicitly predict informative states.

The method Uncertainty-Aware PREdicted STate ReinfOrcement Learning (PRESTO) explicitly decouples the tasks of state prediction and policy optimization, producing a modular method that facilitates transfer between related problems. PRESTO trains a predictive model on labeled data to predict the true state from a history of observations and actions. During training, this label is available, but during testing the agent sees only its observations and actions. The policy is learned over both the predicted state and a measure of confidence provided by the model. This and the details of the problem setting differentiate the proposed approach from a Predictive State Representation [3], which has otherwise similar goals. As the agent gains more data and more confidence in its predictions, it learns to take information-gathering actions when doing so will improve its value-maximizing policy.

The approach considered here applies to partially-observable problems that can be cast as History MDPs (closely related to k-order Markov models [3]). Many types of partially-observable problems fall under this class, and in literature it is common to use histories to create a Markov state without formalizing the approach. When deep Q-learning had its breakthrough debut solving Atari games [4], the practice of stacking observations was popularized and has continued since. This is how many existing methods absorb the task of predicting the true state into the policy optimization task, stacking observations into histories or attempting to model latent parameters. These methods do not have a mechanism to learn to directly map histories to partially-observable state information, thus the belief of the agent is implicit. By separating out the prediction task, the proposed approach preserves this information and makes it available along with the policy. In contrast to using Long Short-Term Memory neural networks (another common way to handle such problems), this offers explicit insight into the decision making of an agent and simplifies an otherwise costly and difficult training process. The modularity of the approach also enables transfer of both the predictor or the policy between related tasks, along with a formalization of the required relationship between tasks to successfully do so. This makes the method robust to changes in the environment and able to make use of existing prediction models or policies.

This chapter introduces PRESTO and describes its several benefits: it (1) conducts policy optimization over a compact prediction, (2) facilitates transfer of either the policy or predictor between related tasks, and (3) makes explicit the agent's prediction of the

hidden state at every time step. These benefits are demonstrated in tabular and continuous state space experiments, showing sample-efficient convergence to an optimal policy and effective transfer learning.

4.1. HISTORY MARKOV DECISION PROCESSES

A *partially-observable MDP* (POMDP) generalizes the MDP described above to problems where the full state cannot be observed [5]. A POMDP is a 7-tuple $\langle S, A, T, R, \Omega, O, \gamma \rangle$, where: S, A, and Ω are the sets of states, actions and observations, T and R are the transition and reward functions as described above, O is the observation function which maps state-actions to observations $O\colon\ S \times A \times O \ \rightarrow\ [0,1]$, and γ is the discount factor. In a POMDP, the environment transitions according to the hidden true state and emits an observation according to the observation function $o_t \sim O(\cdot \mid s_t, a_t)$. When the state space S and observation space Ω are equivalent, this is equivalent to the definition of an MDP.

A history h is a k-length sequence of observations and actions, $h\colon (\Omega \times A)^k \times \Omega$. The set of histories is denoted H. The belief state of a POMDP b is the probability distribution over states given history h, $\mathrm{b}(s,h) = Pr(s_t = s | h_t = h)$.

A POMDP can be cast to an MDP by observing a history of experience at every time step, rather than a single observation. The result is a *History MDP*, formed by a tuple $\overline{\mathbb{H}} = \langle H, A, \overline{T}, \overline{R}, \gamma \rangle$, where

$$\overline{T}(h_t a_t o_t | h_t, a_t) = \sum_{s_t \in S} \sum_{s_{t+1} \in S} \mathrm{b}(s_t, h_t)\, T(s_{t+1} | s_t, a_t)\, O(o_t | s_{t+1}, a_t)$$

and

$$\overline{R}(h_t, a_t) = \sum_{s_t \in S} \mathrm{b}(s_t, h_t)\, R(h_t, a_t)$$

[6]. This work focuses on a common subclass of POMDPs where history length k is finite.

4.2. UNCERTAINTY-AWARE PREDICTED STATE REINFORCEMENT LEARNING (PRESTO)

This section introduces a method that decouples state prediction from policy optimization, called Uncertainty-Aware PREdicted STate ReinfOrcement Learning (PRESTO). This method is aimed at problems where the state is partially-observable by the agent during testing, but fully-observable during training. One reason for this could be that information is limited during training on purpose. For example, in a real-world problem, taking high accuracy measurements may require an expensive and fragile sensor, whereas it is preferred to equip an RL robot with a cheaper, more robust sensor. With only the cheap sensor, the problem is partially-observable because a single reading does not offer a Markov state; with the highly accurate sensor the problem is fully-observable. The expensive sensor can be used in tandem with the cheap sensor in a lab environment, providing the real values along with those expected in deployment. Once deployed, the robot no longer has access to the high accuracy readings and must use only its equipped sensor.

The second type of problem pointed to in this chapter, which is fully-observable during training but not at action selection, is one where information is revealed in hindsight. This is motivated with the example task of fraud mitigation. Consider the problem of deciding which bank accounts to suspend based on corresponding lists of recent transactions, where the objective is to minimize disruption to customers while preventing fraud. It is not possible to see which accounts are committing fraud when choosing which to suspend, however, after an investigation is completed it is revealed whether a particular account was indeed committing fraud or not. This type of revealed information is referred to as *hindsight* knowledge.

This approach is formalized as a transfer learning task where the source task (fully-observable) and target task (partially-observable) have different observation spaces.

4.2.1. LEARNING FROM HISTORY

The problem of learning a policy for a partially-observable MDP in a fully-observable MDP is modelled as transfer learning task between two POMDPs. Consider the target task as a POMDP $M_T = \langle S, A, T, R, \Omega, O, \gamma \rangle$. The source task is defined as another POMDP $M_S = \langle S, A, T, R, \overline{\Omega}, \overline{O}, \gamma \rangle$, which shares the same state and action space, but where $\overline{\Omega} = S \times \Omega$, $\overline{O}$: $S \times A \times \overline{\Omega}$, and

$$\overline{O}(\langle s', o \rangle | s, a) = \begin{cases} O(o|s,a) & s = s', \\ 0 & \text{otherwise.} \end{cases}$$

This augmentation formalizes the receipt of an observation *and* the true state in the source POMDP at every time step, whereas in the target POMDP, the agent receives only the observation.

An important assumption is made here that the source and transfer POMDPs can be cast to History MDPs $\mathbb{H}_S$ and $\mathbb{H}_T$ (respectively) by maintaining the same k-length history of observations and actions, where k is an expert-determined parameter. This is the approach taken when observations are stacked, such as is common when solving Atari games where k is often an empirically-validated hyperparameter. Note that only the observations (thus not the state available in the source task) are stored in the history, making $\mathbb{H}_S$ and $\mathbb{H}_T$ equivalent.

The general approach to learning in a History MDP would be to treat it as an MDP and learn a policy over the history space. In PRESTO, the agent's prediction of the current state (given the history) is instead explicitly modeled. The agent's predictive model is trained at the same time as learning a policy that maximizes rewards. The policy is learned over the predicted state and confidence in the prediction. This makes explicit the agent's understanding over which state it is in (an insight which is lost otherwise), while training a policy that takes the model uncertainty into account, learning to favour information-gathering actions when necessary for reward maximization. Separating out the prediction in this way forms a stationary task suitable for supervised learning algorithms. Figure 4.1 depicts these two approaches, one which learns directly from history and the other which uses the history to predict a state, learning on the joint space of state prediction and confidence.

The value function (V^π) according to following policy π is defined over the joint space of predicted states and confidence values, mapping pairs of predicted state and confi-

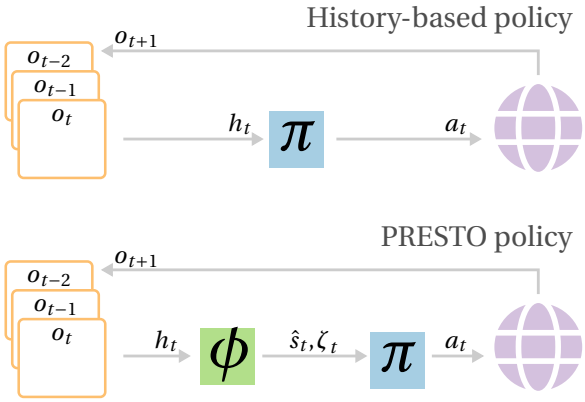

Figure 4.1: An overview of the proposed method in comparison to learning from a history. The history-based policy maps stacks of observations to actions. The PRESTO policy takes a predicted state and model confidence as input and maps this to actions.

dence to actions, $\pi(\hat{s}_t, \zeta_t) = a_t$. The predicted state and confidence correspond closely to a belief state in the solution to a POMDP, however depending on how they are defined, they could be less informative than a belief; for example, by only showing the probability of being in the most likely state and not how this probability is distributed over the entire state space.

4.2.2. THE PREDICTIVE MODEL

A PRESTO agent has a predictive model $\phi(h)$, that maps histories to both the predicted true state ($\hat{s}$) and a confidence (ζ) in this prediction (a value greater than or equal to 0) $\phi\colon H \mapsto (S \times \mathbb{R}_{\geq 0})$. During training, the model is updated at each time step t:

$$\phi(h_t) \leftarrow s_t.$$

The model output given history h_t is:

$$\phi(h_t) = (\hat{s}_t, \zeta_t).$$

The choice of model is dependent on the specifics of the problem, including, for example, whether the history space and the predicted state space are discrete, continous, low- or high-dimensional, and whether the prediction task can be presented by a linear or non-linear model. Particularly for high-dimensional and complex state spaces or highly non-linear prediction tasks, the progress made in supervised machine learning offers the opportunity to tackle difficult prediction tasks. A measure of confidence ζ is also required with each prediction. As with the choice of model, the way to represent model confidence is something that must be chosen by a designer as suitable for the problem. In a Bayes-Adaptive POMDP, a measure of uncertainty is represented by visitation counts [7]. While this can be effective, it requires an extra parameter to scale the number of counts into a meaningful number. Other options, such as extracting class

probabilities or predicting with an ensemble of models and calculating their deviation, were preferred in this work and are detailed below.

The specifics of model selection are considered out of scope of this work, but two possible methods are highlighted which were applied in experiments.

BINARY CLASSIFIER

In the case where the true state can only take on a small number of discrete values, a classifier may be used. In the tiger problem, which is detailed in the experiment section, this is the case. With only two possible true states, a binary classifier was used as the state predictor. The underlying model is a multilayer perceptron, where the activation of the output layer g is the sigmoid function which gives a value between 0 and 1.

For a multiclass classifier, this can be the softmax function which normalizes the outputs to sum to 1, giving the probability for each possible class (corresponding to each possible hidden state). The parameters of the model (the weight of each unit) are trained on labeled data using backpropagation. This labeled data consists of pairs of histories and the true hidden states that the agent was in.

The output of the model is the probability that an input (history of observations) belongs to a certain class (hidden state). In PRESTO, this probability ρ predicts the state (whichever class has the highest predicted probability) and itself forms the confidence measure.

REGRESSION ENSEMBLES

In the case where the predicted state lies in a continous space, a regression model is more suitable. In experiments, again a multilayer perception as described above was used, but where the output layer uses the identity function as its activation function, meaning the output is simply the linear combination of the last layer.

In the case of such a regression model, deriving some measure of uncertainty is less obvious. There has been some success in using an ensemble of models to provide a practical heuristic for uncertainty [8, 9]. This is the approach utilized in the continuous state-space experiments, where the variance between the ensemble members is used as a measure of confidence (less variance = more confidence). The ensemble $\mathscr{E}_q$ thus consists of q models:

$$\mathscr{E}^q = \{\phi_1, ..., \phi_q\},$$

where it is assumed that the models are identical before initialization. Each model is initialized independently and trained on unique batches of the dataset. This causes the models to deviate, the idea being that the deviation will be greater in areas where the dataset coverage is sparse or highly varied (therefore deemed low in confidence).

The predicted state $\hat{s}$ given a history h is the average of the ensemble outputs:

$$\hat{s} = \frac{1}{q} \sum_{i=0}^{q} (\phi_i(h)),$$

and the confidence then is the inverse of the standard deviation of each ensemble mem-

Algorithm 3: PRESTO: Training in $\mathbb{H}_S$ with Q-Learning

Input: Length of history k
Input: A value to represent empty history elements ι
Input: Number of training steps N
Input: Learning rates for Q and ϕ as α and β
Input: Batch of episode trajectories $\mathscr{D} = (d_0, d_1, ...)$, where $d_i = (o_0, s_0, a_0, r_0, o_1, s_1, ...)$

1 Initialize policy π
2 Initialize predictor ϕ
3 Set $t = 0$
4 **while** $t < N$ **do**
5 $\hat{d} \sim \text{Uniform}(\mathscr{D})$
6 $h = \{\iota_0, ..., \iota_k\}$
7 i = 0
8 $h \leftarrow PushToQueue(h, o_i)$
9 **while** $i < length(\hat{d})$ **do**
10 $(\hat{s}_i, \zeta_i) = \phi(h)$
11 $\phi \leftarrow Train(h, s_i)$; ▷ `Training can be done in batches instead of every time step`
12 $h \leftarrow PushToQueue(h, a_i)$
13 $h \leftarrow PushToQueue(h, o_{i+1})$
14 $(\hat{s}_{i+1}, \zeta_{i+1}) = \phi(h)$
15 $Q(\hat{s}_i, \zeta_i, a_i) \leftarrow (1-\alpha) Q(\hat{s}_i, \zeta_i, a_i) + \alpha(r_i + \gamma \max_a Q(\hat{s}_{i+1}, \zeta_{i+1}, a))$
16 $i \leftarrow i + 1$
17 **end**
18 $t \leftarrow t + 1$
19 **end**
20 $\pi(\phi(h)) = \underset{a}{\text{argmax}}\, Q(\phi(h), a), \forall\, h \in H$

Result: State predictor ϕ and policy π

ber's prediction

$$\zeta = \frac{1}{\sqrt{\frac{\sum_{i=0}^{q} (\phi_i(h) - \hat{s})^2}{q}}}.$$

4.2.3. ALGORITHM

The method is summarized in Algorithms 3 and 4 where the base RL algorithm is Q-learning [10]. In these examples, it is shown how the raw output of the environment (observations, actions and – during training – states) is used to build the history which the policy is trained on.

In Algorithm 3, the initialization of the history as an empty queue (Line 6) is necessary at the start of each episode so that the agent learns to take into account that it has

Algorithm 4: PRESTO: Testing in $\mathbb{H}_T$

Input: PRESTO policy π
Input: Predictor ϕ
Input: A value to represent empty history elements ι
Input: Number of testing steps N

1 $t = 0$
2 $h = \{\iota_0, ..., \iota_k\}$
3 **while** $t < N$ **do**
4 Observe o_t
5 $h \leftarrow PushToQueue(h, o_t)$
6 $(\hat{s}_t, \zeta_t) = \phi(h)$
7 $a_t = \pi(\hat{s}_t, \zeta_t)$
8 Execute a_t
9 Receive r_t
10 $h \leftarrow PushToQueue(h, a_t)$
11 $t \leftarrow t + 1$
12 **end**

no (or little) information. In the experiments, the indication of an empty history element ι is set to 0, but the best choice may differ according to the context.

The Q-value function of the PRESTO agent is defined over the predicted state $\hat{s}_t$, confidence ζ_t and action a_t, and is updated iteratively (given experience (h_t, a_t, r_t, h_{t+1})) with:

$$Q(\hat{s}_t, \zeta_t, a_t) \leftarrow Q(\hat{s}_t, \zeta_t, a_t) + \alpha(r_t + \gamma \max_a Q(\hat{s}_{t+1}, \zeta_{t+1}, a) - Q(\hat{s}_t, \zeta_t, a_t)),$$

where $(\hat{s}_{t+1}, \zeta_{t+1}) = \phi(h_{t+1})$. This Q update (Line 15) could be replaced with the policy optimization step of other RL methods.

During training (Algorithm 4), the agent learns on trajectories containing both observations and true states, allowing the model ϕ to be trained to predict the state from history along with training the Q-function. If the task is not episodic in nature, the batch $\mathcal{D}$ will be a single trajectory rather than a set of episodes. In the online setting, $\mathcal{D}$ comes from the experience during training, and is iteratively fed back into the algorithm. In the offline setting, this set could come from another behaviour policy. While PRESTO could have potential as an offline method, to what extent it falls prey to the common pitfalls of offline RL is left to future work.

During testing (Algorithm 4), the agent sees only observations. It uses the model to predict the state and probes the learned policy to provide actions given its experience of observations and executed actions.

SIMPLIFYING THE PREDICTION TASK

The proposed method relies on the predictive model being able to predict states from histories. This task is enabled by the existence of labeled data which is independent of the policy, but it is recognized that predicting high-dimensional states may require very

powerful models or large data-sets. One way to reduce this potentially complex task of predicting states is to predict only the un-observable state information. A partially-observable problem can be modelled as a decoupled POMDP [11]. This is identical in concept to the mixed observability MDP [12], but the notation of the decoupled POMDP is adopted here. In this formulation, the state space is factored into observable and un-observable parts. A decoupled POMDP is represented by a tuple $\langle U, Z, \Omega, A, T, R, O, \gamma \rangle$, where U and Z consist of an un-observed and observed finite state space, respectively. If the mapping between histories and only the un-observable part of the state space is relatively easy to model, applying PRESTO will no longer be complicated by the observable state space. In such a case, the joint state the policy is defined on is formed by the observation, predicted un-observed state, and confidence.

4.2.4. MITIGATING UNCERTAINTY

The prediction confidence can be improved in two ways: improve the prediction model or provide a more informative history. As the agent interacts with the environment, it collects more data which is used to train the model. In theory, with this continuous training the model will improve until it can no longer be improved with additional (or more diverse) data, forming a perfect predictor.

Definition 4 (Perfect hidden parameter predictor)**.** A hidden parameter prediction model ϕ is considered a perfect hidden parameter predictor ϕ^* for a History MDP when, for all states $h \in H$, the model returns a predicted state and confidence $\phi^*(h) = (\hat{s}, \zeta)$ and:

$$\hat{s} = \max_s Pr(s|h), \zeta = Pr(\hat{s}|h).$$

This means that the model returns the most likely state given history h and that the confidence corresponds to the probability of being in the predicted state given the same history. This probability is equivalent to the belief $\mathrm{b}(\hat{s}, h)$.

There is some amount of uncertainty that is irreducible given a particular history. For example, without any observations, a predictor will be (at best) as uncertain in its prediction as dictated by the initial state distribution of the environment. After exhaustive data collection and model training produces the ideal model, the remaining confidence forms a measure of how certain a prediction can be made given the input. This confidence can only be improved by taking informative actions (changing the history). By conditioning the policy on the model confidence, the agent learns to take information-gathering actions when doing so will improve its policy.

4.2.5. TRANSFERRING THE PREDICTOR AND THE POLICY

A crucial benefit and perhaps best motivation to taking a modular approach is that separable pieces are maintained that can be reused for new tasks, avoiding the need to learn again from scratch. This method facilitates transfer of both the predictor and the policy. Consider the case where the source task (casted to a History MDP) $\mathbb{H}$ and target task (casted to a History MDP) $\widetilde{\mathbb{H}}$ share the same k-history and action space. Under the following sufficient conditions, the predictor ϕ and the policy learned when optimizing $\mathbb{H}$ can be re-used when optimizing $\widetilde{\mathbb{H}}$.

PREDICTOR TRANSFER

The problem of predicting the true state from a history h is preserved between two tasks $\mathbb{H}$ and $\widetilde{\mathbb{H}}$ if, given the same k-length history of observations and actions, a perfect hidden parameter predictor ϕ^* would yield the same prediction and confidence:

$$(\hat{s}_t, \zeta_t) = \phi^*(h_t) = \phi^*(\widetilde{h}_t) = (\widetilde{\hat{s}}, \widetilde{\zeta}_t)$$

In this case, assuming the model ϕ trained in task $\mathbb{H}$ has converged to ϕ^*, it can immediately be injected as $\widetilde{\phi}$ when training on task $\widetilde{\mathbb{H}}$, requiring only the policy $\widetilde{\pi}$ to be trained.

POLICY TRANSFER

The policy can be transferred between tasks if the same policy optimizes both tasks. Two tasks $\mathbb{H}$ and $\widetilde{\mathbb{H}}$ with value functions V and $\widetilde{V}$ share an optimal policy $\pi^*(\cdot \mid \langle \hat{s}, \zeta \rangle)$, if:

$$V(\pi^*) \geq V(\pi') \; \forall \pi', \text{ and}$$

$$\widetilde{V}(\pi^*) \geq \widetilde{V}(\widetilde{\pi}') \; \forall \widetilde{\pi}'.$$

Under these conditions, the policy π^* learned from task $\mathbb{H}$ can be directly applied to task $\widetilde{\mathbb{H}}$ requiring only the predictor to be retrained during policy optimization of task $\widetilde{\mathbb{H}}$.

4.3. EMPIRICAL DEMONSTRATION

PRESTO was evaluated in two problems, a continuous control task and a discrete task. In these experiments, it is shown how explicitly predicting the true state allows for efficient use of data and facilitates reliable transfer of both the policy and the predictor.

4.3.1. SIM2REAL - CARTPOLE

The following experiments are performed on a modified version of the cartpole problem from OpenAI Gym [13]. The cartpole problem is a continuous control task, in which the RL agent controls a platform which has a pole balanced on top of it. The agent can move the platform either left or right (applying a fixed force) and aims to keep the pole balanced as long as possible. The agent observes the cart position and velocity and the pole angle and angular velocity. There is a reward of +1 for every step where the pole remains upright, and the episode ends if the pole falls down. In the modified cartpole environment, the sensor which provides the angle and angular velocity of the pole has a lower sampling frequency and only provides a reading at certain intervals. Using PRESTO means that intermittent data from the sensors with a higher sampling frequency does not need to be thrown out.

All test agents are running Proximal Policy Optimization (PPO) as provided by the `stable-baselines3` library [14]. The figures plot the episode reward according to the number of training steps, during which both the predictor and policy are training. The reward is averaged over 25 test episodes. A maximum episode length of 200 steps is imposed on the environment.

The experiments show the performance of four RL agents. The Oracle agent can see the sensor readings at every time step, regardless of any sensor limitations. This provides

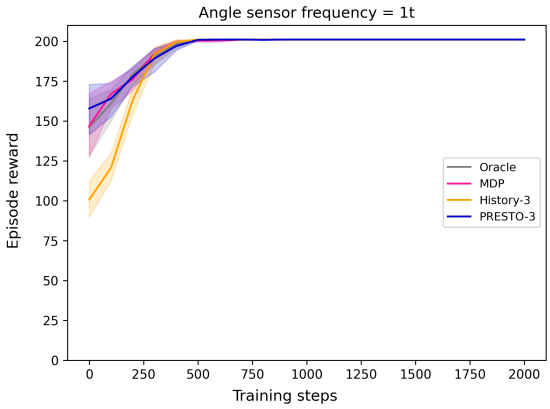

Figure 4.2: Average of 25 trials on the modified cartpole problem when the angle sensor sample frequency is the same as for the position sensor (every time step). Shaded region depicts the 96% confidence interval between trials.

an idea of the expected performance of the agent in the laboratory/simulator setting. The MDP agent treats the environment as though it is an MDP, even if it not. The History agent learns a policy over the problem cast to a History MDP, formed by observing a history of length 3. The PRESTO agent treats the problem as a decoupled POMDP and trains a predictive model to predict the unknown angle and angular velocity (and confidence) from the same history of observations as given to the History agent. The PRESTO agent combines the prediction with the observation, forming the predicted state. An ensemble of 3 linear regression models are used as the prediction model, with the ensemble mean providing the predicted angle and angular velocity and the inverse standard deviation between ensemble members acting as a measure of confidence. The policy is learned over the joint space of the predicted state and this model confidence. The PRESTO agent trains its predictor on the labelled data-set every 25 steps, training the policy every step.

Figure 4.2 shows the performance of each agent when the sensor has a reading every time step (i.e., highest possible sampling frequency). The MDP and Oracle agents are identical in this case, as the problem is fully observable. The PRESTO agent, which performs similarly, has to train its predictor and learn a policy over the joint space of the prediction and the confidence. However it is also given the true state in the form of labelled data, something which the History agent cannot make use of.

The same agents were run on the same task but with a reading of angle and angular velocity provided every other time step (the sample frequency is reduced by a half). The results are shown in Figure 4.3. In this experiment, the difference between the agents is clear. The MDP agent treats the environment as fully observable, which it is not, and does not converge. The History agent has a larger state space over which its policy must be optimized over but does perform well. The PRESTO agent learns an optimal policy slightly faster than the History agent, which is unsurprising as the History agent does not receive feedback about the true state. However, one of the main benefits of PRESTO is providing a mechanism to learn from such feedback when it is available, as well as the

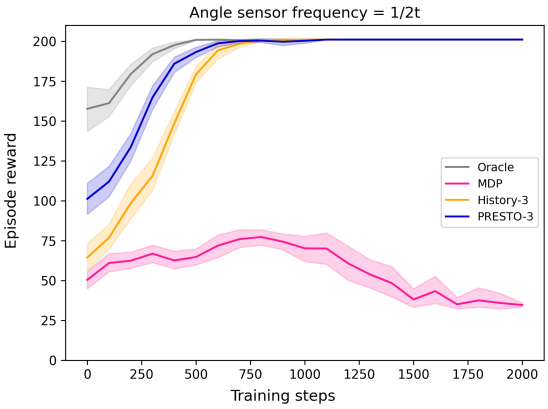

Figure 4.3: Average of 25 trials on the modified cartpole problem where the sensor sample frequency is $\frac{1}{2}$/timestep, i.e., it gets a reading every other time step. Shaded region depicts the 96% confidence interval between trials.

opportunities for transfer which is demonstrated further.

PRESTO offers robustness to sensor degradation (through policy transfer) In a real-world example, a sensor may degrade or break after some time. Generally, this would require replacement of the sensor or for an RL agent to be trained from scratch with the degraded sensor. If the degradation effects on the sensor are known, this is still a non-stationary problem, requiring time to be included as a state component in order to maintain the Markov property. This is where another benefit of PRESTO comes into play. If the sensor degrades, the supervised learner can be retrained without retraining the policy. This transferability is demonstrated in experiments where the sensor sample frequency in the source task is $\frac{1}{2}$/timestep and in the target task is $\frac{1}{3}$/timestep (now once every 3 timesteps). The results are pictured in Figure 4.4.

In these experiments, the PRESTO-PolicyTransfer agent reuses the policy learned by the PRESTO agent in the source task, and retrains only its predictor in the target task. As a comparison, the PRESTO-NoTrainTransfer agent uses both the policy *and* predictor from the source task.

The History agent, which is trained from scratch in the target task, performs relatively well, but is still outperformed by the PRESTO agent without transfer. The benefits of learning on the compact predicted state space are more clear as the implicit mapping of histories to relevant information becomes more difficult for the History agent.

The PRESTO-PolicyTransfer agent shows a very successful result from transferring the source policy. In this case, the policy is robust to uncertainty in the predictor, showing good performance even when the predictor has not seen much data in the target environment, and improving quickly to an optimal policy. The benefits of *knowing what it does not know* are quite clear here.

The PRESTO-NoTrainTransfer agent shows how poorly the policy would perform if the source predictor was used in the target environment. The reason for this is believed

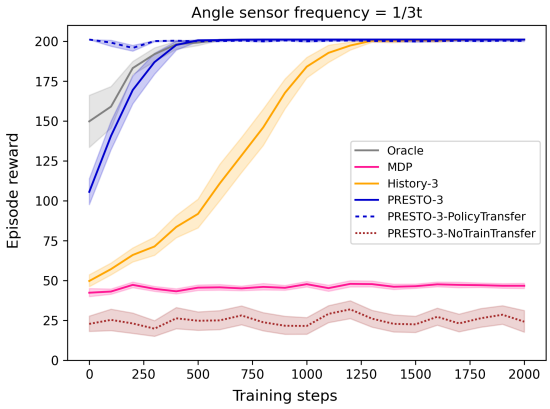

Figure 4.4: Average of 25 trials on the modified cartpole problem where the sensor sample frequency is $\frac{1}{3}$/timestep. The policy is transferred from the task problem but where the sample frequency is $\frac{1}{2}$/timestep. Shaded region depicts the 96% confidence interval between trials.

to be because the deviation between the predictor ensemble members is low (thus, confidence is high), even though the predictions are incorrect, leading to the predictor being undeservedly confident about its output. In this case, the agent *does not know what it does not know*.

PRESTO enables transfer learning (via predictor transfer) If a sensor is used on another agent, or for a different task, the predictor can be reused. This is demonstrated by transfer from the original problem to another version of the cartpole problem where the agent must keep the pole balanced while keeping the cart within 1 unit from the middle position; the results from this experiment are pictured in Figure 4.5.

Again, the PRESTO-NoTrainTransfer agent depicts the results if both the predictor and policy from the old environment are reused. The PRESTO-PredictorTransfer reuses just the predictor, but retrains its policy.

The PRESTO-PredictorTransfer performs similarly to the PRESTO agent without transfer in this particular example; most importantly, it still finds an optimal policy and is not subject to the negative transfer experienced by the PRESTO-NoTrainTransfer agent. This is an important case to show in transfer learning, as it is often impossible to guarantee no negative transfer, where the transfer agent cannot converge to a good (ideally optimal) solution. By re-training only the sub-task that changed (the policy), performance is guaranteed.

4.3.2. HINDSIGHT - TIGER

The cartpole problem is not an ideal environment to demonstrate the need for including information-gathering actions in the policy. This is because the actions for gathering information (trying to keep the pole up so that more sensor readings can be obtained) are the same actions required for reward maximization. To better show how PRESTO pro-

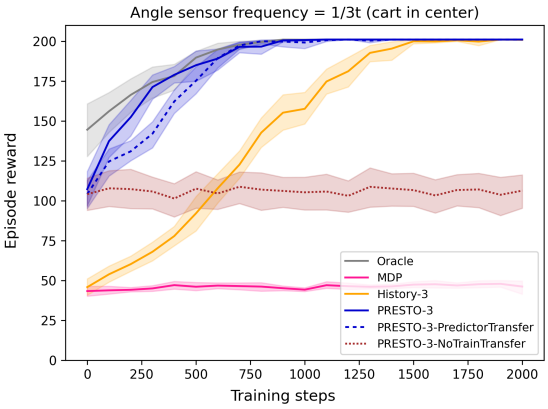

Figure 4.5: Average of 25 trials on a different version of the modified cartpole, showing transfer of the predictor from the original problem. In this experiment, the task has changed and the cart must be kept in the center of the field. The shaded region depicts the 96% confidence interval between trials.

duces a policy that effectively trades off gathering information with gathering rewards, this is demonstrated in the tiger problem.

The tiger problem is a classic POMDP [5], where an agent is faced with two doors and must decide which of two doors to open. Behind one of the doors is a dangerous tiger and behind the other a treasure. The agent cannot observe the location of the tiger, but it can execute listening actions which reveal the location of the tiger with some probability of noise. An episode ends when the agent opens a door, with the goal being to open the door to the treasure.

This problem fits nicely into the idea of hindsight knowledge. *After* the agent opens a door and ends the episode, it is immediately obvious where the tiger was located based on the reward received. In the tiger problem, hindsight can be used to label past experience, producing trajectories that contain observations and the true states. These are used to train both the model and Q-value function in batches.

A PRESTO agent is compared to a Q-learning agent and a deep Q-learning (DQN [4]) agent that both observe a history of observations which form the new state space. The history-based space grows exponentially with the history length and a tabular agent can quickly outgrow the memory capacity of a standard machine. The deep Q-learning agent serves as a second baseline as it uses the same neural network as the PRESTO agent to compress the large history-based observation space, avoiding the memory issues of the tabular agent.

The PRESTO agents use a multi-layer perceptron as a classifier that predicts the true location of the tiger, trained on experiences labelled in hindsight. The class probability output by the model is used as the measure of confidence in the prediction; it is discretized into three bins ([0 - 0.50), [0.50 - 0.99), [0.99 - 1.0]). While this can introduce some approximation errors, it also makes the solution more compact, ensuring the method remains scalable [15]. The state space of the PRESTO agent grows linearly with the number of bins used in this discretization, thus its size is independent of the length

of history. The input to predictor is the same history as the one used by the tabular agent and the deep Q-learning agents.

PRESTO agents and history-based agents with different history input lengths k were compared in versions of the problem with varying noise probabilities. All parameters were tuned individually to ensure good performance for each agent. Hyper-parameters used with the DQN and PRESTO agents are provided in Table 4.1.

Table 4.1: Experimental hyper-parameters for PRESTO agents in tiger domain

Parameter	Noise		
	0.15	**0.20**	**0.25**
Training frequency (steps)	30	50	50
Batch size	50	50	100
k (Hidden layer size)	4 (20,)	5 (50,)	6 (60,)
	5 (25,)	6 (60,)	7 (84,)
	6 (36,)	7 (70,)	8 (112,)
Learning rate	0.01	0.001	0.0005

The performance of Partially-Observable Upper Confidence Bound (PO-UCT) [6] is also plotted, which was implemented using the `pomdp_py` library [16]. The performance of PO-UCT is not achievable by a reinforcement learning agent, as applying it requires complete knowledge of the observation, transition and reward functions, thus it simply gives some reference of an upper performance bound. The expected return of the policy returned by PO-UCT is plotted after 15000 simulations.

PRESTO FINDS A BETTER POLICY

The results of the first experiments are shown in Figure 4.6 where only the best performing value of k corresponding to each agent for each noise probability is shown. The PRESTO agents converge to a better performing policy in each of the tasks. The performance of the tabular and deep Q-learning agents are very similar (with the lines almost completely overlapping), and both initially achieve a faster increase in performance, but ultimately they do not converge to the same policy found by the PRESTO agent within the 500 episodes shown. The initial speed-up in this task may to be due to the higher learning rates these agents can use, whereas the PRESTO agent requires a lower learning rate, as it experiences oscillatory behaviour with similarly high learning rates. It is believed that this occurs because two learning processes are happening at once (training the predictor and training the policy); when the predictor converges, the learning of the policy becomes more stable. The PRESTO agents all avoid the lowest rewards at the very start of learning, as this approach enables them to quickly plan against the worst case scenario (where uncertainty is high).

THE PREDICTOR UNCERTAINTY CONVERGES TO THE ENVIRONMENT STOCHASTICITY.

Another experiment investigates empirically whether the uncertainty output of the supervised learner is converging to the irreducible uncertainty in the environment as expected. In Figure 4.7, the prediction probability for hidden state “tiger left" is plotted

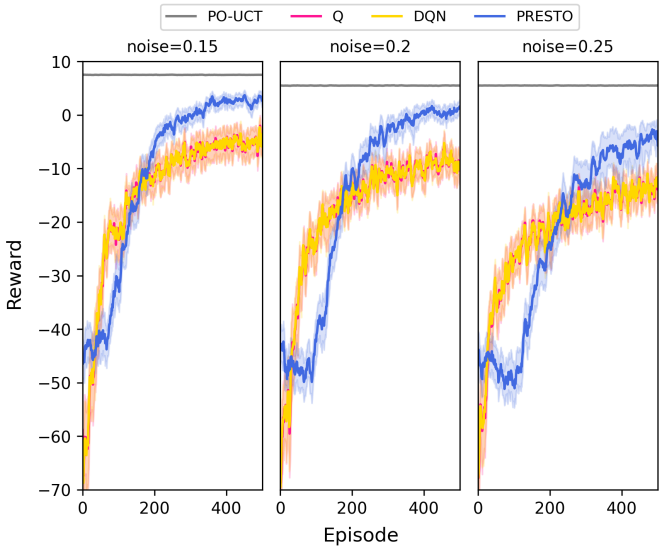

Figure 4.6: Results comparing PRESTO agents to history-based agents in the tiger domain for three different noise probabilities and history lengths 6, 7 and 8 respectively. Total episode rewards are averaged over 100 trials with the 96% confidence interval depicted by the shaded region. The plot is smoothed by a moving average window of 10.

against training iterations of the predictor ϕ. In the top plot, it is seen that the predictor cannot improve beyond 50% certainty when the agent has not received any observations; this aligns with the initial distribution of the two hidden parameter values (uniformly random). When the agent has heard the tiger behind the left door 10 times in a row, the predictor is almost certain (this is expected to converge to $1-0.10^{10}$). When the agent has received one observation that the tiger was heard behind the left door, the certainty converges to around 90%. This aligns with the noise in the environment, i.e. the probability of incorrectly observing where the tiger is.

In the bottom plot of Figure 4.7, this last case is investigated further. The prediction probability given a single observation of the tiger behind the left door is shown against training iterations for four different noise probabilities exhibited in the environment. The uncertainty reflects the environment noise, albeit with a small deviation from the theoretical expectations.

TRANSFERRING THE PRESTO POLICY CAN SPEED-UP LEARNING IN NEW TASKS

In another experiment, transfer of the policy is demonstrated considering two versions of the tiger domain where the noise probabilities differ. The policy is first trained by applying PRESTO to the tiger task with noise probability 0.15. The Q-values are then transferred to another PRESTO agent faced with a new task where the probability has been changed to 0.20. Only the predictor is retrained in the second task while the Q-values remain fixed; this also means that exploration is not necessary. The PRESTO-PolicyTransfer agent to another PRESTO agent that learns a new policy from scratch to

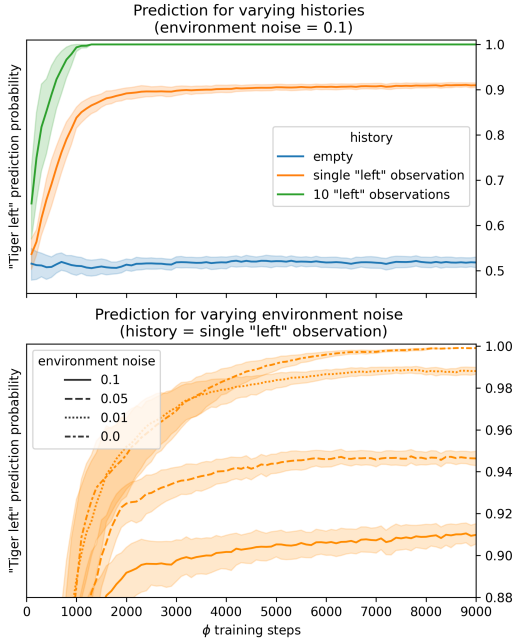

Figure 4.7: Visualizing the uncertainty output of the supervised learner ϕ averaged over 25 trials with shaded areas depicting the 98% confidence interval.

show the effectiveness of the transfer. All non-transfer agents were ensured start with as low an exploration factor as possible and it was found that setting an initial ϵ of 0.1 resulted in the best performance; higher and lower values affected the speed of convergence negatively.

The results of the transfer learning experiment are presented in Figure 4.8. There is a considerable benefit to using the pre-trained policy in the new environment (even in this small problem), and it converges to the same performance as the PRESTO agent trained from scratch.

4.4. RELATED WORK

The idea of learning a POMDP policy on a prediction of the true state has been mentioned as the “Most Likely State" heuristic [17]. This is expanded on by learning on a prediction (output by a supervised learning model) and confidence, addressing the issues that otherwise arise, and bringing it to the reinforcement learning setting. Predictive State Representation (PSR) [3] has a similar goal of bypassing the limitations of the POMDP formulation and conducting policy optimization over a compact state, however PSR requires the definition of tests and cannot make explicit which state the agent thinks it is in (as no labeled data is ever available).

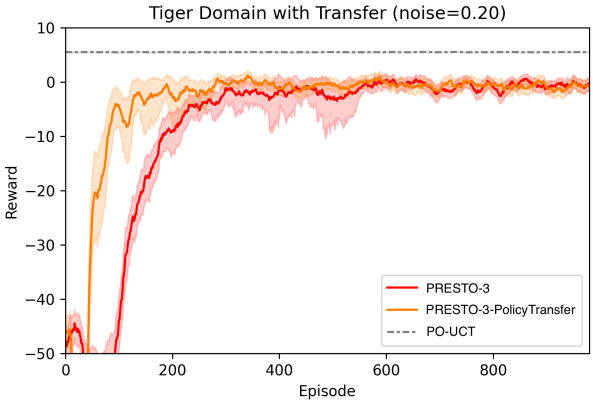

Figure 4.8: Results for transfer of the policy learned from the tiger domain with 0.15 probability of noise and applied to a new task where noise probability is changed to 0.20. The average of 50 trials is shown with shaded areas depicting the 98% confidence interval. The plot is smoothed by a moving average window of 5.

The proposed approach has similarities to learning on a latent space or with hidden parameters. Other methods that model learning with hidden parameters commonly involve reasoning over the latent parameter space to facilitate transfer learning between related tasks [18–20] or reason about an opponent strategy modeled by those latent variables [21]. This setting is conceptually related to the idea of influence-based abstraction [22]. The hidden observations can be considered as the 'influence source' and the supervised learning model ϕ is related to the idea of an influence predictor [23]. However, this method explicitly outputs a measure of uncertainty that is used to encourage information-gathering actions in the policy. Deep anticipatory networks are another approach that focus on this goal of information-gathering, combining the goals of maximizing prediction rewards and information gain [24]. The method presented here considers the case where extra information is available during training and provides a mechanism to make use of it. A similar setting also motivates an actor-critic method that makes use of additional latent information that is only available during offline training, but this method does not consider explicit state prediction [25].

There has been recent emergence of methods that label experience after action execution and use this labelled experience to learn. Hindsight Experience Replay involves saving trajectories in memory in order to reuse the experience after re-labeling these trajectories according to different goals they may be better suited for [26]. This approach has been extended to policy gradient methods [27] and was later generalized to tasks which are not specifically goal-oriented by applying techniques from inverse reinforcement learning [28]. These approaches have in common that they label saved trajectories in order to reuse data and learn multiple tasks more efficiently. In their case, the labels are reflected in the rewards of a trajectory, which differ between tasks while the transition dynamics are unaffected. In contrast, this work considers additional knowledge not in terms of the goal but in the form of state information. This enables the PRESTO agent to benefit from information that is not there at the time of action selection, which

may come in the form of hindsight knowledge but can also be artificially removed during training in order to create a more robust policy.

4.5. CONCLUSION

This chapter introduced a modular reinforcement learning algorithm for POMDPs named Uncertainty-Aware Predicted State Reinforcement Learning (PRESTO). PRESTO learns on the joint state space of predicted states and confidence in the prediction, finding a policy that trades off information gathering with reward maximization. This method is suitable for problems which are partially-observable at the time of action selection, but can be labelled with full observability during training. By incorporating a predictive model, a PRESTO agent makes explicit its knowledge of the state it is currently in, information which is otherwise lost in the policy optimization of a traditional RL agent. Further, taking a decoupled approach facilitates transfer of both the policy and the predictor between different tasks. The conditions that must be satisfied in order to safely transfer are explicitly stated, something which is often hard to determine in transfer learning problems but is necessary to avoid negative transfer [29].

PRESTO is promising as a method that can enable designers to apply predictors or policies learned on offline data-sets to further improve the efficiency of reinforcement learning agents. One interesting question that remains is whether a predictor learned on an offline data-set would be as subject to the same issues of data distribution mismatch (which plague other offline RL approaches). The proposed approach offers one way to make use of offline data while making explicit the confidence the model has in its predictions, possibly being less susceptible to such issues.

The PRESTO approach combines the strengths of model-free and model-based reinforcement learning, using a supervised learner to build a partial model of the unobservable state components, and applying model-free policy optimization on this prediction. By creating a joint state space of the predicted state and uncertainty in the prediction, the agent learns a policy that tradesoff information gathering with reward maximization. However, PRESTO is also limited by some of the same issues facing model-based learning, namely the difficulty of learning a model to predict states. As demonstrated in the cartpole experiments, one way to reduce this burden is to model the problem as a decoupled POMDP and instead learn on the joint state of the observation, predicted state and prediction confidence. Another interesting direction to explore is whether this method can use the supervised learning model to learn a compact representation of state that preserves sufficient information without the overhead of learning the true state, something left for future work. Overall, the PRESTO approach is an important step in developing methods that do not rely solely on learning from scratch, by either labelling data in hindsight, or purposefully perturbing training environments to hide information that is unobservable in the real-world.

REFERENCES

[1] C. T. Ponnambalam, D. Kamran, T. D. Simão, F. A. Oliehoek, and M. T. J. Spaan, *Back to the Future: Solving Hidden Parameter MDPs with Hindsight,* in *Adaptive*

and Learning Agents Workshop at the 21st International Conference on Autonomous Agents and MultiAgent Systems (2022).

[2] W. A. Johnston and V. J. Dark, *Selective Attention,* Annual Review of Psychology **37**, 43 (1986).

[3] M. Littman and R. S. Sutton, *Predictive Representations of State,* in *Advances in Neural Information Processing Systems*, Vol. 14, edited by T. Dietterich, S. Becker, and Z. Ghahramani (MIT Press, 2001).

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing Atari with Deep Reinforcement Learning,* arXiv (2013), arXiv:1312.5602.

[5] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, *Planning and Acting in Partially Observable Stochastic Domains,* Artificial Intelligence **101**, 99 (1998).

[6] D. Silver and J. Veness, *Monte-Carlo Planning in Large POMDPs,* in *Advances in Neural Information Processing Systems*, Vol. 23, edited by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (Curran Associates, Inc., 2010).

[7] S. Ross, B. Chaib-draa, and J. Pineau, *Bayes-Adaptive POMDPs,* in *Advances in Neural Information Processing Systems*, Vol. 20, edited by J. Platt, D. Koller, Y. Singer, and S. Roweis (Curran Associates, Inc., 2007).

[8] B. Lakshminarayanan, A. Pritzel, and C. Blundell, *Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles,* in *Advances in Neural Information Processing Systems*, Vol. 30 (Curran Associates Inc., 2017) p. 6405–6416.

[9] J. Smit, C. T. Ponnambalam, M. T. Spaan, and F. A. Oliehoek, *PEBL: Pessimistic Ensembles for Offline Deep Reinforcement Learning,* in *International Joint Conference on Artifical Intelligence, Workshop on Robust and Reliable Autonomy in the Wild (R2AW)* (2021).

[10] C. J. C. H. Watkins and P. Dayan, *Q-Learning,* Machine Learning **8**, 279 (1992).

[11] G. Tennenholtz, U. Shalit, and S. Mannor, *Off-Policy Evaluation in Partially Observable Environments,* Proceedings of the AAAI Conference on Artificial Intelligence **34**, 10276 (2020).

[12] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee, *Planning under Uncertainty for Robotic Tasks with Mixed Observability,* International Journal of Robotics Research **29**, 1053–1068 (2010).

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *OpenAI Gym,* (2016), arXiv:1606.01540.

[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, *Stable-Baselines3: Reliable Reinforcement Learning Implementations,* Journal of Machine Learning Research **22**, 1 (2021).

[15] N. Roy, G. J. Gordon, and S. Thrun, *Finding Approximate POMDP solutions Through Belief Compression,* Journal of Artificial Intelligence Research **23**, 1 (2005).

[16] K. Zheng and S. Tellex, *pomdp_py: A Framework to Build and Solve POMDP Problems,* in *Proceedings of the International Conference on Automated Planning and Scheduling, Workshop on Planning and Robotics (PlanRob)* (2020).

[17] M. T. J. Spaan, *Partially Observable Markov Decision Processes,* in *Reinforcement Learning: State-of-the-Art*, edited by M. Wiering and M. van Otterlo (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 387–414.

[18] T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez, *Robust and Efficient Transfer Learning with Hidden Parameter Markov Decision Processes,* in *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017).

[19] S. Saemundsson, K. Hofmann, and M. P. Deisenroth, *Meta Reinforcement Learning with Latent Variable Gaussian Processes,* in *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (2018).

[20] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, *Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables,* in *Proceedings of the 36th International Conference on Machine Learning* (PMLR, 2019) pp. 5331–5340.

[21] A. Xie, D. Losey, R. Tolsma, C. Finn, and D. Sadigh, *Learning Latent Representations to Influence Multi-Agent Interaction,* in *Proceedings of the 2020 Conference on Robot Learning*, Vol. 155, edited by J. Kober, F. Ramos, and C. Tomlin (PMLR, 2021) pp. 575–588.

[22] F. A. Oliehoek, S. Witwicki, and L. P. Kaelbling, *A Sufficient Statistic for Influence in Structured Multiagent Environments,* Journal of Artificial Intelligence Research **70**, 789 (2021).

[23] J. He, M. Suau, and F. A. Oliehoek, *Influence-Augmented Online Planning for Complex Environments,* in *Advances in Neural Information Processing Systems*, Vol. 33 (Curran Associates, Inc., 2020) pp. 4392–4402.

[24] Y. Satsangi, S. Lim, S. Whiteson, F. A. Oliehoek, and M. White, *Maximizing Information Gain in Partially Observable Environments via Prediction Rewards,* in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems* (IFAAMAS, Richland, SC, 2020) p. 1215–1223.

[25] A. Baisero and C. Amato, *Unbiased Asymmetric Reinforcement Learning under Partial Observability,* in *Proceedings of the 21st International Conference on Autonomous Agents and MultiAgent Systems* (IFAAMAS, Richland, SC, 2022) p. 44–52.

[26] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, *Hindsight Experience Replay,* in *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V.

Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017).

[27] P. Rauber, A. Ummadisingu, F. Mutz, and J. Schmidhuber, *Hindsight Policy Gradients,* in *International Conference on Learning Representations* (2019).

[28] A. Li, L. Pinto, and P. Abbeel, *Generalized Hindsight for Reinforcement Learning,* in *Advances in Neural Information Processing Systems*, Vol. 33, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Curran Associates, Inc., 2020) pp. 7754–7767.

[29] M. E. Taylor and P. Stone, *Transfer Learning for Reinforcement Learning Domains: A Survey,* Journal of Machine Learning Research **10**, 1633 (2009).

5

ABSTRACTION-GUIDED POLICY RECOVERY FROM EXPERT DEMONSTRATIONS

சென்ற விடத்தாற் செலவிடாத் தீதொரீஇ
நன்றின்பா லுய்ப்ப தறிவு.
- திருக்குறள் 422

Without letting the mind to wander on its own will,
wisdom steers it away from harm, and towards good.
- Tirukkuṟaḷ 422

Reinforcement learning is an attractive approach to automating decision-making in increasingly complex systems. As highlighted, learning methods often fall short of meeting the requirements of real-world problems. One of the main limitations to this is the need to take random actions, which in several domains, particularly safety-critical applications, is not feasible. However, learning algorithms get much of their power from this ability to explore. The case where no exploration is possible and agents can only learn from a given data set is the offline reinforcement learning paradigm. Offline reinforcement learning is notoriously difficult, as the agent cannot use exploration to fill in gaps in its knowledge.

One way to assuage the difficulties of offline RL is to incorporate demonstrations by an expert and train agents to copy and deploy exemplar behavior with as little human supervision required as possible. This is the idea behind behavioral cloning [2], where the agent is expected to directly replicate the demonstrated behavior rather than develop its own understanding of how to complete the task. This is the case considered in this chap-

Elements of this chapter are published in [1].

ter; an expert has provided examples of correct behavior that and the goal is to automate the decisions of the expert without any exploration or further learning done in the real environment.

In a typical behavior cloning scenario, a predictive model can be used to supply expert actions trained on the example data provided by the expert [3]. As with any predictive model, the ability to generalize over data never seen before is not guaranteed. If the model is used for planning, the error will accumulate over every planning step. It may be infeasible for a human to define the correct behavior for every state even in relatively small problems. When presented with a state far from its expert data set, an agent can either follow its potentially very incorrect model or take random actions, both of which may be undesirable. Another option is to facilitate human intervention whenever the agent is lost; this is effective but compromises the overall autonomy of the system. All of these issues limit the applicability of behavior cloning.

The method presented in this chapter aims to remedy this by producing a robust behavior cloning method that can recover from parts of the state space not covered by its imitation policy. State abstraction is applied to expert trajectories, effectively multiplying the available data. This new data set is then used to generate a simulator for the problem of recovering the agent to a state covered by its imitation policy. Optimizing for expected reward in the simulator results in a recovery policy which does this optimally. Instead of generalizing over the entire decision-making task, the agent focuses on the simpler problem of getting back to states covered by the demonstrations, thus minimizing the effect of model error. Even with a biased data set (containing only successful trajectories) and without making potentially dangerous conclusions about never-before-seen states, the lost agent can be recovered back to a known policy.

This method, named RECO (from policy RECOvery), boosts the ability of behavior cloning agents and makes efficient use of offline data. It is aimed at problems where expert demonstrations have limited coverage of the problem space and where un-modelled effects may take an agent to a state outside of the given data. For example, an autonomous vehicle (AV) may encounter construction on the road and be forced to a state outside of its planned route. A RECO AV will adjust its goal to first aim to get back on track, using data from previous routes to do so, and then continue to execute the original plan.

In this chapter, the policy recovery problem is formally defined and it is demonstrated how data from expert trajectories can be used to specify and solve it. This chapter includes a discussion on criteria for suitable abstraction selection and how to use several candidate abstractions to generate the best recovery policy. The performance of RECO is assessed against several baselines in tabular and continuous problems, demonstrating its ability to make use of fewer trajectories while maintaining good performance in the environment. Finally, related methods in the literature are discussed and the chapter concludes with several ideas for extending this work.

5.1. RECO: ABSTRACTION-GUIDED POLICY RECOVERY

The aim in this work is to automate the process of making decisions in an environment given examples of good behavior provided by an expert. Trajectories of expert decisions include states and corresponding expert actions, but given the size of realistic problems,

do not cover the entire space of possible states and actions. The behavior cloning agent must use the given data to compute a policy that will be deployed online without any additional data collection or computation.

The first focus is on the discrete state-space (tabular) case, where the lack of function approximation renders a behavior cloning agent policy-less in states outside the data set. It is assumed that a human supervisor is available to provide an action when the agent has no policy to follow. The goal of the RECO method in these problems is to reduce the number of calls to the human supervisor using only the data provided by the expert without a substantial loss in performance.

In subsequent sections, the method is extended to continuous state-space problems and the modifications required to do so are introduced. With the help of function approximation in continuous domains, the agent can use its model to choose actions from a state even if the expert has not visited the same state; however, as demonstrated in the experiments, this can lead to sub-optimal behavior when the model is inaccurate. Here the goal is to recover the agent from states where the imitation policy is uncertain due to lack of coverage by the expert data.

5.1.1. PROBLEM FORMALIZATION

The problem considers optimizing a task represented as a Markov decision process, where the state and action space are known but the transition and reward function are unknown. The data set of trajectories provided by an expert is in the form

$$D = \{(s_1, a_1, s_2), (s_2, a_2, s_3), ..., (s_T, a_T, s_{T+1})\}$$

containing T (state, action, next state) tuples. The set of states found in D with actions provided by the expert is called S_D, where $S_D = \{s \in S \mid (s, a, s') \in D\}$ and $S_D \subseteq S$. An imitation policy $\pi_D(s)$ that aims to directly copy the expert behavior is defined for all $s \in S_D$.

An abstraction mapping function

$$\mu : S \mapsto \bar{S}$$

maps the state space S to a state space $\bar{S}$, where $\bar{S} \subseteq S$. It is assumed that the state space of the task is factored, thus the state space S is the space spanned by the domains of k state variables [4]:

$$S = \{S_0 \times ... \times S_{k-1}\}.$$

The factored MDP description facilitates hand-design of state abstractions where $\mu(s)$ is a masking function that returns only the variables in s that are relevant to the recovery task: $\mu(s) = \bar{s}$. The inverse function $\mu^{-1}(\bar{s})$ returns the set of ground states that map to the abstract state $\bar{s}$ under the mapping function μ.

RECO uses state abstraction to generate data for learning a recovery policy π_ρ that returns an agent to a state in the expert trajectories from states outside S_D.

EXAMPLE

Recall that in the classical taxi problem, a taxi agent in a grid world is tasked with picking up a passenger from a given location and dropping them off at their destination [5]. The

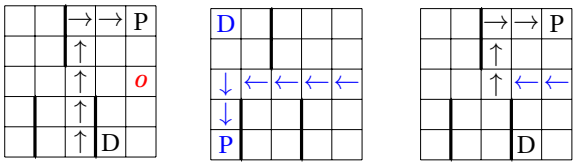

Figure 5.1: Examples of expert trajectories are shown in (a) and (b), given for two problem instances of the classical taxi domain where the grid location is the location of the taxi, P indicates the passenger location, and D the destination of the passenger. The RECO policy for initial state o is shown in (c), learned from the data in (a) and (b).

variables in the factored state description indicate the x- and y-coordinate of the taxi and the locations of the passenger and destination, such that $s = \{x, y, passenger, destination\}$. The actions available are $A = \{south, north, east, west, pickup, drop\text{-}off\}$. Figures 5.1(a) and (b) indicate given expert behavior for different locations of the taxi in the grid, when the passenger and destination locations are as shown. Suppose the agent finds itself at location o, pictured in red, with the passenger and destination shown in black. For this particular state of the passenger and destination, the expert has not provided an example, only the path given by the black arrows in Figure 5.1(a). However, another trajectory, shown as the blue arrows in Figure 5.1(b), for a different configuration of passenger and destination does contain this taxi location. By ignoring irrelevant information (the passenger and destination), the agent can reason about actions that return it to a known state. A RECO agent leverages the actions in the blue path, taking them from its position at o and getting back to the known policy (black path) which it then executes, depicted in Figure 5.1(c). It does this by applying what it knows in the abstract state space (x- and y-position) to get back to a known state. In this case, $\mu = \{x, y\}$.

First, the mechanics of RECO when a single state abstraction is provided are described. In the section titled Abstraction Selection, the conditions for suitable abstractions are layed out, and RECO is adapted to the case where several candidate abstractions have been provided and can be used in parallel.

5.1.2. RECO IN TABULAR DOMAINS

In tabular domains, the imitation policy $\pi_D(s)$ samples an action a according to a weighted distribution, weighted by the number of times a was executed by the expert from state s. This generalizes to the case where the expert policy may be stochastic. The environment is considered to be deterministic to simplify notation, and the extension to stochastic domains is described later in this section. In a deterministic setting, the agent can find itself in an unknown state only when an episode begins according to the initial state distribution. The given abstraction mapping function μ is applied to each state entry in the data set to get an abstracted data set

$$\bar{D} = \{(\bar{s}_1, a_1, \bar{s}_2), (\bar{s}_2, a_2, \bar{s}_3), ..., (\bar{s}_T, a_T, \bar{s}_{T+1})\}.$$

The recovery problem of getting back to a state in the data set from a state outside of S_D is modelled as an MDP.

RECOVERY MDP

The recovery MDP M_ρ is defined over the entire state space S augmented with an additional sink state z. The action space is copied from the original MDP. Any transition from a state in the expert trajectories transitions to the sink state, as the goal of recovery has been reached. Abstract transitions are projected onto all the ground states that map to the abstract state; this means the abstract transitions are copied for all possible combinations of the un-abstracted variables. Transitions that are not represented in the abstracted data set also transition to the sink state, as there is no data available for these. The reward for taking action a_t from state s_t is 1 if s_t is present in S_D, otherwise 0.

Definition 1 (Recovery MDP M_ρ for tabular domains)**.** Given a deterministic MDP, a data set D generated by an expert policy and an abstraction function μ, recovery MDP $M_\rho = (S_\rho, A_\rho, T_\rho, R_\rho, \gamma_\rho)$ is specified as:

$$
\begin{aligned}
S_\rho &= S \cup \{z\},\\
A_\rho &= A,\\
R_\rho(s_t, a_t, s_{t+1}) &= \begin{cases} 1, \text{if } s_t \in S_D, \\ 0, \text{otherwise.} \end{cases}\\
T_\rho(s_t, a_t, s_{t+1}) &= \begin{cases} 1, \text{if } s_t \in S_D \wedge s_{t+1} = z, \\ 1, \text{else if } (\bar{s}_t, a_t, \bar{s}_{t+1}) \in \bar{D}, \\ 1, \text{else if } (\bar{s}_t, a_t) \notin \bar{D} \wedge s_{t+1} = z, \\ 1, \text{else if } s_t = s_{t+1} = z, \\ 0, \text{otherwise.} \end{cases}\\
\gamma_\rho &= (0, 1).
\end{aligned}
$$

To describe the transition function in plain terms, the agent transitions from states found in the data set to the sink state. All transitions in the abstract data set (from abstract states to abstract next states) are projected back onto the recovery transition function. All other transitions transition to the sink state, including from the sink state itself.

When the environment is stochastic, the transition function of M_ρ is defined by its maximum likelihood estimate. This means that instead of assuming abstract transitions happen with probability 1, the 1 on the second line of the transition function in Definition 1 is replaced with:

$$
\frac{\sum_{s_t^* \in \mu^{-1}(\bar{s}_t)} N(s_t^*, a_t, s_{t+1})}{\sum_{s_t^* \in \mu^{-1}(\bar{s}_t)} N(s_t^*, a_t)},
$$

where recall that $\mu^{-1}(\bar{s})$ returns all the ground states that map to the same abstract state as s.

POLICY RECOVERY USING THE RECOVERY MDP

The fully-defined recovery MDP can be solved using an off-the-shelf method such as Value Iteration, producing the recovery policy π_ρ which maximizes the expected discounted reward in the recovery problem. The recovery policy takes actions that correspond to finding the shortest path back to a state in the expert trajectories. When acting online, if a RECO agent is presented with a state outside of its expert trajectories, it has the option of executing the recovery policy.

The success of recovery depends on the data given, and there may not be the ability to control how or how much data is collected. Fortunately, there is a simple way to check for which states the recovery policy is able to recover the agent. The definition of a *recoverable* policy is adapted from the *proper* policy definition applied to stochastic shortest path problems (SSPs) [6]. The recovery problem is similar to an SSP except instead of no discounting and a negative reward for every action taken, it is an infinite-horizon discounted-reward problem with a discount factor $0 < \gamma_\rho < 1$, a reward of 1 for taking any action from a goal state, and a reward of 0 everywhere else.

Definition 2 (Recoverable policy)**.** A policy π applied to M_ρ is *recoverable* over a set of states S_P if, when following π from any state $s \in S_P$, there is a positive probability that some goal state $s_d \in S_D$ will be reached in a finite number of steps.

The set of states S_P can be found as defined below.

Definition 3 (Recoverable states)**.** With the solution to a recovery MDP π_ρ, the set of recoverable states $S_P \in S$ is defined as the set of states where π_ρ has a positive value, i.e., where $V(\pi)_{s_0=s} > 0$.

With this satisfied, every state in S_P has some path under policy π_ρ that leads to a state in the expert trajectories. Another difference between the recovery problem and SSPs is that there is no guarantee that the recovery policy can recover from any state, thus it must be assessed whether a policy π_ρ is *recoverable*. Without access to the reward function, one can make no conclusions about the value of the policy in the real environment. However, it is known that the only way for a policy to have a non-zero value in the recovery MDP is if it reaches a state in the expert trajectories.

By checking the value of a policy in a particular state in the recovery MDP, non-recoverable policies can be eliminated in which the destination is not reached from that state. For states without a recoverable policy, the agent executes an emergency action that requests an action from an expert. Combined, the overall RECO policy is defined as follows.

Definition 4 (RECO policy in tabular environments)**.** A tabular RECO agent given an expert data set D and a single recovery abstraction μ follows the aggregated policy from state s_t:

$$\pi_{\text{RECO}}(s_t, D, \mu) = \begin{cases} \pi_D(s_t), \text{if } s_t \in S_D, \\ \pi_\rho(s_t), \text{if } s_t \in S_P, \\ request, \text{otherwise.} \end{cases}$$

The state spaces S_D and S_P in the taxi example are highlighted in Figure 5.2.

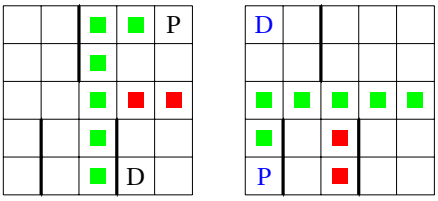

Figure 5.2: For the same two instances of the passenger and destination locations (P and D respectively) and given the expert trajectories shown in Figures 5.1(a) and 5.1(b), green indicates the states in S_D over which the imitation policy is defined and red indicates the states in S_P for which there is a *recoverable* policy.

5.1.3. ABSTRACTION SELECTION

The method requires an abstraction in the form of a subset of state variables relevant to recovery, but not necessarily relevant to finding the optimal policy in the true environment. In this section, the conditions on the abstraction are introduced that ensure that a *recoverable* policy will recover an agent in the ground MDP. This includes a discussion on emergent trade-offs in choosing an appropriate abstraction as well as how to manage these trade-offs.

Two important concepts that require introduction are recovery-relevant actions and model-consistency in the recovery MDP. In the recovery MDP, abstract transitions are projected back onto ground states, transitioning only the abstracted variables according to the data set. A recovery-relevant action is defined as one that, in the recovery MDP (which is dependent on the data set), results in a change of state with a non-zero probability (excluding transitions to or from the sink state). This leads to the following definition:

Definition 5 (Recovery-relevant actions)**.** The set of recovery-relevant actions Δ for M_ρ is defined as:

$$\Delta_{M_\rho} = \{a \in A_\rho \,|_{\exists s^* \in S_\rho}\, T_\rho(s, a, s^*) > 0 \wedge s \neq s^* \wedge s \neq z\}.$$

In other words, any action that does not have any effect on the abstract state is irrelevant to recovery. The set of recovery-relevant actions is a function of both an abstraction mapping and the expert trajectories to which it is applied. In the taxi problem, if the abstraction $\mu = \{x, y\}$ is selected, then the pickup and drop-off actions are recovery-irrelevant as they have no effect on the x- or y-coordinate of the agent.

Model-consistency refers to whether the behavior in the abstract space is obeyed for all the ground states which map to an abstract state. The bar notation is again used to simplify notation, thus $\bar{s} = \mu(s)$.

Definition 6 (Model-consistent recovery MDP)**.** A recovery MDP M_ρ is model-consistent with the ground MDP M if the transition function agrees for all *relevant actions* in all ground states that map to the same abstract state (excluding transitions to the sink state z):

$$T_O(s_t, a_t, s_{t+1}) = T(s_t^*, a_t, \bar{s}_{t+1}),$$
$$\text{where } s_{t+1} \neq z, \forall a_t \in \Delta_{M_\rho} \forall s_t^* \in \mu^{-1}(\bar{s}_t).$$

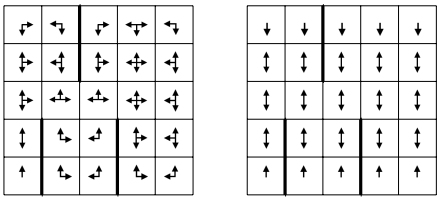

Figure 5.3: Recovery-relevant actions from the expert trajectories for the $\{x, y\}$ (left) and $\{y\}$ (right) abstractions projected back onto the taxi domain where the expert data set covers solutions for all possible locations of passenger and destination.

The method assumes the transition model in the recovery MDP is model-consistent with the transitions in the ground MDP for relevant actions. With this criteria met, it is guaranteed that a *recoverable* policy will return the agent in the ground MDP to a state in the expert trajectories. In the taxi example, the model-consistency criteria is not met by any abstraction that includes $\{x\}$ without $\{y\}$ because of the walls in the environment; in some cases, a right movement will take the agent right but if a wall is present, it will not. It is assumed that expert knowledge of the domain can produce abstractions that meet this criteria, though they can also be determined empirically with some confidence depending on the amount and coverage of expert data given. It is also important to note that if the domain of any state variable is increased (exponentially increasing the size of the state space), the abstraction definition will remain fixed and no further burden is put on the designer.

DYNAMIC POLICY RECOVERY WITH PARALLEL ABSTRACTIONS

It is possible that several abstractions fit the criteria for model-consistency. A finer abstraction (i.e., smaller abstract state space, more compression) might require less coverage of the state space in the expert trajectories but then have a lower recovery potential. This is depicted graphically in Figure 5.3, where it is clear that a $\{y\}$ abstraction can only recover to states with the same x-position. A coarser abstraction takes into account more information when determining the closest state, thus potentially recovering to more relevant states, however finding recoverable policies requires more coverage of the state space in the expert data. While several abstractions may fit the defined criteria for suitability, their success is highly dependent on the actual data available. This brings us to the question: given a fixed set of expert trajectories, how can one determine the best abstraction for recovery?

As the recovery MDP is defined and solved offline, there is an opportunity to generate and solve the recovery problem for several candidate abstractions. This set of policies can be used, which is called Π, to choose the most suitable for a state encountered online. Upon deployment, presented with a state s outside of the trajectories, a recovery policy is chosen from the set of policies which are *recoverable* in s. This dynamic policy recovery is presented in Algorithm 5. It is assumed that the recoverable policy produced by the coarsest abstraction is the best choice as it incorporates the most state information in planning its path, and thus lies nearer to the expert behavior. Therefore, the function `RecoveryPolicySelection()` on line 15 returns the recoverable policy produced by the abstraction with the largest abstract state space. In empirical evaluations, it is

shown how taking a dynamic abstraction approach can improve performance of a RECO agent by effectively trading off abstraction size with the coverage of the data set.

Algorithm 5: Dynamic Tabular RECO

Result: Policy π_{RECO}
Given a set of expert trajectories D;
Given a set of suitable candidate abstractions μ_C;
Given number of test steps N;
Initialize empty set of recovery policies $\Pi = \emptyset$;
foreach $\mu \in \mu_C$ **do**
    $\pi_\rho = \texttt{GetRecoveryPolicy}(\mu, D)$;
    $\Pi = \Pi \cup \{\pi_\rho\}$;
end
$t = 0$;
while $t < N$ **do**
    Receive state s_t;
    if $s_t \in S_D$ **then**
        $a_t = \texttt{ImitationPolicy}(D, s_t)$;
    else if $\textit{RecoverablePolicies}(\Pi, s_t) \neq \emptyset$ **then**
        $\pi_R = \texttt{RecoveryPolicySelection}(\Pi, s_t)$;
        $a_t = \pi_R(s_t)$;
    else
        $a_t = \texttt{RequestExpert}(s_t)$;
    end
    Execute action a_t;
    $t = t + 1$;
end

5.1.4. RECO IN CONTINUOUS DOMAINS

In this section, RECO is adapted to continuous-state discrete-action problems. With the use of function approximation and the now continuous state-space, the assumptions and formal definitions presented in tabular RECO are no longer applicable. This section is meant to touch on the changes necessary in adapting tabular RECO to continuous problems as well as their current limitations. Continuous RECO is presented assuming a single abstraction has been given, though in principle, dynamic switching between abstractions could still be applied.

MODEL DEFINITION

The imitation policy is learned with a supervised approach, predicting expert actions for a given state, trained on the data in D. This policy is again called π_D. As in the tabular version, a given abstraction function $\mu(s)$ is applied to the data set resulting in abstracted trajectories $\bar{D}$.

To gauge how close an encountered state is to the expert trajectories, a distance mea-

sure $d(s, D)$ is calculated. For this measure, a simple weighted L1 norm is used where the weight vector is the size of the number of state variables. This returns the norm between state s and the closest state in D (i.e., the smallest weighted norm). A second distance measure $d_a((s, a), D)$ returns the distance between a (state, action) pair and the trajectory. For d_a, the norm is calculated considering only states in the trajectory from which action a was taken. Many other distance measures are possible and can replace the one described [7–9]. In general, the issue of determining when a state is out-of-distribution is an orthogonal problem to the one presented here.

The transition and reward functions (see Definition 7) of the continuous recovery problem can no longer be represented in tabular form. However, the transitions must be captured in the abstract space, thus a supervised learning model is trained on $\bar{D}$ to predict abstract next states from abstract states and actions. This predictor is called ϕ_T, where $\phi_T(\bar{s}_t, a_t) = \bar{s}_{t+1}$. In the tabular setting, the abstract transition was copied for all possible combinations of the un-abstracted variables to effectively multiply the data set, whereas enumeration over a discrete number of combinations is no longer possible in the continuous case. For simplicity, an assumption is instead made that the abstract state will transition according to ϕ_T and the other variables remain unchanged. The transition function T'_ρ thus takes a full state s_t and action a_t and transitions only the abstract variables, resulting in s_{t+1}.

The reward function takes a (s_t, a_t, s_{t+1}) tuple and gives a reward of 1 for entering a state only if the distance between it and the closest state in the expert trajectories is lower than a threshold ζ_D. The reward is 0 if $(\bar{s}_t, a_t)$ is within a second distance threshold ζ_μ of the closest abstract state in the abstract data set and -1 if neither is true. An episode ends if the agent reaches such a state or gets a reward of -1.

This leads to the following definition.

Definition 7 (Recovery MDP M'_ρ for continuous domains)**.**
Recovery MDP $M'_\rho = (S'_\rho, A'_\rho, T'_\rho, R'_\rho, \gamma'_\rho)$ is defined as:

$$
\begin{aligned}
S'_\rho &= S, \\
A'_\rho &= A, \\
T'_\rho(s_t, a_t, \phi_T(\bar{s}_t, a_t)) &= s_{t+1}, \\
R'_\rho(s_t, a_t, s_{t+1}) &= \begin{cases} 1, \text{if } d(s_{t+1}, D) < \zeta_D, \\ 0, \text{else if } d_a((\bar{s}_t, a_t), \bar{D}) < \zeta_\mu, \\ -1, \text{otherwise}, \end{cases} \\
\gamma'_\rho &= (0, 1).
\end{aligned}
$$

With the transition and reward function defined, the recovery environment can be simulated. First, a random starting state is initialized by sampling a state from the expert trajectories and replacing the abstracted variables (those in mapping μ) with a uniformly random value within their bounds. Given an action, the environment transitions according to the transition function, producing a next state and reward from the reward function. In practice, it was found that learning can be sped up if an episode ends when

a maximum number of steps is reached with a large reward penalty. The simulator can be used to solve the recovery problem with a standard continuous state-space, discrete action-space reinforcement learning method, obtaining π_ρ. For the continuous experiments presented in the next section, a variant of Proximal Policy Optimization was used [10].

In the continuous case, function approximation is commonly used to generalize the policy to states outside the trajectories, thus avoiding the need for a human supervisor, activating the recovery policy when the agent is far from the expert trajectories. As in the tabular case, the recovery policy is used only if it has a positive value in the simulated MDP, meaning that it found a path from the particular state the agent is in to a state appropriately close to the trajectories. This also stops us from trusting the recovery policy if the simulator was not run long enough to solve for a particular state. The space which spans the states over which the recovery policy is *recoverable* is again called S_P. The combined RECO policy is defined below.

Definition 8 (RECO policy in continuous environments)**.** A continuous RECO agent given data set D, abstraction mapping function μ, and distance thresholds ζ_D and ζ_μ takes an action from s_t according to the following policy:

$$\pi_{\text{RECO}}(s_t, \mu, D, \zeta_D, \zeta_\mu) = \begin{cases} \pi_D(s_t), \text{if } d(s_t, D) < \zeta_D, \\ \pi_\rho(s_t), \text{if } s_t \in S_P, \\ \pi_D(s_t), \text{otherwise.} \end{cases}$$

The recovery policy is used to safely guide the agent closer to the given expert trajectories when the agent is in a state far from the given data and it is confident that the recovery policy can do so.

5.2. EMPIRICAL EVALUATION

Experiments were conducted in the classic taxi problem [5] and a continuous problem where the agent is tasked with navigating to a given location in an environment with obstacles. In each trial, the learning for each agent was done offline on the same set of given expert trajectories (a trajectory is one episode). Agents were then deployed and evaluated on their performance online in the test environment, initialized to a random starting state every episode ensuring each agent is given the same starting state.

5.2.1. TABULAR EXPERIMENTS

The taxi problem consists of a 5x5 grid and has 4 possible passenger locations (plus 1 for when the passenger is in the taxi) and 4 destination locations, leading to a total of 500 states. Expert trajectories were generated by initializing the agent in a random state and following a trained and converged Q-learning greedy policy until episode termination.

The performance of four RECO agents is compared: three use a single abstraction and the last is the dynamic approach which chooses the best abstraction for every state according to Algorithm 5. All RECO agents use a discount factor of 0.95 in solving the recovery MDP. The RECO agents are then compared to three baselines on their performance in 100 trials of 100 test episodes. The Imitation agent follows the imitation policy

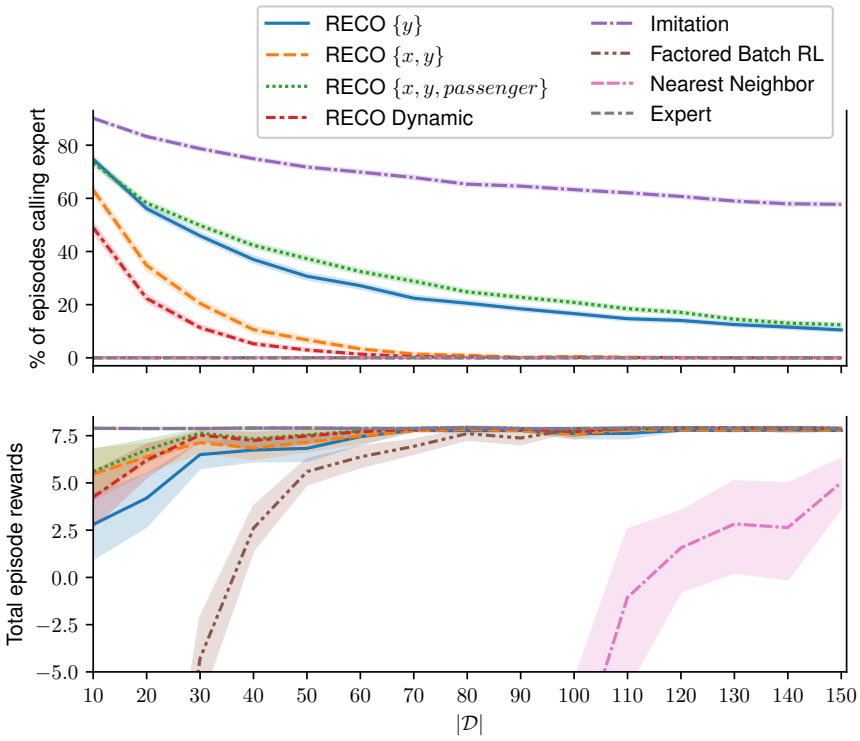

Figure 5.4: Performance on 100 test episodes of the 5x5 taxi problem averaged over 200 trials of random initial states. 98% confidence interval indicated by shaded regions.

and calls for help in any state outside the expert trajectories given. The Nearest Neighbor agent uses a crude nearest neighbor approach to choose actions when outside the expert trajectories, sampling an action from the nearest Cartesian state in the given data set. In order to maximize the performance of this simple model, the agent uses the nearest abstract Cartesian state according to the abstraction function that gave the best performance in experiments (in this case: $\{x, y\}$). The Factored Batch RL approach is a method that has been used as a baseline in other offline RL work [11]. It uses complete knowledge of the transition dynamics in the form of a dynamic Bayesian network and calculates the parameters of the model with the offline data. This baseline acts as the upper bound of offline learning performance in a factored MDP.

RESULTS

Figure 5.4 presents the results from the tabular experiment. The top plot shows the percent of episodes in which at least one request for an expert action was made and the bottom plot displays the average episode reward over 100 test episodes. Note that the Expert, Nearest Neighbor and Factored Batch RL agents never call the expert. All RECO agents show substantially fewer calls to the expert than the Imitation agent, with the Imitation agent calling for an expert in 60% of test episodes even after 150 trajectories are

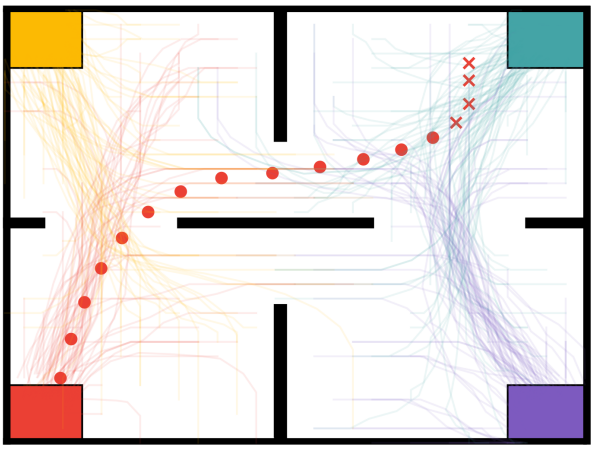

Figure 5.5: Actions taken by the RECO agent in one test episode of the continuous problem. Translucent lines depict the 100 expert trajectories provided to the agent, where color denotes the goal location. Solid points are actions taken online; X marks the recovery policy and circles indicate the imitation policy.

provided. In comparing the RECO agents, the choice of abstraction has a substantial effect on the performance. As expected, there is a higher reward loss with the smallest abstraction when given few expert trajectories; this is due to the high loss of information, where recovery is planned using actions less related to the expert behavior. The largest abstraction slightly dominates the reward performance when few trajectories are provided. The biggest difference between RECO agents is seen in the number of calls to a supervisor. Dynamic RECO effectively trades off the number of calls to the expert with performance, demonstrating the fewest calls (of the RECO agents) while maintaining high rewards. It can also be seen that Dynamic RECO is on par with the upper baseline Factored Batch RL, converging to near-expert performance around 80 trajectories with almost no calls to the expert. If a threshold is allowed of around 10% calls to an expert, RECO reaches near-optimal performance with almost half the amount of data required as the upper baseline. The Nearest Neighbor approach, despite attempts to tune it to be as good as possible, performs very poorly.

5.2.2. CONTINUOUS EXPERIMENT

In the continuous problem (pictured in Figure 5.5), an agent is tasked with navigating to one of 4 possible goal locations in an environment that includes walls. There is a -1 reward every time step to encourage the agent to solve the task in as few time steps as possible and a large penalty for hitting a wall or for reaching a maximum number of time steps, all of which end the episode. The agent receives a +100 reward when it reaches the goal (also ending the episode). The factored state is made up of the variables $\{x, y, x\text{-}displacement, y\text{-}displacement, goal\ x, goal\ y\}$ and the agent must choose an action from $\{north, south, west, east\}$. Each action results in a discrete magnitude of acceleration propelling the agent in the corresponding direction, with acceleration decay-

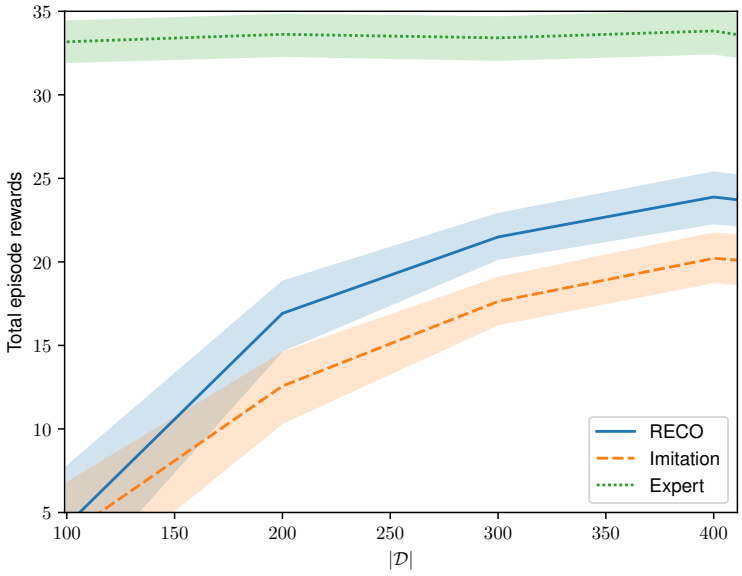

Figure 5.6: Performance on 100 test episodes of the continuous problem averaged over 200 trials of randomly initialized episodes. 98% confidence interval indicated by shaded regions.

ing each time step. The initial position is purposefully forced to be always on the same vertical half (in terms of x-position) as the goal to ensure that parts of the state space are left un-demonstrated by the expert.

The “expert" is a reinforcement learning agent trained with Proximal Policy Optimization (PPO) on 3e6 time steps of this task. In order to preserve reproducibility, all executions of PPO use the standard PPO2 implementation of the Stable Baselines library [12]. A weighting for the distance measures of $(1, 1, 0.1, 0.1, 10, 10)$ is used, meaning that the goal location is weighed heavily and the displacement much less so when considering distance between states. The distance thresholds are set to $\zeta_D = 10$ and $\zeta_\mu = 6$ and the discount factor is 0.99. The abstraction used is $\{x, y, x\text{-}displacement, y\text{-}displacement\}$. Both the imitation agent and the transition functions are trained neural networks with a single hidden layer of 100 neurons. PPO ran on the simulated recovery domain for 3e4 time steps to obtain the recovery policy. The performance of RECO is compared against the Imitation agent, which follows the imitation policy for any state encountered.

RESULTS

Figure 5.5 is a visualization of a single test episode in the continuous environment, showing how RECO guides an agent back to known states using actions that are similar to those taken before in an abstract state. The imitation policy is then executed and the task solved. The results of the experiments are plotted in Figure 5.6. The RECO agent is superior to the Imitation agent once it has enough data to leverage, indicating the success of

the recovery policy in retrieving the agent from states where its imitation model would be incorrect. Both agents converge to sub-optimal performance due to the purposeful limiting of the state space covered by the expert demonstrations. When the expert is allowed to initialize trajectories from any random state, the imitation policy converges to near-expert performance after around 1000 given trajectories and the RECO policy does not have a noticeable advantage. This emphasizes that RECO is especially suited to improve behavior cloning in problems where the expert demonstrations do not have good coverage of the state space.

5.3. RELATED WORK

In behavior cloning or imitation learning, the offline data is produced by an expert who is assumed to follow an optimal policy. The focus here is on agents that, once deployed, cannot execute random actions (exploration) in the environment nor learn from their online interactions. All offline reinforcement learning techniques must deal with the issue of what to do when the agent encounters an out-of-distribution or anomalous state during execution. This issue of *distributional shift* is a well-studied problem in the literature [13–15]. Several solutions have been proposed, such as reverting to a safe policy [16], forcefully resetting the agent [17], or requesting human intervention [18, 19]. In the problem definition, it is assumed that a safe policy is not known outside of the expert trajectories provided, that resetting the agent is not possible and that human supervision in the true environment is very costly and therefore undesirable.

The notion of recovery has recently been applied to safe reinforcement learning, where offline data is used to identify unsafe zones and a learned recovery policy steers the agent back to safe states [20], though here the goal is to satisfy safety constraints during online learning rather than follow policies learned on expert demonstrations. Another online method involves an agent simultaneously learning to perform a task and learning to undo the actions it has done [21]. In this way it learns policies that avoid irreversible states and aims to minimize the need for human intervention. There have been several imitation learning methods more related to the problem setting that aim to steer the agent towards the expert behavior [22, 23]. In *soft Q imitation learning*, the learning agent is incentivized to take actions that lead back to states in given expert demonstrations [24]. Similarly to RECO, they define a sparse reward function that provides reward for taking demonstrated actions from demonstrated states, training the agent to favor behavior close to the expert. They do not, however, isolate the planning problem of returning the agent to known states. Doing this in conjunction with state abstraction to facilitate simulation of the recovery problem from a fixed data set is the novel contribution to existing work.

5.4. CONCLUSION

Safety-critical or otherwise costly real-world applications are unlikely to allow automated decision-making algorithms to freely take random actions online. Encoding existing knowledge can make automated methods more applicable in such scenarios. RECO does this by applying state abstraction to expert trajectories to learn a recovery policy that steers an agent from states outside of the given data back to states where it can

confidently execute its imitation policy. It was shown in experiments that RECO can drastically improve the performance of a behavior cloning agent in tabular domains, requiring far fewer calls to a human supervisor than a naïve behavior clone and achieving near-optimal performance on par with the baseline. It was also demonstrated how a RECO agent can be implemented in continuous domains to recover from states where a supervised-learned-based imitation policy would otherwise fail. This method is especially equipped for scenarios where there are un-modelled effects in the environment that force the agent into a state where its imitation policy is poorly trained or defined. In future work, it would be fruitful to investigate more efficient distance measures in continuous problems, including measures that take into account the uncertainty in the imitation model. This novel approach will hopefully inspire tangential research into more generalized problem descriptions.

REFERENCES

[1] C. T. Ponnambalam, F. A. Oliehoek, and M. T. J. Spaan, *Abstraction-Guided Policy Recovery from Expert Demonstrations,* in *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 31 (2021) pp. 560–568.

[2] M. Bain and C. Sammut, *A Framework for Behavioural Cloning,* in *Machine Intelligence*, Vol. 15 (Oxford University Press, GBR, 1999) p. 103–129.

[3] S. Ross and D. Bagnell, *Efficient Reductions for Imitation Learning,* in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Vol. 9 (JMLR Workshop and Conference Proceedings, 2010) pp. 661–668.

[4] C. Boutilier, R. Dearden, and M. Goldszmidt, *Stochastic Dynamic Programming with Factored Representations,* Artificial Intelligence **121**, 49 (2000).

[5] T. G. Dietterich, *Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition,* Journal of Artificial Intelligence Research **13**, 227–303 (2000).

[6] Mausam and A. Kolobov, *Planning with Markov Decision Processes: An AI Perspective*, Synthesis Lectures on Artificial Intelligence and Machine Learning (Morgan & Claypool Publishers, 2012).

[7] M. E. Taylor, B. Kulis, and F. Sha, *Metric Learning for Reinforcement Learning Agents,* in *The 10th International Conference on Autonomous Agents and Multiagent Systems* (2011) pp. 777–784.

[8] J. García and F. Fernández, *Safe Exploration of State and Action Spaces in Reinforcement Learning,* J. Artif. Int. Res. **45**, 515–564 (2012).

[9] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, *Unifying Count-Based Exploration and Intrinsic Motivation,* in *Advances in Neural Information Processing Systems*, Vol. 29, edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc., 2016).

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms,* arXiv (2017), arXiv:1707.06347, 1707.06347 [cs.LG] .

[11] T. D. Simão and M. T. J. Spaan, *Safe Policy Improvement with Baseline Bootstrapping in Factored Environments,* in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (AAAI Press, Honolulu, USA, 2019) pp. 4967–4974.

[12] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable Baselines 2.10.0,* https://github.com/hill-a/stable-baselines (2018).

[13] S. Fujimoto, D. Meger, and D. Precup, *Off-Policy Deep Reinforcement Learning without Exploration,* in *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 (PMLR, Long Beach, California, USA, 2019) pp. 2052–2062.

[14] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, *Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction,* in *Advances in Neural Information Processing Systems*, Vol. 32, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019).

[15] A. Kumar, A. Zhou, G. Tucker, and S. Levine, *Conservative Q-Learning for Offline Reinforcement Learning,* in *Advances in Neural Information Processing Systems*, Vol. 33, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin (Curran Associates, Inc., 2020) pp. 1179–1191.

[16] C. Richter and N. Roy, *Safe Visual Navigation via Deep Learning and Novelty Detection,* (Robotics: Science and Systems Foundation, 2017).

[17] S. Ainsworth, M. Barnes, and S. Srinivasa, *Mo'States Mo'Problems: Emergency Stop Mechanisms from Observation,* in *Advances in Neural Information Processing Systems*, Vol. 32, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019).

[18] M. Laskey, S. Staszak, W. Y. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg, *SHIV: Reducing Supervisor Burden in DAgger using Support Vectors for Efficient Learning from Demonstrations in High Dimensional State Spaces,* in *2016 IEEE International Conference on Robotics and Automation* (2016) pp. 462–469.

[19] J. García and F. Fernández, *Probabilistic Policy Reuse for Safe Reinforcement Learning,* ACM Transactions on Autonomous and Adaptive Systems **13** (2019).

[20] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, *Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones,* arXiv (2020), arXiv:2010.15920.

[21] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, *Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning,* in *International Conference on Learning Representations* (2018).

[22] T. Hester, M. Vecerík, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. P. Agapiou, J. Z. Leibo, and A. Gruslys, *Deep Q-Learning from Demonstrations,* in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (AAAI Press, Honolulu, USA, 2019) pp. 4967–4974.

[23] N. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, N. Heess, and M. Riedmiller, *Keep Doing What Worked: Behavior Modelling Priors for Offline Reinforcement Learning,* in *International Conference on Learning Representations* (2020).

[24] S. Reddy, A. D. Dragan, and S. Levine, *{SQIL}: Imitation Learning via Reinforcement Learning with Sparse Rewards,* in *International Conference on Learning Representations* (2020).

6

CONCLUSION

ஆற்றின் வருந்தா வருத்தம் பலர்நின்று
போற்றினும் பொத்துப் படும்.
- திருக்குறள் 468

Hard work, the wrong way, will be futile
even if the world applauds.
- Tirukkuṟaḷ 468

The emergence (and re-emergence) of reinforcement learning is one of the most promising advances in artificial intelligence, bringing forth agents that learn, through only their own experience, how to perform tasks optimally. The potential is great, and with increasing data and computation power, RL is being posed as the solution to many important societal and technological problems. In healthcare, it could be used to develop effective treatment plans [1], detect lung cancer and offer clinical decision support [2]. In education, RL could be used to personalize lesson plans [3] or teaching strategies [4]. RL could improve industrial control algorithms and lead to better train scheduling [5], reservoir management [6], and power and energy systems operations [7].

The reality is that the landscape of applied RL is quite different from the exciting potential described above. There are working examples of RL algorithms being used today in financial trading by IBM [8], to determine discount pricing in Walmart [9] and to suggest content to Netflix subscribers [10]. However, in arguably more impactful areas such as healthcare, education or industrial control, reinforcement learning is not penetrating real-world systems. Significant road blocks keep it from being widely applied in practice, particularly in safety-critical or high-risk fields. These are varied and many, but speak mainly to the challenges mentioned in the introduction of this thesis [11]. In the time this thesis was under review, the introduction of ChatGPT, a natural language-processing chat bot trained in part with reinforcement learning, has promised immense disruption and brought AI to the forefront of mainstream conversation. While inarguably impressive and a feat of technology, the discourse often credits ChatGPT as

being more intelligent than it can be made out to be. Noam Chomsky recently wrote that humans are not "statistical engine[s] ... extrapolating ... the most probable answer to a scientific question", but are efficient machines that seek to create explanations [12]. This work suggests researchers take a few steps back from massive data-hungry models and continue to draw inspiration from the fundamental principles of human intelligence lying in the gap between RL and Real-Life.

6.1. THESIS CONTRIBUTIONS

This thesis centers around a belief that reinforcement learning without human-interpretable abstraction will continue to be susceptible to several pitfalls no matter how far advancements are made in computation power or black-box prediction methods.

Deep learning methods, which have several powerful properties, are not perfect. The sheer amount of data needed in order to train new deep reinforcement learning methods can make them an impractical, infeasible or even irresponsible approach (when one accounts for ethical and environmental considerations). Current deep reinforcement learning advances are celebrated for moving further and further away from the use of expert knowledge, but the price is that trust and safety assurances can be lost. Whether or not humans are purposefully included in the loop, the decisions made by an RL algorithm are ultimately developed *for* humans and must be ultimately accepted *by* humans. Another important point is that humans are required in order to define the state space of an environment either by building a simulator or by making decisions about the sensing capabilities of the automated system. In this way, humans are already imposing their ideas of the correct model or information needed in order to act optimally. The ideas proposed in this thesis only allow human knowledge to be incorporated even more, offering safe ways to empower the reinforcement learning agent with abstractions that it can learn to use effectively.

Modularity can be used in conjunction with abstraction to make model-free learning less goal-dependent and more flexible and transferable. It enables the use of several state abstractions in tandem, making more explicit on which features a decision is made. By using expert knowledge to enable modularity and abstraction, reinforcement learning can be made more transferable, more efficient and more transparent.

6.1.1. TRANSFER

Reinforcement learning is inflexible to changing environments. To maintain theoretical guarantees, it assumes the problem is stationary. In practice, even very small pixel-level changes have caused working deep RL agents to fail [13]. Taking a modular approach to learning imparts an element of robustness to this issue. When changes in the environment affect only one module, which might represent a part of the state space, a subtask, or both, ideally only that module needs to be retrained. This was demonstrated in the transfer learning capabilities of Parallel Learning in Chapter 3 and in the PRESTO method (Chapter 4).

6.1.2. EFFICIENCY

The aspect of efficiency has been mentioned several times in this thesis, most often referring to the number of samples required to learn high-performing policies. In RECO (Chapter 3), this can mean the difference between learning a good policy and a bad one, as the size of the given data set is fixed. In online methods such as Parallel Learning and PRESTO (Chapters 3 and 4), sample efficiency of the approach may come at the expense of increased computation in small problems. However, in larger problems, the benefits of abstraction and modularity can be greater and reach a point where, compared to approaching the problem in a naïve way, the amount of computation required to learn a good policy is less in addition to the number of samples required.

Finally, modularity enables tranferring of knowledge between AI systems, which facilitates reusable parts that eliminate the need for retraining, potentially having a much larger effect on efficiency in terms of both computation and amount of data.

6.1.3. TRANSPARENCY

One of the biggest problems facing many machine learning solutions today is that of transparency. Neural networks are extremely susceptible to reinforcing biases inherent in the data or the data collection policy (e.g., in healthcare [14]). Model-free reinforcement learning, even when applied without function approximation, chooses a policy that maximizes a learned value function, which, especially in complex and long-horizon problems is not interpretable. It can quickly become impossible to glean why a particular decision is chosen as optimal, and due to which facets of the problem. Abstraction simplifies the problem, but also the solution. Particularly when a task is broken down and decisions are chosen according to one of several abstractions, the problem space upon which a certain action was chosen is smaller. This can provide interesting or crucial information about factors that were and were not used to make a certain decision.

6.1.4. RESEARCH QUESTIONS

A summary of how the work in this thesis attempts to answer the research questions presented in the introduction is provided below:

- Is there value to abstraction mappings that are intuitive and human-definable, even when they form poor abstractions of the MDP?
  - In Chapter 3, the approach takes potentially several human-defined mappings as input and makes no assumptions of their correctness. The agent learns to use what it learns in each abstract space to solve the main task effectively, using the long-term rewards as the metric on which to judge each abstract policy. When the human-defined mappings have merit, it will learn faster (with fewer interactions). When they do not, the agent will still recover the optimal ground truth policy. The method in Chapter 5 uses abstractions that are again not assumed to hold for the MDP, but rather for a sub-problem that can be used to return the agent to a state where it can confidently execute its learned policy. This is found to be very effective in maintaining high performance in offline problems where the given expert data has poor coverage of the problem space.

- Is it worth the additional computation requirements that learning in several parallel processes incurs?
  - Every approach contained in this thesis employs some kind of parallel learning process. In the method in Chapter 3, it is found that if the number of parallel problems initiated is strictly less than the number of actions, then for large problems the gains made can be so great that the additional computation is dwarfed by the saved computation. In Chapter 4, one learning process is simplified by the decomposition into two simpler problems. This means the potential additional overhead is problem-dependent, but not guaranteed, and there may even be lower computational demands. Finally, in Chapter 5, the addition of the recovery problem allows the agent to reach a level of performance that would otherwise be impossible with the data given, requiring more data to be collected. Collecting data can be a very expensive and timely procedure, and the additional computational demand would likely be welcome as an alternative. Finally, one of the biggest motivations for these parallel learning approaches is modularity. The long-term computational savings of modular learning are very difficult to quantify, as each module can potentially be used in several problems, resulting in even more computational savings down the line.

- How can modular approaches contribute to the goals of lifelong RL, such as robustness, transfer and learning from offline data?
  - Modular learning allows the agent to be flexible to changing environments, evident most obviously in the approach of Chapter 4, where both the policy and the state-predictor can be transfered or retrained. This is also relevant to robustness, as when assumptions made about the environment are incorrect, the agent can adapt only the relevant part of its learning. Again in Chapter 4, the policy considers the agent's uncertainty over the state its in, meaning if the problem has been modelled incorrectly, some of this can be accounted for. The concepts of robustness and transfer are related, in that problems can be parameterized according to a range of potential models they may fall into, forming a family of potential MDPs. The MDPs in the family can be treated as separate problems that share some modules that can be reused between them, reducing the amount of necessary learning to solve them all. Finally, there is and remains a big gap in literature regarding the use of explicit abstraction for offline RL. In Chapter 5, one method that uses abstraction to make intelligent use of a limited data set is provided and has great potential when expert data is given.

6.2. WHERE TO NEXT?

This thesis is not in any means considered the end of the journey, but rather should act as an invitation and motivation for development into reinforcement learning methods that explicitly put abstraction and modularity at their core. There are several directions which this work should consider in its next steps.

6.2.1. DEEP RL

The methods in this thesis are largely targeted toward discrete MDPs with some attempts made to demonstrate how they could be extended to continuous problems. There is a justification for first considering discrete problems (which, it should be noted, does not mean the problems are small); devising abstractions for high-dimensional and continuous problems may not actually be possible for humans and thus the benefits are much more likely in tabular, and particularly factored, domains. Especially when abstractions are defined as sub-sets of factored state variables, a nice property follows where the set of possible abstraction definitions is only dependent on the number of state variables and not the size of the state space. This means that problems large in size but with relatively few state variables can see great improvements from methods such as QLiA (Chapter 3) and RECO (Chapter 5). In the PRESTO approach (Chapter 4), this depends on how much history is needed and how difficult the prediction task is overall.

Methods that have shown incredible progress in large, high-dimensional problems have often relied on implicit abstraction in the form of deep neural networks. The techniques shown here are not touted as a replacement for neural networks as abstractors, but rather show that it is possible to devise modular methods that give a more explicit understanding as to which parts of the observation are used in their decisions. Such approaches can compliment the impressive abilities of neural networks to compress huge and complex states (such as images) into implicitly meaningful features. Not only could abstraction-guided modular approaches boost the power of neural networks by removing unnecessary information when possible (thus reducing the data required for training and avoiding the tendency to overfit), they could also bring more control over what information is used for decisions, exposing or limiting inherent biases that may be present in the data.

Further, modular approaches may be a necessary next step to enable RL agents to learn from given information or even from each other. Lake *et al.* suggested that "more structure and inductive biases could be built into the networks or learned from previous experience with related tasks, leading to more human-like patterns of learning and development. Networks may learn to effectively search for and discover new mental models or intuitive theories, and these improved models will, in turn, enable subsequent learning, allowing systems that learn-to-learn – using previous knowledge to make richer inferences from very small amounts of training data." [15]. Neural networks are built on the idea of interconnected nodes; these large complex networks of dependencies go precisely against any sense of modularity. Only with intention can methods be developed that have such a property.

6.2.2. MODEL-BASED RL

This thesis presents methods that are all model-free approaches (with the exception of PRESTO which combines elements of both model-free and model-based learning). In model-free RL, the general aim is to estimate the value of taking certain actions, and then plan according to this estimation. While they form some of the most impressive examples of RL, model-free approaches also require much more data (in principle) than model-based approaches. It was previously noted that preserving the value function is crucial to maintaining optimality of abstract MDPs when solving with traditional meth-

ods [16]. It was also noted that devising abstractions that preserve the value of actions can be unintuitive, particularly in hierarchically-structured tasks with a global reward function, where the value of an action is dependent on all the following subtasks. This was one motivation for creating methods that made use of policy-preserving-abstractions, which are easier to devise.

However, there is another way to preserve optimality in an abstract MDP without preserving the value function, which is to preserve the model. The angle of inserting abstractions in model-based methods was not explored, but has potential. What is particularly promising is that checking whether an abstraction is suitable should not require learning a complex value function. However, in practice, model-based methods can be difficult to implement, and crucially the best model-preserving abstraction at a single state (according to prediction accuracy) is not necessarily the best for reward optimization overall.

Research in neuroscience has supported the idea that humans employ multiple different types of decision-making processes, some model-based and other model-free, in parallel. It has been further theorized that they trade-off the use of these processes based on uncertainty [17]. Uncertainty as a metric was explored in both Chapter 4 and 5, in the former both classification probability and ensemble deviation were used as a representation of uncertainty, and in the latter a simple Euclidean distance measure. Further development could be made in using such measures to trade off the use of abstraction when the ground truth is too uncertain.

6.2.3. LIFELONG RL

Finally, this work is seen as contributing to the goals of Lifelong RL. In the Lifelong RL paradigm, an agent continuously receives tasks from a distribution of possible MDPs, and must learn a policy or family of policies that takes this into account. This is challenging for several reasons, such as issues of non-stationarity and effective transfer learning. A modular abstraction-guided approach would allow an RL agent to train only the parts of its knowledge that differ between tasks, and to be efficient in learning the parts that are shared. Applying state abstractions for Lifelong RL has been explored in existing literature [18], but there is much more to be learned from using abstraction in the more loose and flexible terms which are introduced in this work.

REFERENCES

[1] S. Saria, *Individualized Sepsis Treatment using Reinforcement Learning,* Nature Medicine **24**, 1641 (2018).

[2] A. Coronato, M. Naeem, G. De Pietro, and G. Paragliola, *Reinforcement Learning for Intelligent Healthcare Applications: A Survey,* Artificial Intelligence in Medicine **109**, 101964 (2020).

[3] S. Sarkar and M. Huber, *Personalized Learning Path Generation in E-Learning Systems using Reinforcement Learning and Generative Adversarial Networks,* in *IEEE International Conference on Systems, Man, and Cybernetics* (2021) pp. 92–99.

[4] A. Iglesias, P. Martínez, R. Aler, and F. Fernández, *Learning Teaching Strategies in*

an Adaptive and Intelligent Educational System through Reinforcement Learning, Applied Intelligence **31**, 89 (2009).

[5] D. Šemrov, R. Marsetič, M. Žura, L. Todorovski, and A. Srdic, *Reinforcement Learning Approach for Train Rescheduling on a Single-Track Railway,* Transportation Research Part B: Methodological **86**, 250 (2016).

[6] M. Hooshyar, S. J. Mousavi, M. Mahootchi, and K. Ponnambalam, *Aggregation–Decomposition-Based Multi-Agent Reinforcement Learning for Multi-Reservoir Operations Optimization,* Water **12** (2020).

[7] D. Cao, W. Hu, J. Zhao, G. Zhang, B. Zhang, Z. Liu, Z. Chen, and F. Blaabjerg, *Reinforcement Learning and Its Applications in Modern Power and Energy Systems: A Review,* Journal of Modern Power Systems and Clean Energy **8**, 1029 (2020).

[8] A. Srinivasan, *Reinforcement Learning: The Business Use Case, Part 2,* (2021).

[9] Y. Chen, P. Mehrotra, N. K. S. Samala, K. Ahmadi, V. Jivane, L. Pang, M. Shrivasta, N. Lyman, and S. Pleiman, *A Multiobjective Optimization for Clearance in Walmart Brick-and-Mortar Stores,* INFORMS Journal on Applied Analytics **51**, 76 (2021).

[10] J. Basilico, *Netflix Explains Recommendations and Personalization - Web Presentation,* (2021).

[11] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, *Challenges of Real-World Reinforcement Learning: Definitions, Benchmarks and Analysis,* Machine Learning **110**, 2419 (2021).

[12] N. Chomsky, *The False Promise of ChatGPT,* The New York Times (2023).

[13] S. Gamrian and Y. Goldberg, *Transfer Learning for Related Reinforcement Learning Tasks via Image-to-Image Translation,* in *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97, edited by K. Chaudhuri and R. Salakhutdinov (PMLR, 2019) pp. 2063–2072.

[14] N. Norori, Q. Hu, F. M. Aellen, F. D. Faraci, and A. Tzovara, *Addressing Bias in Big Data and AI for Health Care: A Call for Open Science,* Patterns **2**, 100347 (2021).

[15] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, *Building Machines that Learn and Think like People,* Behavioral and Brain Sciences **40** (2017).

[16] L. Li, T. J. Walsh, and M. L. Littman, *Towards a Unified Theory of State Abstraction for MDPs,* in *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics* (2006) pp. 531–539.

[17] N. D. Daw, Y. Niv, and P. Dayan, *Uncertainty-Based Competition Between Prefrontal and Dorsolateral Striatal Systems for Behavioral Control,* Nature Neuroscience **8**, 1704 (2005).

[18] D. Abel, D. Arumugam, L. Lehnert, and M. Littman, *State Abstractions for Lifelong Reinforcement Learning,* in *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, edited by J. Dy and A. Krause (PMLR, 2018) pp. 10–19.

ACKNOWLEDGEMENTS

The process of doing this research was as much about the people around me as it was the work that made its way into this dissertation. I certainly would not be writing this if I had been left on my own the last 5 years. No part of this process was without support, and I am full of gratitude for those who offered it.

At the top of the list is my direct supervisor. Matthijs, you took a chance on me that changed my life's course in a way I had never envisioned for myself. Regardless of my doubts and insecurities, you have been unwaveringly supportive. With all this academic encouragement, you also placed importance on life outside of the office, and my experience has been all the better because of this. It is because of who you are that I made it this far, and that I genuinely had a wonderful time getting here.

To Frans, I am grateful that you joined my supervision team. You have been nothing but encouraging and supportive, even when offering strong critiques. These welcomed discussions have improved my research and understanding and given me confidence in my ideas. I am proud to have been one of your students and am better for it.

To the many members of the Algorithmics/Interactive Intelligence groups that I have had the luck to share the office with and to call friends: Natalia Romero, Lei He, Koos van der Linden, Erwin Walraven, Greg Neustroev, Qisong Yang, Anna Stawska, Rolf Starre, Miguel Suau, Elena Congeduti, Aleksander Czechowski, Moritz Zanger, Eghonghon Eigbe, and any others I've failed to mention. You have all taught me, inspired me, and/or believed in me at some crucial time and for that I am very thankful. An extra thank you goes to Max Weltevrede for translating the summary of this dissertation into Dutch.

I must single out a great friend, colleague and mentor, Thiago Dias Simão. Thiago, it is with utmost sincerity that I say I could not have come this far without you. Before, during and after a global pandemic, you were a constant in my academic and personal life that I needed more than I can put into words. I am extremely grateful for all of your patient help, advice and friendship. I know that I will never be able to thank you enough without embarrassing you into never speaking to me again.

To the other close friends I have made during the PhD: Yannick Hopf, Can Umut Ileri, Mariane Urias, Taraneh Younesian, Victor Jarzagaray, Hannah Backes, Sowmya Kumar, Aarthi Sundaram, Matin Nabavi Niaki and Sarah Gebhard. You have been pillars of my experience and life here, and made a new place feel like home.

Speaking of home, I am lucky to have more than one family in The Netherlands, who all made a tough transition possible. Mariam Ben Meftah, Iljaas Dhonre and the kids Dienn and Sammi, in those early days you were the only thing keeping me from running back to my parents by welcoming me into your life so seamlessly. Marjan Wijn and Marcel Gründemann, Christine Lommerse and Peter van Dolen, your support (and food!) has helped me feel a sense of belonging and security. Judy Swann and Hans van Bemmelen, I will forever be grateful that (in addition to giving us food!) you were willing to take in our dog while we were away. To those who have met our dog, you know how good

a friend one has to be to do so. And to Mirjam Post and Arnold Heemink, words simply cannot describe what you two have done for me. I am only one of many that have felt your generosity, and I have seen the great gratitude and admiration you deservedly receive from others as lucky as I have been. I have the same and so much more for you. Thank you for making me part of your very loving family.

And now to address my family back in Canada. Thank you to the Freemans for letting me take your son with me and for keeping us a wonderful place to go home to whenever we need to. To my sister Maria-Saroja, it is funny that we somehow grew a little closer while being far apart, but it is one of the things that reminds me that life can keep getting better. Thank you for taking the time to visit us even though our house is very cold. I love you immensely; I think you are incredibly talented and I learn so much from you. To my sister Kumary Chiquinquira, you taught me what it means to have values, to stay true to those values, and to never accept the status quo. That is a lesson that makes me proud to be who I am, and continues to give me something to strive for. I am in awe of your mind and your strength, and I love you. And to my brother Ildemaro Naveen, in these last few years I have seen you grow in wonderful ways; you have overcome challenges that I cannot even imagine facing. I love you for every bit of who you are, and I know that I will continue to be impressed by new surprising sides of you.

To my dad: Appa, I have no idea where you get your spirited energy from and I wish I could say that I got the same from you, but I know that I will forever be playing catch-up. I am so lucky to have a father that is enthusiastic about learning anything and everything, and so willing to change, to better himself, and to listen.

நீங்கள் இட்ட அடித்தளத்திலிருந்து உயர்ந்து நிற்கிறேன். அத் தொடக்கத்தின் வலிமையை பார்த்து இப்போது வியந்து போகிறேன்.

To my mom: Mamá, in the end, you are the only person who ever gave me a convincing reason to do this. Because of you I know purpose now more than ever, which is an enormous gift to receive. Me enseñas cada día cómo luce la fuerza, a medida que crezco aprecio más y más lo que haces y has hecho por ti y por la familia. Estoy asombrada de tu espíritu, valor y fuerza de voluntad.

To my partner: Carmel, you went on this journey with me without any hesitation. I would not have done it without you enthusiastically by my side and I am so grateful you gave that to me. Through all my troubled days, you have been the one unfortunate to bear my distress, and perhaps the only person in the world with enough patience to do so. I hope you know how sincerely thankful I am for all of your support. I know it couldn't have been easy. I am so much better because of who you are, and I hope to provide something resembling the same to you.

To the dog: Matisse, I know you are trying your best. Thank you for teaching me patience and the gift of learning to love a wild beast that may not love me back for a very long time. You have brought a very special joy into our family.

And in summary, I am fortunate to have a large circle of people, in my extended family and beyond, who demonstrate generosity, wisdom, and love. To those who are not mentioned here, it is not because you are not in my thoughts but simply due to exhaustion caused by the realization that I have so many wonderful people in my life. Thank you all for giving me an example of what I hope to be for others.

CURRICULUM VITÆ

Canmanie Teresa PONNAMBALAM

08-08-1992 Born in Kitchener, Canada.

EDUCATION

2011 - 2016 Bachelor of Applied Science in Systems Design Engineering
Unversity of Waterloo

2016 - 2018 Master of Applied Science in Mechanical and Industrial Engineering
University of Toronto
Thesis: Effects of searching for street parking on driver behaviour, physiology, and visual attention allocation: An on-road study
Supervisor: Prof. dr. ir. B. Donmez

2018 - 2023 PhD in Computer Science
Technische Universiteit Delft
Thesis: Abstraction-Guided Modular Reinforcement Learning
Promotors: Dr. M. T. J. Spaan & Dr. F. A. Oliehoek

AWARDS

2016 University of Toronto Fellowship

2015 Norman Esch Enterprise Co-op Award

List of Publications

Related to the PhD research

6. Kamran, D., Yang, Q., Simão T. D., **Ponnambalam, C. T.**, Fischer, J., Spaan M. T. J., Lauer, M. *A Modern Perspective on Safe Automated Driving for Different Traffic Dynamics using Constrained Reinforcement Learning.* Proceedings of the IEEE International Conference on Intelligent Transportation Systems (2022).

5. **Ponnambalam, C. T.**, Kamran, D., Simão T. D., Oliehoek, F. A., Spaan, M. T. J. *Back to the Future: Solving Hidden Parameter MDPs with Hindsight.* Adaptive Learning Agents Workshop at the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2022).

4. Smit J., **Ponnambalam, C. T.**, Spaan, M. T. J., Oliehoek, F. A. *PEBL: Pessimistic Ensembles for Offline Deep Reinforcement Learning.* Workshop on Robust and Reliable Autonomy in the Wild at the 30th International Joint Conference of Artificial Intelligence (IJCAI) (2021).

3. **Ponnambalam, C. T.**, Oliehoek, F. A., Spaan, M. T. J. *Abstraction-Guided Policy Recovery from Expert Demonstrations.* Thirty-First International Conference on Automated Planning and Scheduling (ICAPS) (2021).

2. **Ponnambalam, C. T.**, Oliehoek, F. A., Spaan, M. T. J. *Abstraction-Guided Policy Recovery from Expert Demonstrations - Workshop Version.* Offline Reinforcement Learning Workshop at the Thirty-fourth Annual Conference on Neural Information Processing Systems (NeurIPS) (2021).

1. Neustroev, G., **Ponnambalam, C. T.**, de Weerdt, M. M., Spaan, M. T. J. *Interval Q-Learning: Balancing Deep and Wide Exploration.* Adaptive and Learning Agents (ALA) Workshop at the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) (2020).

Not related to the PhD research

- **Ponnambalam, C. T.**, Donmez, B. *Searching for street parking: effects on driver vehicle control, workload, physiology, and glances.* Frontiers in Psychology 11, 2618 - 2631 (2020).

- **Ponnambalam, C. T.**, Cheng, R., Donmez, B. *Effects of Searching for Street Parking on Driver Behaviour and Physiology: Results From an On-Road Instrumented Vehicle Study.* In Proceedings of the Human Factors and Ergonomics Society Annual Meeting (Vol. 62, No. 1, pp. 1404-1408) (2018).

- Kaya, N. E., Ayas, S., **Ponnambalam, C. T.**, Donmez, B. *Visual attention failures during turns at intersections: An on-road study.* In Proceedings of the 28th Canadian Association of Road Safety Professionals Conference (2018).

- Giang, W. C., **Ponnambalam, C. T.**, He, X., Donmez, B. *Medical dispatch decision support for transfer time estimation: Individual operator differences in system use.* In Proceedings of the International Symposium on Human Factors and Ergonomics in Health Care (Vol. 7, No. 1, pp. 38-43) (2018).