THE UNIVERSITY *of* EDINBURGH

Policy Space Abstraction for a Lifelong Learning Agent

Majd Hawasly

Doctor of Philosophy

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

2014

Abstract

This thesis is concerned with *policy space abstractions* that concisely encode alternative ways of making decisions; dealing with discovery, learning, adaptation and use of these abstractions. This work is motivated by the problem faced by autonomous agents that operate within a domain for long periods of time, hence having to learn to solve many different task instances that share some structural attributes. An example of such a domain is an autonomous robot in a dynamic domestic environment. Such environments raise the need for transfer of knowledge, so as to eliminate the need for long learning trials after deployment.

Typically, these tasks would be modelled as sequential decision making problems, including path optimisation for navigation tasks, or Markov Decision Process models for more general tasks. Learning within such models often takes the form of online learning or reinforcement learning. However, handling issues such as knowledge transfer and multiple task instances requires notions of structure and hierarchy, and that raises several questions that form the topic of this thesis – (a) can an agent acquire such hierarchies in policies in an online, incremental manner, (b) can we devise mathematically rigorous ways to abstract policies based on qualitative attributes, (c) when it is inconvenient to employ prolonged trial and error learning, can we devise alternate algorithmic methods for decision making in a lifelong setting?

The first contribution of this thesis is an algorithmic method for incrementally acquiring hierarchical policies. Working with the framework of *options* - temporally extended actions - in reinforcement learning, we present a method for discovering persistent subtasks that define useful options for a particular domain. Our algorithm builds on a probabilistic mixture model in state space to define a generalised and persistent form of 'bottlenecks', and suggests suitable policy fragments to make options. In order to continuously update this hierarchy, we devise an incremental process which runs in the background and takes care of proposing and forgetting options. We evaluate this framework in simulated worlds, including the RoboCup 2D simulation league domain.

The second contribution of this thesis is in defining abstractions in terms of equivalence classes of trajectories. Utilising recently developed techniques from computational topology, in particular the concept of persistent homology, we show that a library of feasible trajectories could be retracted to representative paths that may be sufficient for reasoning about plans at the abstract level. We present a complete framework, starting

from a novel construction of a simplicial complex that describes higher-order connectivity properties of a spatial domain, to methods for computing the homology of this complex at varying resolutions. The resulting abstractions are motion primitives that may be used as *topological options*, contributing a novel criterion for option discovery. This is validated by experiments in simulated 2D robot navigation, and in manipulation using a physical robot platform.

Finally, we develop techniques for solving a family of related, but different, problem instances through *policy reuse* of a finite policy library acquired over the agent's lifetime. This represents an alternative approach when traditional methods such as hierarchical reinforcement learning are not computationally feasible. We abstract the policy space using a non-parametric model of performance of policies in multiple task instances, so that decision making is posed as a Bayesian choice regarding what to reuse. This is one approach to transfer learning that is motivated by the needs of practical long-lived systems. We show the merits of such Bayesian policy reuse in simulated real-time interactive systems, including online personalisation and surveillance.

Acknowledgements

None can deny that which You bestow,
and none can bestow that which You hold back;
Of no avail is greatness or luck without You.

First of all, I would like to express my sincere thanks to my PhD supervisor, Dr. Subramanian Ramamoorthy, for all his help and support throughout my years in Edinburgh. The countless hours we spent in discussion and dialogue have shaped my understanding and my research mind.

I also thank my second supervisor Dr. Michael Rovatsos and Prof. Sethu Vijayakumar for their roles in my PhD panel, and Dr. Michail G. Lagoudakis and Dr. Michael Herrmann for their roles in the examination committee.

I would like to thank all my colleagues, collaborators and officemates. Special mention to Benji Rosman for long hours of coding, scribbling on (green) blackboards, and head scratching. Many thanks to my collaborator Dr. Florian T. Pokorny from KTH for his contribution in reorienting my scientific appetite, and to Dr. Hassan Mahmud for his input and help.

I thank my sponsors, Damascus University for giving me the chance to pursue my academic aspiration in the UK, and Edinburgh University for letting me experience the truly wonderful and buzzing research world of Informatics.

I cannot express enough gratefulness to my loving family back home, one by one, for their support, care, and devotion. My mother, without you, I would never have been what I am.

I really thank numerous dear friends who kept trying to keep in touch even when I could not do the same, and I thank my flatmate, Ziad, for helping me balance my busy life in recent months.

Finally, a big ‘Thank You!’ to everyone who somehow helped in making this happen, who motivated and inspired me to reach here.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Majd Hawasly*)

To *Shaam*.

Table of Contents

List of Figures

List of Notations

β	Option's termination condition
$\mathcal{K}$	Simplicial complex
$\mathcal{O}$	Set of Options
Δ	Probability distribution
γ	Discounting factor
$\mathcal{F}$	Observation model
$\mathcal{M}$	Family of MDPs
$\mathcal{N}$	Normal distribution
$\mathcal{R}$	Regret
$\mathcal{T}$	Space of types
E	Expectation
H	Entropy
P	Probability
Π	Policy library
π	Policy
Σ	Observation signal space
$\mathcal{A}$	Action space
$\mathfrak{m}$	MDP, Markov Decision Process

Q	Q-function, state-action value
R	Reward function
S	State space
T	Transition function

Chapter 1

Introduction

1.1 Preface

This thesis is about learning policy space abstractions for a sequential decision making agent. An *agent* is a decision making entity that interacts with an environment to solve a problem, and a *policy* is one way of making decisions, or taking *actions*, towards the agent's goal in response to the environment. The *policy space* is the collection of different ways available to the agent to make decisions. A policy *abstraction* is a *re-description* of the policy space in a way that is easier to manage and store, or more efficient – from a sample-complexity point of view – to reason with and use. One example of a policy abstraction is a hierarchy of sub-policies. We aim in this thesis to automatically learn policy space abstractions for an agent from a bank of experience of previous, possibly different, tasks acquired by the same agent.

In general, abstractions are useful because they 'tame' some aspect of complexity in the agent's original problem, be it a time-, sample-, storage-, memory-complexity, *etc.* For this work, the grand motivation is the autonomous robots that live and work along with humans in their contexts. These include, for example, domestic service robots at home, and manufacturing robots in human-friendly industrial environments. These robots need to adapt continually to the possible spectrum of tasks in such uncontrolled settings, where pre-deployment learning might be insufficient when compared to the richness in the real world. Moreover, there exist situations where the robot does not have the privilege of extended deliberation as per task specification. Abstractions can help the robot have the edge to handle sample-complexity in this case, like many other realistic planning and learning scenarios.

The benefits of having policy hierarchies to combat complexity have been realised and exploited in works in different fields (*e.g.* from behavioural robotics (Brooks, 1991) and reinforcement learning (Mahadevan and Connell, 1992; Digney, 1998), to autonomous cars (Urmson et al., 2007) and distributed power generation (Dörfler et al., 2014)). Hierarchical policies are like *distributed* controllers, and thus more resilient to change and perturbation. Moreover, decomposing the policy space into components facilitate reuse and knowledge transfer between similar tasks, allowing better performance, faster.

Nonetheless, no consensus exists on the best way to autonomously *learn* policy hierarchies. In many works, the hierarchy components are learnt, but the hierarchy itself is designed. For example, in Hierarchical Reinforcement Learning (Barto and Mahadevan, 2003), the agent is provided with a collection of objectives for the subtasks

that make the hierarchy, and the agent's role is to learn policies to achieve these subtasks. By specifying the hierarchy, the designer injects domain knowledge into the agent, and constrains the degrees of freedom in the search space in order to make the learning process tractable. However, the flexibility that our motivation requires, in adapting to a growing spectrum of tasks, suggests that the agent should both learn the hierarchy and its components.

In this thesis, we discuss methods for *learning policy hierarchies from experience in related tasks by a learning agent*. This is related to concepts like learning from demonstration and transfer learning, but it departs from them in subtle aspects. For example, we learn from demonstrated solutions to previous tasks solved in the domain, but the demonstrations may involve multiple interacting skills rather than a unique skill. Then, the gained experience can only be transferred to a new task once the unknown distribution that generated it is approximated from the previous interactions.

1.2 Setting

We consider agents in a *lifelong, real-time learning setting* in a *dynamic domain*. The agent is presented with a sequence of different, but related, tasks in the domain.

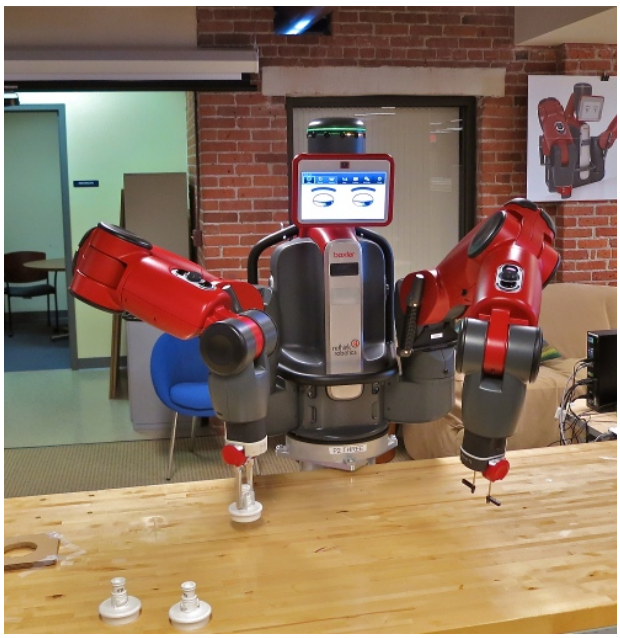
- A *lifelong* agent differs from a conventional (single-task) agent in that it needs to build on *experience* acquired in previous tasks to handle future tasks. Experience could refer to the behaviours that the agent learnt previously, or to the understanding it gained about the domain and its dynamics or the tasks and their variability.

- *Real-time* learning refers to the requirement to produce decisions and behaviours in relatively short time. This requirement calls for learning approaches that parallelise learning and acting, and which operate in batch mode or in the background, and rules out the ones that require a prolonged learning phase for individual tasks.

- *Dynamic domains* are ones that imply *variability* featured in tasks. The scale of this variability, however, does not deny the domain some inherent properties (or *structure*) that define it. Dynamic domains are manifested by *families* of qualitatively-related tasks, rather than single tasks.

- A sequence of tasks in such a dynamic domain is a collection of samples from the task family, presented to the agent one by one in sequence, each for a limited

(a) Domestic robot in a home environment. (*Photo from (Chen et al., 2013).)*



(b) Industrial robot in a human-friendly work environment. (*Photo courtesy of Steve Jurvetson, used under a Creative Commons license.)*

Figure 1.1: Robots in human environments.

time.

Example domains and tasks One grand challenge that has the aforementioned properties is the case of domestic and industrial robots that share the human environment in order to perform certain services or tasks or to offer help (Fig. 1.1). The robot needs to build on acquired experience in the environment to tackle new tasks more quickly, and needs to interact and respond promptly to the human needs and requests. The environment, on the other hand, can change in subtle ways which affect the behaviour of the robot, but do not change the nature of the domain. As it goes, the robot is continually presented with new tasks from an unknown family of possible chores.

An example that we will be considering later in this thesis is robotic soccer (Fig. 1.2 and Appendix B). This is a multi-robot strategic task that has the elements of many interesting real-world problems. In robotic soccer, a team of robots needs to score in the opponent's goal, while protecting its own. This task requires real-time response and does not allow lengthy deliberation, making previous experience a valuable resource to exploit. Each match is a member of a family of instances in the soccer domain, with each opponent team exhibiting a different style of play that affects the game dynamics.

However, the structure in the domain is maintained by the rules of the game.

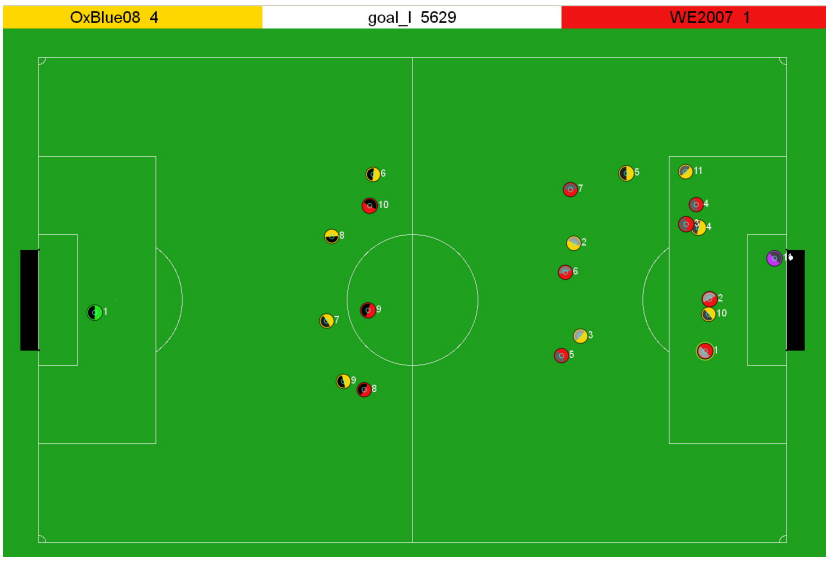


Figure 1.2: RoboCup 2D Simulation League.

We argue that many real-world problems share some or all of their features and that many state-of-the-art learning techniques are not designed to handle such requirements.

1.3 Problem Statement

We consider an agent that has experienced and solved a collection of tasks in some dynamic domain. The input for our work is a library of *experiences* that has been accumulated in these tasks. This can be the collection of policies learnt in the previous tasks, a collection of trajectories that the robot used, or a description of the task instances that the robot experienced.

The thesis tackles the following questions:

- how can the agent acquire hierarchical abstractions of the domain in an online and incremental manner starting from a library of learnt policies,
- how to devise mathematically rigorous ways to abstract a library of trajectories, produced for some task, based on their qualitative similarity, and
- what alternate algorithmic methods for decision making can be taken when it is inconvenient to employ prolonged trial and error learning, in a lifelong setting.

1.3.1 Rationale

Typically, sequential decision making problems are attacked using forms of reinforcement (Sutton and Barto, 1998) and online learning (Auer et al., 2002). However, these methods either do not scale to the complexity of the considered domains (as, for example, with bandit algorithms) or tend to require prolonged periods of interaction to solve a particular instance (as, for instance, with plain Q-learning).

An alternative to using plain learning is the use of transfer learning (Taylor and Stone, 2009). In reinforcement learning transfer, some of the knowledge gained in learning one task can bootstrap the learning of another task that shares some element of similarity. Transfer most often happens between two tasks, one called the source, and the other called the target. Nonetheless, transfer does not come without caveats. *Negative transfer* is the problem that arises from using an unsuitable source for some target, and which most often leads to delayed convergence rather than improvement in the learning performance. Hence, choosing what to transfer is essential for the success of the process and that requires knowledge about both tasks.

For transfer to work in our situation, the qualities of the target, being the new unknown task, should be first *modelled*, then a *suitable* source, or sources, should be identified.

One approach to model target tasks is through the hierarchical structure that is shared between their policies. This structure can be captured using one of the techniques of Hierarchical Reinforcement Learning (HRL) (Barto and Mahadevan, 2003), such as options (Sutton et al., 1999). Previously-seen tasks can help in defining the structure as well as populate the hierarchy components with policies. Then, transfer to a target task becomes a process of learning of a hierarchical policy.

However, using hierarchy in a lifelong learning setting should accommodate the gradually-experienced variability in the domain, and thus requires that the hierarchical structure is flexible and adaptive. This is in contrast to the case typically considered in HRL where the structure is *a priori* fixed. Thus, the difference between this setting and typical approaches of hierarchy learning and option discovery is that it should not only consider a single task, but a family of tasks. This requires a process of discovery and learning to be employed to *create*, *update* and *repair* the hierarchical structure as the interaction evolves, as well as *learn* its individual components. Furthermore, novel notions of *persistent subtasks* that are useful across the task family or which capture qualitative importance in the domain should be used.

A more direct way to model target tasks and facilitate transfer is by using manifold learning. This process is straightforward if the tasks are parametrisable (*e.g.* (Silva et al., 2012)), but that limits its applicability to only simple tasks. Indirect ways of mapping between abstractions of tasks and policies are needed in many scenarios and domains that do not have this property (*e.g.* when task parameters are not observable, such as in adversarial interaction). With this, transfer becomes an issue of 'looking up' a policy from a 'dictionary' of policies, using the abstracted description of the task, with no need for further learning online.

1.3.2 Properties of the solution

Our approach is to *autonomously* develop, and *incrementally* maintain, abstractions of the policy space, using *suitable representations* of the accumulated *experience* in previous tasks, as an operational model for the domain that is suitable for *transfer*. This model generates low sample-complexity responses to new instances.

Then, the requirements for a solution is that it provides

1. a rich description of the policy space of the tasks of the domain. The description is built unsupervised from proper representations of previous experience,
2. it should be adaptable when new experience is acquired, and
3. a procedure to generate a solution for a new task using a small amount of samples and the policy space model.

Our approach is summarised in Fig. 1.3.

1.4 Contributions

We propose three approaches to define policy space abstractions:

- We build on the hierarchical reinforcement learning concept of *options* and we extract skills from a bank of experience. The resulting hierarchical model is transferable to new tasks, and learning with options is employed for planning.

  We extract options from the transitions of all previous tasks using batch reinforcement learning to get **Offline Options** (Hawasly and Ramamoorthy, 2012). To discover persistent options, we fit a mixture of Gaussians to a collection of

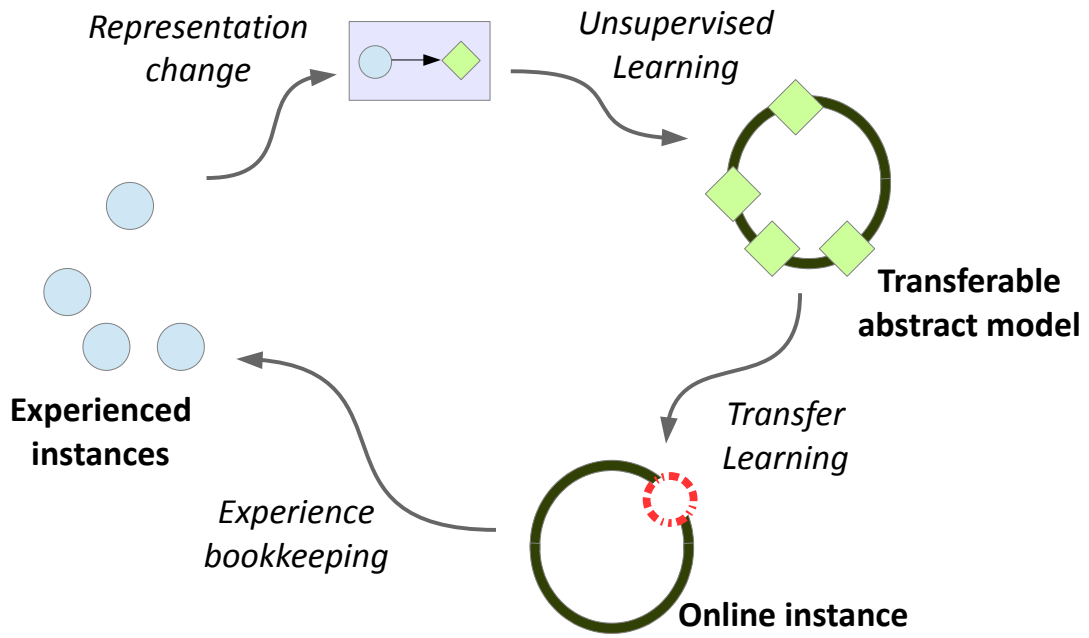


Figure 1.3: A depiction of the approach. A transferable model is learnt using unsupervised learning from a collection of previous experiences, after a suitable representation change. Then, a technique of transfer uses the model to generate a solution for a new instance. The resulting policy is integrated into the experience and the model.

trajectories extracted form previous tasks. The components in the mixture identify important regions in the state space (dubbed *generalised bottlenecks*). We acquire options constrained to these regions using a process of policy reuse. We employ a continual process that proposes and removes options as the interaction evolves, giving the framework of **ILPSS** (Incremental Learning of Policy Space Structure) (Hawasly and Ramamoorthy, 2013a,b).

- We employ tools from **computational topology** to a collection of trajectories of some task to identify the different ways to achieve the task under different conditions of the dynamics/costs in the domain. This uses a special *simplicial complex* construction that captures higher-order information of connectivity on different levels of persistence. The construction maps each trajectory to a cycle in the complex and identifies the classes of the cycles using *persistent homology* theory (Pokorny et al., 2014).

  We tie this approach back to the option discovery framework and define a novel

criterion for option discovery based on qualitative uniqueness. Then, we use the classes of trajectories extracted from a collection of tasks to define the smallest set of **Topological Options**.

- For tasks with unobservable parametrisation, we build a non-parametric model of policies using observables that are correlated with policy performance in the set of experienced tasks. We propose **Bayesian Policy Reuse** to choose a response to a new task from the set of input policies. This is achieved through contrasting the values of these observables to the values of the experienced tasks. This is especially important when learning in the new task is infeasible.

1.4.1 List of publications

- M. Hawasly and S. Ramamoorthy; 'Task Variability in Autonomous Robots: Offline Learning for Online Performance,' International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS), 2012.
- M. Hawasly and S. Ramamoorthy; 'Lifelong learning of structure in the space of policies,' AAAI Spring Symposium Series on Lifelong Machine Learning, 2013.
- M. Hawasly and S. Ramamoorthy; 'Lifelong transfer learning with an option hierarchy,' IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013.
- F.T. Pokorny and M. Hawasly and S. Ramamoorthy; 'Multiscale topological trajectory classification with persistent homology,' Robotics: Science and Systems (RSS), 2014.

1.4.2 Under review

- B. Rosman, and M. Hawasly and S. Ramamoorthy; 'Bayesian Policy Reuse,' Machine Learning Journal (MLJ), 2014.[1]

1.4.3 In preparation

- F.T. Pokorny and M. Hawasly and S. Ramamoorthy; 'Multiscale topological trajectory classification with persistent homology,' International Journal of Robotics Research (IJRR), 2015.

[1] The first two are jointly first authors of this paper.

1.5 Layout of the Thesis

The rest of the thesis is organised as follows. Chapter 2 summarises the related work and lays out the assumptions and foundations of the work in the thesis. Then, Chapter 3 discusses the use of hierarchy and option discovery for policy abstraction, presenting the framework of ILPSS. After that, the use of computational topology in building abstractions from experienced trajectories and topological options are described in Chapter 4. Next, the scheme of Bayesian policy reuse is detailed in Chapter 5. Finally, we end with discussing future work and concluding remarks in Chapter 6.

Chapter 2

Preliminaries and Related Work

2.1 Tasks

The tasks we are interested in for the setting of this thesis have the following attributes:

1. *Single-agent*. That does not prevent the environment from having other agents, but the tasks of these agents would not be tightly-coupled, as for example is the case in strategic interactions (games). We consider the domain of RoboCup, due to factors of task complexity and limited perception and communication, in this realm.

2. *Sequential*. A task requires the agent to make a series of decisions.

3. *Goal-oriented*. The agent has to achieve some goal to finish the task successfully.

4. *Bounded experience*. A task offers only limited experience, afterwards a new task starts. The agent thus is encouraged to respond quickly to solve a task.

5. *Inherent variability*. There is uncertainty regarding attributes of the task specification that needs to be accommodated in decision making. This variability does not change the objective of the task, but nonetheless changes how the agent should tackle it. An example of this is when the environment contains other agents with 'loosely-coupled' objectives. The behaviour of these agents may entail a change in the dynamics or the cost in the domain. This is in contrast to 'tightly-coupled' interactions (games) where the behaviour of other agents (players) redefines the task objective for the agent.

6. *Continual*. The agent, possibly after an initial training phase, is continually experiencing an infinite sequence of episodic task instances in the domain.

2.1.1 Example tasks

We describe here the tasks for the two motivational examples mentioned previously in light of these attributes.

Human-Robot Interaction (HRI) Consider a robot within a human environment performing a specific kind of short tasks (*e.g.* domestic chores in a residential place) (attribute 2). The task of the robot transpires in time and it aims to achieve some target (*e.g.* setting the dinner table) in some bounded time (attributes 3, 4). There might be active elements in the environment which would affect the task of the robot, though

not in an adversarial way (*e.g.* a pet) (attribute 1, 5). A new task may be different, as a feature of the richness of the domain, or it may be very similar to a previous task, so that building on what is learnt at one time is beneficial for future (attribute 6).

RoboCup Consider a soccer-playing robot in a match. At the highest strategic level, this task is best described as an adversarial interaction between two opponent teams and a cooperative interaction between many team mates. However, many researches (*e.g.*, (Stone, 2000)) find that, as the size and the time scale of the full multi-agent interaction does not immediately reflect on the short-term task of a single autonomous agent, it is more fruitful to treat this as a single-agent task for a specific assigned role in a changing environment, rather than the game-theoretic representation (attribute 1, 2, 3). The variability in this domain is represented by the opponent team (attribute 5) which is only faced for the length of the match (attribute 4). The experience gained with some opponents should then be exploited in future encounters (attribute 6).

2.1.2 Non-examples

On the other hand, the following important example tasks violate some of the required attributes, and thus they are *not* explicit targets of this work:

- Solving a tightly-coupled multiagent interaction (*e.g.*, security games (Tambe et al., 2013)).
- Solving a one-shot online prediction problem (*e.g.*, a multi-armed bandit (Auer et al., 2002)).
- Learning in a non-episodic, infinite-horizon task.
- Learning an optimal policy of a single Markov Decision Process task.

2.2 Markov Decision Processes

We model the individual task instances in the environment as discrete-time Markov Decision Processes (MDPs) (Bellman, 1957; Sutton and Barto, 1998), over discrete or continuous state spaces.

2.2.1 Discrete-state MDPs

A discrete-time, episodic, discrete state, Markov Decision Process (MDP) is a tuple (S, A, T, R, γ):

- S is a finite state space.
- A is a finite action space.
- $T : S \times A \times S \to [0,1]$ is the (stationary) dynamics of the world, where $T(s,a,s')$ is the probability $P(s'|s,a)$ of reaching state s' from state s by taking action a, with
$$\sum_{s' \in S} T(s,a,s') = 1, \ \forall a \in A, \forall s \in S.$$
- $R : S \times A \times S \to \mathbb{R}$ is the (stationary) reward process, where $R(s,a,s')$ is the immediate reward achieved by taking action a at state s and upon reaching s'. R encodes the goal of the task.
- $\gamma \in [0,1]$ is the discounting factor.

The process evolves in discrete time steps $t = 0, 1, 2, \ldots, N-1$ where N is the length of one episode. The agent is expected to make a decision on which action to apply every time step.

The aim of an agent acting in an MDP is to select actions to maximise the expected total discounted reward in the future, $E\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{N-1} r_{t+N-1} | s, a\}$ for the future rewards $r_t = R(s_t, a_t, s_{t+1})$. γ defines how much the agent values the future gains compared to the immediate gains. Setting γ to 1 maximises the total *undiscounted* reward, which is only relevant to episodic tasks. Thus, the agent searches for a *policy*. A Markov policy for an MDP is a (stochastic) mapping from states to actions, $\pi : S \times A \to [0,1]$. That is, $\pi(s,a) = P(a|s,\pi)$ is the probability of taking action a at state s under the policy π.

The expected cumulative reward of state s and action a under a policy π is stored in the *state-action value function* (or the Q*-function*),

$$Q^\pi(s,a) = E^\pi\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s, a\},$$

assuming that π is followed in all the time steps after a.

The *optimal policy* π^* is a policy that maximises the expected cumulative reward, and the optimal state-action value function is the value function of π^*, $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ for all pairs (s,a).

2.2.2 Continuous-state MDPs

A discrete-time, episodic, continuous-state MDP m is similar to its discrete counterpart with the exception that $S \subset \mathbb{R}^k$ is a bounded, infinite state space with k dimensions. Consequently, T and R become functions of infinite domains.

In this case, the Q-function is a continuous function over $S \times A$, and hence cannot be represented using a simple tabular representation as in the discrete case. One way to learn these functions is through incremental function approximation which attempts to find the appropriate parameters of a function class in order to fit the observed Q values.

2.2.3 Family of MDPs

To model the variability in the agent's environment, we represent each possible task with an MDP and we assume that the complete set of all the possible realisations of the tasks can be represented by a family $\mathcal{M}$ of qualitatively-related MDPs. This family comprises MDPs that share the state-action space $S \times A$, but the dynamics $T_i : S \times A \times S \to [0,1]$ and the rewards $R_i : S \times A \times S \to \mathbb{R}$ may be different for each member of the family $m_i \in \mathcal{M}$.

For the dynamics, one way to think of the different transition functions in an environment is to consider them coming from some *uncertainty set* (*e.g.*, (Nilim and Ghaoui, 2005)) or from a generative process over that set, $T_i \sim \Delta[T]$, $\forall i$, (*e.g.*, (Strens, 2000)). If $\Delta[T]$ is known, the family $\mathcal{M}$ can be replaced with one Partially-observable Markov Decision Process (POMDP) with a state space that is a cross-product of S and the support of $\Delta[T]$.[1]

Similarly, the variability in the reward can be modelled using a set of feasible reward function in order to find robust solutions (*e.g.*, (Regan and Boutilier, 2010)).

In this thesis we will always assume that the variability model is fixed, but *unknown* to the agent.

2.3 Reinforcement Learning

Reinforcement Learning (RL) (Kaelbling et al., 1996; Sutton and Barto, 1998) is the branch of machine learning concerned with learning policies from trial and error. This could be approached in an incremental fashion: by estimating the goodness of some

[1] More accurately, if $\Delta[T]$ is known, the family $\mathcal{M}$ can be represented with a MOMDP – a Mixed Observability Markov Decision Process (Ong et al., 2010), as S is assumed observable.

proposed policy (*prediction*), then extracting a better policy from that estimate to act (*control*), and so on.

Seen as an optimisation problem, RL is different from other forms of machine learning in that the target function it is trying to learn (*e.g.*, the value of states or state-action pairs under the optimal policy, or the goodness of policies) is always latent and unobservable, but a signal (*reward/ punishment*) correlated to that function is available. The task of the agent is to autonomously decode that correlation to learn a behavioural policy. RL techniques can be used to solve MDP processes when the dynamics and the reward processes are not known.

Model-based reinforcement learning methods create an estimated model of the environment (the dynamics and rewards), then apply some 'planning' procedure on the estimated model to give a policy. A well-known model-based RL method is R-MAX (Brafman and Tennenholtz, 2003). *Model-free* methods, on the other hand, work directly on estimating the goodness of policies without first having to estimate any model of the environment. One of the most used method in this latter setting is Q-learning (Watkins and Dayan, 1992).

2.3.1 Batch Reinforcement Learning

Usually, sampling in reinforcement learning is sequential, and in many cases, expensive. One approach that mitigates this is batch reinforcement learning, where a large collection of *offline* experiences is used at once to learn behavioural policies (Boyan, 1999; Lange et al., 2012), avoiding the cost of the interactive sampling from the online process. This allows exploitation of historical data and domain knowledge that was collected previously, on the cost of heavier computation and memory requirements.

A simple extension of interactive Q-Learning to batch mode is Fitted Q-Iteration method (Ernst et al., 2005).

2.4 Hierarchical Reinforcement Learning

Reinforcement learning suffers from the *curse of dimensionality*, which refers to the exponential increase in the size of the learning problem with the increase of the size of the description of the domain. To fight that, approaches of temporal abstraction are proposed. Rather than learning policies at the granularity of the action space, hierarchical policies, which take temporally-extended actions that achieve some kind of

useful behaviour in the task, can reduce the severity of the curse of dimensionality in structured tasks. Hierarchical Reinforcement Learning (HRL) (Barto and Mahadevan, 2003) creates coarse abstractions in the action space, either by employing temporally-abstracted actions that achieve subgoals, or by reorganising the policy space into configurations of behaviours that make learning in new tasks more tractable.

2.4.1 Semi-Markov Decision Process

A *semi-Markov decision process* (SMDP) emerges when dealing with temporally extended actions in MDPs in hierarchical reinforcement learning. In a discrete-time SMDP, action execution is allowed to extend over multiple time steps (Howard, 1971). This makes reasoning about execution durations explicit in learning and planning:

$$Q^{\pi}(s,o) = E^{\pi}\{r_{\tilde{t}} + \gamma^{\tau_1} r_{\tilde{t}+1} + \gamma^{\tau_1+\tau_2} r_{\tilde{t}+2} + \ldots | s,o\},$$

where o is a temporally extended action, $\tilde{t}, \tilde{t}+1, \ldots$ are the times when decisions are taken rather than the normal time steps t, and $\tau_1, \tau_2, \ldots$ are the execution durations for the corresponding temporally-extended actions under π. Many of the planning and learning algorithms of MDPs work naturally in such SMDPs (*e.g.*, (Parr, 1998)).

2.4.2 Options

The framework of *options* (Sutton et al., 1999) is one approach of HRL that extend the action repertoire with generic temporally-extended actions.

An option o is the three-tuple $\langle I, \pi, \beta \rangle$. $I \subseteq S$ is a subset of the state space called the initiation set in which the option can be invoked. $\pi : S \times \mathcal{O} \rightarrow [0,1]$ is a (Markov or semi-Markov) policy over the augmented set of actions (containing the actions A and any other temporally-extended options that do not invoke o). This policy is followed when the option is invoked. Finally, $\beta : S \rightarrow [0,1]$ is a probability distribution over the state space that defines the termination condition of the option.

The option framework's strength is that it allows primitive actions to be considered as (trivial) 1-step options for planning purposes. For a set of options $\mathcal{O}$, an option switching policy $\pi^{\mathcal{O}} : S \times \mathcal{O} \rightarrow [0,1]$ is a stochastic map from states to options/ primitive actions. It is known that sequencing a set of (Markov or semi-Markov) options from $\mathcal{O}$ defined over an MDP gives a well-defined SMDP, and allows planning and learning of $\pi^{\mathcal{O}}$ via similar approaches to planning and learning in MDPs (Sutton et al., 1999).

Option Interruption

Normally, an option selected by a switching policy $\pi^{\mathcal{O}}$ at some state continues to run until its termination condition β is satisfied. The *interruption* of an option o at some state s_t refers to the process of switching control from o to a new option before its natural termination (as defined by $\beta^o(s_t)$) if the value of the option o at the interruption state s_t is inferior to the expected value of the policy $\pi^{\mathcal{O}}$ at s_t:

$$Q^{\pi^{\mathcal{O}}}(s_t, o) < \sum_{q \in \mathcal{O}} \pi^{\mathcal{O}}(s_t, q) Q^{\pi^{\mathcal{O}}}(s_t, q).$$

Option interruption is a kind of non-hierarchical execution of hierarchical reinforcement learning policies and it allows for performance improvement over the SMDP policy.[2]

2.4.3 Other HRL approaches

Hierarchies of Abstract Machines

Parr introduced the Hierarchies of Abstract Machines (HAMs) framework (Parr, 1998; Parr and Russell, 1998) as a way to structure the policy space for an MDP by constraining it to what can be realised through a collection of 'programs' or finite-state machines (FSMs). These machines are capable of producing actions for the task MDP when called by the hierarchy controller, they can realise non-deterministic behaviour, they can call other machines, and they have their termination conditions that return control to the calling entity in the end.

HAMs when combined with an MDP results in an SMDP whose decision points occur when the member machines take decisions, and whose realisable policies are constrained to hierarchical executions of the programs of the machines.

MAXQ

Dietterich proposed MAXQ value function decomposition as another method to constrain the realisable policies of an MDP through a hierarchy of controllers, each designed for a specific subgoal (Dietterich, 1999). Organised in a graph, the controllers are constrained in terms of which of the other controllers they can invoke. Each controller has

[2] From here on, we will be using π to refer both to primitive action policies and option switching policies.

its own policy and pseudo-reward function, and it takes into account the *execution stack* of the calling controllers when making decisions.

In MAXQ, each controller defines an SMDP over its children controllers. The hierarchy of SMDPs decompose the value function of the task MDP into a sum of *continuation functions* of the various controllers, each corresponding to the expected value *after* a specific decision is executed.

2.5 Survey of Option Discovery Methods

Options are classically designed for a specific task in order to reduce learning and planning complexity. However, many approaches have been proposed to automatically discover a set of options for a task. Next, we give a brief overview over the different option discovery methodologies.

2.5.1 Trajectory-based methods

These methods rely on trajectories generated from optimal policies as input to the hierarchy discovery process. Options can be generated through trajectory matching to find common behaviour or by employing supervised learning techniques on trajectory sets.

PolicyBlocks

(Pickett and Barto, 2002) For a set of tasks that differ only in the reward process, the agent is given a set of optimal policies. The set of options is produced through a matching between the optimal policies and the metric used is the compactness of the description. Note that this process assumes no learning. Some operators are proposed to merge options, extracting shared structure, and to subtract one from another. The shared option is added to the set, and the original options are replaced with smaller policies resulting from subtracting the new option from the original options.

Motifs

(Zang et al., 2009) search for options using trajectories generated from the optimal policy. The trajectory is treated as a sequence of actions and common motifs are identified as candidate options. The goal of an option is defined to be the state that precedes a change in a state abstraction sufficient to describe the motif, and the initiation

set comprises all the states that can reach that goal. A policy is then learnt to achieve the option's goal from the initiation set using value iteration.

2.5.2 Frequency-based methods

In these methods, the subgoals are discovered through statistics maintained for individual states. (Digney, 1998) introduces two features to identify a useful macro-action. The first is the *square of the gradient of the reinforcement signal* at a state, which is an inter-task feature. Changes in this metric denote the importance of state. The second is the *frequency of visiting* a state, which requires many tasks and hence, is an intra-task feature.

Visitation frequency is the most used state statistic in frequency-based methods for option discovery. A simple frequency-based method is proposed in (Stolle and Precup, 2002) where optimal trajectories for a set of optimal policies of random tasks in a fixed environment are used to calculate the frequency of visiting each state. The state that has the highest frequency is chosen to be a goal of a new option, whose domain is chosen by interpolation between the states that have higher-than-average occurrence in the optimal trajectories which reach the subgoal state. A slightly different approach in (Chen et al., 2007) optimises locally-computed derivative of the visitation frequency to select subgoals.

Diverse Density

In (McGovern and Barto, 2001), the question of discovering subgoals in MDPs is posed as a multiple-instance supervised learning problem. Instead of relying on similarity between policies, this method searches the observation space of the agent to discover useful subgoals, utilising the intuition that bottleneck regions occur more frequently in the observations of the agent in successful trajectories, but not in failed ones. For that, a methodology similar to Diverse Density (Maron and Lozano-Pérez, 1998) of multiple-instance supervised learning can be applied using the trajectories as bags of instances and the bottlenecks as concepts. The states that have the highest value of diverse density are elected as candidate subgoals and later options. The domain of an option is then defined to be the union of a subset of the states that the agent visited before reaching the new subgoal in successful trajectories. A drawback is that the candidate subgoal should not have appeared in a negative trajectory. Another issue is that filtering states near the start state and the goal state is needed to discover meaningful

subgoals.

(Kretchmar et al., 2003) build on the previous method's intuitions while mitigating its shortcomings. Their method, called FD, uses only successful, acyclic trajectories, and computes for each state the product of its appearing frequency and a temporal distance to undesirable subgoal states, such as the task goal or the initial states. The states with the highest value is selected as a subgoal, and a policy is learnt using policy replay.

Relative Novelty

Relative novelty (Şimşek and Barto, 2004) is one metric for bottleneck states, here called *access states* for that they link different regions in the state space. In a given trajectory, the ratio of the average visitation frequency of a certain state and what follows it in the trajectory to the average of states that came before it, up to a horizon which is a parameter of the method, relates to the relative novelty of that state. Any state will see different values of relative novelty each time it is visited, but the distribution of these values in states that are 'interesting' would be different to the distribution in other, 'normal' states, in that it would have heavier tails indicating a higher tendency of larger values. Then, these distributions can be used as an input to a classifier that outputs candidate subgoals. Finally, options are constructed for these subgoals.

A similar intuition appeared earlier in (Goel and Huber, 2003) using the *count metric* which is the probability to reach a specific state in an arbitrarily-starting acyclic trajectory generated by the optimal policy. The choice of subgoals depends on the ratio of the gradients of that metric before and after a candidate state in a trajectory being greater than a specific threshold. An extension to this work appears in (Asadi and Huber, 2005) where the gradient ratio is computed using Monte Carlo sampling rather than exhaustively.

2.5.3 Graph-based methods

In these methods, the model of the environment which the tasks share is constructed from experience, usually in the form of a transition graph. This graph has the states as nodes, and transitions between them as edges. The edges are usually weighted with estimates of the transition probability. Once the graph is built, graph-theoretical attributes are used to define subgoals.

Graph partitioning

Q-Cut (Menache et al., 2002) Bottleneck states happen at the borders of strongly connected areas of the state transition graph. Considering the process as a graph, with relative visitation frequency as capacities, bottlenecks can then be found using Max-flow/Min-cut methods of graph theory. This assumes a global perspective on the full process, as compared to the local, frequency-based methods discussed above. This is also a reward-free method.

Clustering Rather than relying on individual state subgoals, collections of states that are important in various stages in the achievement of the task are considered in (Mannor et al., 2004). To this end, the nodes of the transition graph are clustered in a way that maximises separation and options are learnt to navigate between these state clusters. Clustering can be performed on the topology of the graph alone or with the aid of state value information. This is useful to guide the clustering algorithm into finding clusters with homogeneous values.

Local graph partitioning Here, subgoals are the states that connect two regions in state space via one-step transitions that have low, positive probability. Rather than finding these through a global perspective of the transition graph, L-Cut (Şimşek et al., 2005) uses a local transition graph constructed only with the recent experience of the agent, represented by partial trajectories not necessarily from optimal policies. The algorithm searches for graph cuts that yield high transition probability within partitions, and small probability between partitions, and tries to locate subgoals from the states that have partition-crossing edges.

Linear time complexity While graph partitioning is considered an NP-hard problem, (Kazemitabar and Beigy, 2009) propose a linear time algorithm to extract subgoals from transition graphs. To find the states that connect, with low probability, dense regions in the space, a locally-estimated graph like in (Şimşek et al., 2005) is used to discover strongly connected components, which are state clusters in which every two states are reachable from each other. This is achieved through a depth-first search that gauges the distance from a specific state in the original graph and in its transpose (the directed graph with the original transitions reversed). The computational trick is to use the adjacency list rather than the adjacency matrix to cater for the sparseness of the transition graph. Subgoals are defined to be the borders of these components. Also, the

authors propose to find the single parameter of the method (threshold of the transition frequency for it to be considered an edge) automatically by examining the histogram of edge frequencies for local minima.

Centrality

Betweenness is a graph-theoretic centrality measure that estimates how important a node is by examining the number of shortest paths traversing it. In (Simşek and Barto, 2009), subgoals are the states that locally maximise betweenness on the state transition graph. The full knowledge of the graph is not necessarily needed, as incremental discovery is also possible through the utilisation of decision rules on statistics extracted from experienced subgraphs.

In (Rad et al., 2010) a different centrality measure, a variation of *connection graph stability* which puts more emphasis on local maxima, is used for the same purpose.

2.5.4 State factorisation methods

These methods differ from all the previous methods in that they aim to build hierarchies by exploiting the independent axes in the state space, rather than finding interesting individual, or collections of, states. Using this independence, the state space can be projected down into smaller subspaces which can be individually optimised.

HEXQ

Using the rate of change for different state features, the assumption of HEXQ algorithm (Hengst, 2002) is that it is possible to build an abstraction whose levels are described independently by distinct state features. The lowest level is described by the most-frequently changing state feature. The task of the learner is to explain the state transition using only that feature. Any states that do not comply with that description are considered *exit states*, ready to be exploited by higher layers. The discovered regions of state space along with their possible exits are considered smaller MDPs, and learning is employed to reach all the exit states from anywhere inside. Other layers in the hierarchy use these regions as abstract states and their learnt action policies as abstract actions to transition between them, forming a hierarchy of SMDPs. The resulting hierarchy has a similar structure and semantics to MAXQ[3].

[3]MAXQ hierarchies can be regarded as a special constrained case of an option hierarchy.

In (Kozlova et al., 2009), exits are also automatically discovered. But unlike HEXQ, exits here do not correspond to simple state-action pairs, but to a variable whose value changes at the exit and a collection of constraints on state features that describe the context.

2.5.5 Continuous domains

Skill Chaining

In (Konidaris and Barto, 2009), the goals of options in continuous domains are defined using *goal trigger functions* that flag the realisation of an event of interest. The initiation set can be defined via clustering of state space, using the states on the successful trajectories as input instance. After that, an indicator function of the previous option's initiation set can serve as a new target function of a new option. This process continues to create a chain, or tree, of skills.

2.5.6 Other methods

These are methods that do not fall under any of the previous categories, or have features from multiple of them.

SKILLS

The method in (Thrun and Schwartz, 1995) considers a set of tasks, in the same $S \times A$ space, that differ only in the reward process or dynamics. A *skill* is a partial policy, defined for a subdomain of $S \times A$ which helps the agent to define compact policies for a set of related tasks.

The learning algorithm ascends a measure of compactness and optimality by modifying the skill policy, domain, and usage rates. The skill policy is learnt using Q-Learning, while the domain is evolved from a single state by adding more relevant states when possible. A state is added to a skill domain if adding it will make the compactness-optimality measure at that state higher than not adding it. Usage is optimised similarly using a stochastic gradient descent process.

Macro-actions

In (McGovern, 1998), subgoals are elected for showing an increase in reward compared to previous states in reward histories for a robotic application. Because the domain is

continuous, the target of the 'macro-action' is defined as a region in the sensory space of the robot defined through clustering and chosen using statistical summaries of the clusters that reflect their visitation frequency compared to individual states. The policy to reach that region is learnt using Q-Learning.

In (Girgin et al., 2010), a *conditionally terminating sequence (CTS)* is a sequence of actions with their corresponding state regions, called continuation sets, that describe a behaviour. A *sequence tree* is a collection of CTS's with decision points. The nodes in the tree are action sequences with their continuation sets and the edges represent action choices. The idea is to build and keep adapting a sequence tree on the fly while interacting with the environment. This tree represents one big *meta*-option, due to the continuous evolution of the tree, with the decisions made using SMDP values and frequency statistics.

In (Drummond, 2002), the structure of a learnt task is extracted in the form of a graph that describes important features in the optimal value function. These features are discovered using a vision technique called the 'snake' to find smooth regions in the state space delimited by a high value gradient boundary. A policy for a new task can then be produced by composition of transformed subgraphs with their corresponding solutions.

Bisimulation metrics

(Castro and Precup, 2012) propose a method to use an optimal policy for a smaller MDP in solving a larger MDP by extracting reusable options defined over patches of 'bisimilar' states (similarity established using *bisimulation* – the quality of not being able to distinguish two systems by observing their moves, or evolution). An option is built for every source state in the source MDP, with the policy proposed by its bisimulation to the target states. The termination condition is defined to capture the boundaries of similarity to the source state, while the initiation set is defined so that the option terminates in a useful way in a limited number of steps.

2.6 Related Work: Learning Skills

In this section, we discuss some proposals from the literature that touch upon the problem of abstraction learning for a lifelong learning agent by developing subpolicies and skills.

2.6.1 Control theory approaches

In hybrid control, when no single control law can regularise a complex system, there might exist a set of local control laws that can collectively achieve the desired effect. In (Burridge et al., 1999), the policy space of a robot is decomposed into a collection of state-stabilisation feedback controllers, each with a domain of attraction that is a member of some 'lower' controller. The policies of these dynamical systems are befittingly called *funnels*. This essentially creates a graph hierarchy in the policy space and decomposes the state space into overlapping cells.

Achieving a specific goal is possible through sequential composition of funnels, back-chaining from the goal state back to the initial system state to activate the appropriate controllers. More recently, (Tedrake, 2009) propose a randomised motion planning algorithm that builds a tree of Linear Quadratic Regulator (LQR) stabilised trajectories, connecting the states of the system to a goal state, and prove that it is probabilistically complete in covering the input states.

2.6.2 Hierarchical reinforcement learning approaches

Portable options (Konidaris and Barto, 2007) define a joint abstraction for multiple qualitatively-similar tasks that differ in their state description. This abstraction, called the *agent space*, encodes only the qualities that are shared in the family, such that the related tasks look the same when projected to that abstraction. Thus, options defined in the agent space can be easily transferred to any other task that shares the same abstraction.

In (Konidaris et al., 2012a), goal-oriented 'skills' are extracted from expert demonstrations in continuous domains, then organised in *skill trees*. The demonstrations are segmented into chains of sub-trajectories, each generated by a different skill that has its own goal.

Each skill could have a different state-action abstraction and value function model, and the proposed algorithm automatically selects the suitable abstraction for the skill from an abstraction library. The boundaries of a skill are determined using backchaining (Konidaris and Barto, 2009), by trial-and-error learning of the states from which the goal can be reached, or using an approach of *change-point detection* based on a Hidden Markov Model (HMM) of possible tasks implemented using a particle filter (Konidaris et al., 2012a).

In (Mehta et al., 2008b), a dynamic Bayesian network is used to extract the causal

and temporal relationships between the actions in a single successful trajectory of the task. This results in a partitioning of that trajectory, then used to construct a MAXQ hierarchy.

2.6.3 Dynamical Motion Primitives

A *motor* (or *movement*, *motion*) *primitive* is a parametrised template of a complete motion. A Dynamical Motion Primitive (DMP) is a motor primitive that is based on a dynamical system to provide the encoding of the behaviour, usually a point attractor or a limit cycle. The parameters of the encoding can then be adapted using statistical learning to represent a larger spectrum of movements (Schaal et al., 2005, 2007).

Designing DMPs requires first identifying the equations that describe the dynamical system in order to guarantee the desired properties, *e.g.* stability. Then, the parameters can be acquired using imitation learning or reinforcement learning. The former allows finding a quick assignment of values to generate a good solution, while the latter can do the fine tuning to optimise a suitable performance criterion (Peters et al., 2003, 2005).

2.6.4 Machine learning approaches

In (Mahmud et al., 2013), a clustering technique is used to segment a set of MDPs into clusters so that a policy exhibits similar performance when applied to MDPs within the same cluster, and a *landmark MDP* is chosen to represent each cluster. Then, the set of optimal policies for these landmark MDPs represent an abstraction of the policy space that is used to drive a policy reuse algorithm.

2.7 Related Work: Learning to Generalise

In this section, we discuss some proposals from the literature that touch upon the problem of abstraction learning for a lifelong learning agent by developing generalisation for new tasks from example task instances and demonstrations.

2.7.1 Transfer Learning for RL

Transfer Learning in the context of reinforcement learning is concerned with learning to act in a set of related tasks by leveraging the experience gained in some of them (for a review, see (Taylor and Stone, 2009)). The idea is to use a set of source tasks

to accelerate learning in a novel target task different in rewards, dynamics, or even state-action space.

Some successful transfer methods rely on an explicit and observable parametrisation of the task space (*e.g.* (Mehta et al., 2008a; Silva et al., 2012)), while others assume a known distribution of variability in the tasks (*e.g.* (Perkins and Precup, 1999)), or a distribution that can be approximated from previously-seen source tasks (*e.g.* (Tanaka and Yamamura, 2003; Snel and Whiteson, 2012)). Bayesian methods can use these distributions as priors to estimate, then solve, the novel target task (*e.g.* (Sunmola and Wyatt, 2006; Wilson et al., 2007, 2012)).

A different approach is to find a reduced, ego-centric representation that has the same semantic in all tasks, in which the source and target tasks look effectively the same (Konidaris et al., 2012b), then to transfer policies on that level of description. Similarly, but under relational reinforcement learning, (Croonenborghs et al., 2008) uses a relational description of the task to transfer skills, or *relational options*, to other tasks that share the same symbols.

Policy reuse

Policy reuse is another Transfer Learning approach that deals with task families. Using a mechanism to select the most similar previously-seen task to the novel one, a policy can be used *as is* on the target task (Mahmud et al., 2013), in a model selection style. For a learning approach, the reused policies can be used to accelerate learning in the new target task, *e.g.* by biasing the exploration scheme in the new instance (Fernández and Veloso, 2006).

2.7.2 Lifelong learning

Intrinsically motivated reinforcement learning (Singh et al., 2005; Soni and Singh, 2006) defines skills that achieve unpredicted *salient* events in the domain of the agent. That is, no external objective is needed to define the skills, but they are learnt for any discovered state that seems interesting according to this internal function, usually in the form of options.

2.7.3 Learning from demonstration

Learning from demonstration (or *programming by demonstration*, *imitation learning*) is a supervised learning process that evolves policies for a robot by using positive (or

negative) examples of task achievement (Billard et al., 2008; Argall et al., 2009). The positive examples are handled as local maxima points in the policy search space, while the negative examples represent constraints on the search space. This allows faster and more direct approach for robot skill acquiring.

The two general classes of learning from demonstration methods are *trajectory encoding* where low-level mapping between perceptions and actions in simple tasks is learnt, allowing generalisation to similar tasks, and *symbolic encoding* where sequencing of primitives is rather acquired, allowing capturing higher-level knowledge of the task.

Mapping functions

A function that maps states to abstract actions can be learnt by considering the demonstrations as an input for a classification technique. Also, a function that maps states to primitive continuous actions can be learnt by regression. Afterwards, the learnt policy can be improved by experience (Bentivegna, 2004).

In the case of structured task space with known parametrisation, it is likely that the policies of the tasks live in a smooth lower-dimensional manifold (or in several charts of a piece-wise smooth manifold) in the policy space. Exploiting this, (Silva et al., 2012) propose 'parametrised skills' as a method to learn the mapping between the task parameter space and the policy parameter space in such scenarios, starting from a set of solved instances. The mapping, defined through a set of non-linear regression functions, is then used to generalise and propose solutions to new, unseen task instances.

For dynamical motion primitives, attempts have been made to map task parameters to meta-parameters of a DMP, like, *e.g.*, the end position of the motion or its duration. These meta-parameters not only fine tune the behaviour, but also define its shape. Reinforcement learning can be employed to learn the relation between task parameters and meta-parameters from a set of demonstrations and trials to generate more generic DMPs (Kober et al., 2012). Alternatively, in (Bitzer et al., 2008), a dimensionality reduction technique is first applied on the demonstrated trajectories to extract a smaller latent representation which is invariant to the redundancy in the demonstrations and which captures the essence of the task.

Model-learning approaches

Using demonstrations, the agent can learn a model of the domain dynamics using reinforcement learning techniques. To allow generalisation to unseen state-actions,

function approximation can be employed.

On the other hand, the agent has to learn the reward function for the desired task. One approach is to use the techniques of Inverse Reinforcement Learning (Abbeel and Ng, 2004) which try to estimate the parameters that define the cost function of the demonstrator, or in other terms, the weights of the different features that specify the skill reward. Then, this metric can be optimised to imitate the skill.

Symbolic plans

Demonstrations can be understood as sequences of symbols, and a symbolic representation of the task can be extracted then used in a symbolic planning framework.

2.8 Related work: Topology-based Approaches

In this section, we discuss some proposals from the literature that touch upon the problem of abstraction learning by methods that utilise or reason about the topological structure in the policy space.

Sampling-based motion planning algorithms that utilise a set of samples from the free configuration space of a robot have been quite successful in answering questions about path-connectivity in motion planning tasks. In particular, approaches based on RRTs and PRMs (LaValle and Kuffner, 2001; LaValle, 2006; Kavraki et al., 1996) have attracted unabated interest since their discovery (Lindemann and LaValle, 2005; Masehian and Sedighizadeh, 2007; Karaman and Frazzoli, 2011). These approaches typically represent the planning domain by a graph whose vertices are the sampled points and where an edge is inserted between two vertices if they are within some distance threshold. These graphs act as abstract representations of the space of paths.

Approaches which attempt to reduce the space into equivalence classes of paths include the work on (Jaillet and Simon, 2008) on path deformation roadmaps. The authors there propose a graph-based representation of the space of paths up to a class of continuous deformations.

On the other hand, *homotopy* of paths can also be used in identifying the path classes (see Appendix. A). One advantage of knowing the homotopy classes of paths in a domain, with respect to the obstacles in the configuration space, is that a motion planning algorithm can utilise efficient replanning algorithms within each such class (Brock and Khatib, 2000), deforming and optimising a given input path continuously. It is hence

advantageous to maintain a set of homotopy inequivalent classes, each of which can be optimised using gradient or variational methods.

Also, homotopy-equivalence allows reasoning about the equivalence classes directly rather than about all the realisable trajectories in the domain. In (Kuderer et al., 2014), a collection of homotopy-distinct paths between two points are maintained by a mobile robot to allow negotiating dynamic obstacles on the go. Another application of the same idea is to enable *shared-autonomy*, as in the case of a robotic wheelchair that takes into account the sparse user control input (Fig. 2.1 shows a number of different motion classes in red, and the one chosen by the user in green). Using a similar technique, a hierarchical probability distribution is learnt in (Kretzschmar et al., 2014) in a plan recognition task for a set navigating agents.

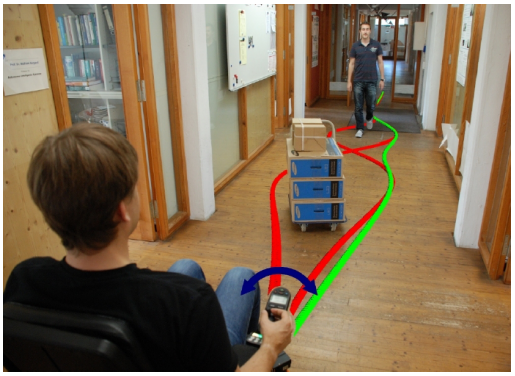


Figure 2.1: Planning with homotopy-equivalence classes for a shared autonomy wheelchair (Image from (Kuderer et al., 2014)).

On the other hand, different methods have been developed to identify equivalence classes, or to generate trajectories that fall in some specific class, using *homology*, which is a weaker version of homotopy (see Appendix. A). It's argued that in problems of robot path planning, there is almost no difference between the two concepts.

For example, in (Kim et al., 2012), path generation in 2D with homology constraints is formulated as a Mixed-Integer Quadratic program by endowing path segments with binary labels that identify their relation to the domain obstacles, then solved by an anytime planner. Using the classical residue theorem of complex analysis (Bhattacharya et al., 2010) studied an application of homology classes to 2D motion planning, in the case where the obstacles in the configuration space can be contracted into representative points. In (Bhattacharya et al., 2011), this was extended to 3D via the electromagnetism theory and Ampere's law where obstacles can be contracted into skeletons and modelled as current-carrying wires. A further generalisation in (Bhattacharya et al., 2013) to

arbitrary dimension Euclidean spaces is proposed, where the integration of differential 1-forms over cycles is shown to be sufficient to find the invariants of homology classes using de Rham cohomology theory (Fig. 2.2 shows a number of different classes of movements between two fixed points with respect to which obstacles the go through or avoid).

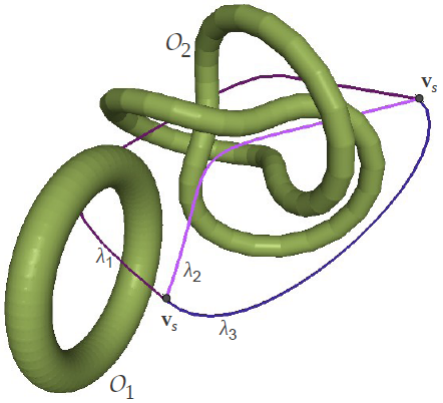


Figure 2.2: Homotopy classes in a 3-dimensional space with obstacles (Image from (Bhattacharya et al., 2013)).

2.9 Concluding Remarks

In this chapter, a quick review of the required concepts in this thesis are presented, as well as a sample from the literature that touches upon the topic of learning abstractions in policy space.

Our concept of lifelong learning meets learning from demonstration in its need to generalise from a set of examples, where the examples are the solutions for previous tasks solved in the domain. Our task can be considered as learning from demonstration by assuming the robot to be the teacher, that the demonstrations are executed by the body of the learner, and that the learning progresses interactively while new demonstrations are being produced. One difference to learning from demonstration though is that a demonstration may involve multiple interacting skills that the agent has to acquire and comprehend, rather than a single unique skill.

Intrinsically motivated reinforcement learning as a kind of developmental learning is related to our concept of lifelong learning in its autonomous search for hierarchies of generalised skills that can be applied to a wide range of problems in the agent's domain. However, it differs in that its target is only the intrinsic motivation, while a lifelong

learning agent responds to that extrinsic motivation of solving particular problems in the domain. However, these two approaches are not orthogonal and nothing prevents a lifelong learning agent form extending its knowledge of the domain by exploration and intrinsic curiosity.

Transfer learning is related to our learning framework, but it usually assumes that the target task is fully known so that a proper source task can be chosen, or a suitable mapping can be developed. In our case, the new task comes from an unknown distribution, which the agent can only approximate from previous interactions.

For mapping functions in the general case, the task space parametrisation is either too complicated to be captured, latent (or partially observable) in general or in the run time. Examples of such tasks include walking on rough terrains, *adhoc* interaction with human users, and real time surveillance and monitoring, respectively. On the other hand, policies might be too complicated to be parametrised or the discontinuities in the space might prevent fitting smooth manifolds. Examples of that include soccer playing strategies. Finally, the mapping between the two spaces might be intractable to compute, especially when the task and policy spaces are vast and the seen examples comprise only a small subset.

Dynamical motion primitives are based on parametrising fixed dynamical systems, assuming specific formalisations of skills and constraining the shape of their trajectories, as encoded by the dynamical system equations. Also, DMPs that have meta-parameters assume that meta-parameters are known and specified beforehand, and that tasks can be parametrised.

Using topology to abstract policy space is promising. One problem that the above mentioned approaches suffer from is the fact that *they require an explicit description of the obstacles in the configuration space*, *e.g.* as unions shapes, each of which is contractible to a geometrically specified point or skeleton. Typically, such information is not easily available for real robotic systems, because an explicit description of the free configuration space is either impossible or too expensive to compute.

Chapter 3

Policy Space Abstraction by Option Discovery

3.1 Introduction

We argue in this chapter that many interesting domains have key components, or *subtasks*, that are shared between the task instances, and which are *neutral to the variability in the instances*. We argue that capturing these would improve the agent's performance in a novel task instance in the domain. We seek to automatically extract a set of such subtasks from the agent's experience, then refine the discovered hierarchy with new gained knowledge.

Thus, we address in this chapter the problem of learning policy space abstraction through the options framework of Hierarchical Reinforcement Learning (HRL). We introduce our notion of *generalised bottlenecks* in Sec. 3.4.1 as a metric to define persistent options for a domain. To do this, we employ a probabilistic unsupervised learning method that automatically identifies these state bottlenecks, capturing interesting contiguous regions in state space that occur in various previously-seen instances of the task. We propose a method to extract the options by using a process of policy reuse that populates these regions with the exact same behaviours learnt in previous instances, creating pieces of skill. We integrate these concepts with a continual update procedure that keeps the abstraction refined and up-to-date as more tasks are solved in the ILPSS framework in 3.4.

Fig. 3.1 gives an overview of the approach in this chapter.

3.1.1 Contributions

The main contributions of this chapter are[1]:

- Extracting options for transfer for a collection of tasks using Batch Reinforcement Learning.
- A concept for persistent bottlenecks in a family of tasks called the generalised bottleneck.

[1]The work in this chapter has been published in:

- Hawasly, M.; Ramamoorthy, S., 'Task Variability in Autonomous Robots: Offline Learning for Online Performance,' International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS), 2012.
- Hawasly, M.; Ramamoorthy, S. 'Lifelong learning of structure in the space of policies. In Lifelong Machine Learning,' AAAI Spring Symposium Series, Machine Lifelong Learning, 2013.
- Hawasly, M.; Ramamoorthy, S., 'Lifelong transfer learning with an option hierarchy,' IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp.1341,1346, 2013.

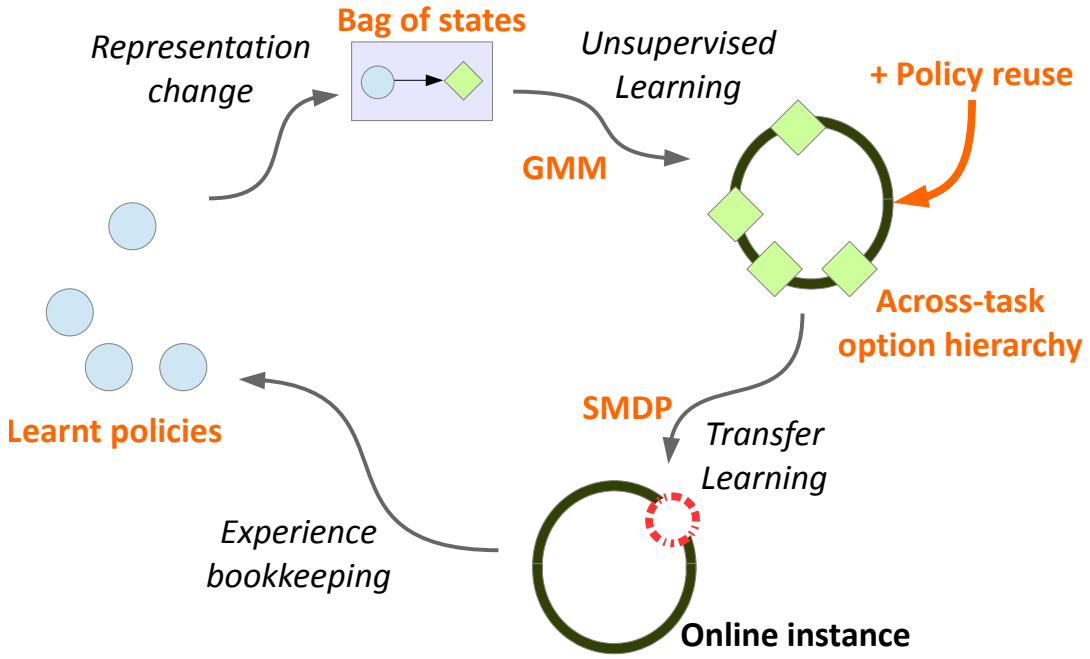


Figure 3.1: Overview of the approach in this chapter. The experience, a set of learnt policies, generates a 'bag of states' encoding. The unsupervised learning step is implemented via a mixture model learning, which structures the state space. A process of policy reuse transforms the state space model into a policy space model, creating options. Then, learning a new instance becomes an instance of SMDP learning. The learnt policy is then added to the policy collection and the process repeats.

- Extracting persistent options from a family of tasks by employing unsupervised learning technique to trajectories of successful trials, and a process of policy reuse.
- A process that manages the collection of options in a lifelong process by proposing new options and decaying old ones.

3.2 Abstraction using Option Hierarchies

3.2.1 Related work

Exploiting hierarchical policy structure for transfer purposes has seen some success. Originally, exposing hierarchical structure in policies is considered a way to counter

complexity in reinforcement learning, and a number of methodologies to organise and learn a hierarchical policy for a specific task were proposed (see (Barto and Mahadevan, 2003) for an overview of HRL methods).

One of the most flexible methodologies is the framework of options (Sutton et al., 1999), and its potential in transfer was also examined. For example, (Konidaris and Barto, 2007) introduces portable options which are abstract actions defined in a reduced state space, called the agent-space, instead of the full problem-space. The motive for that is that they can be transferred to any problem that shares that reduced representation. In (Konidaris et al., 2010), 'skills' are extracted and chained to construct skill trees from expert demonstrations. Finally, generalisation for parametrisable tasks can be achieved through the discovery of smooth low-dimensional spaces where the policies of these tasks lie (Silva et al., 2012).

To employ hierarchy into lifelong learning, the abstraction should be learnt rather than designed. One way to uncover the policy hierarchy in a task is to discover subgoals that are essential to the achievement of the task objective (see Sec. 2.5 for an overview). An important notion of this is a *bottleneck*, which is a landmark state through which successful trajectories tend to pass, while unsuccessful ones do not. Finding bottlenecks has been approached using many metrics, including state visitation frequencies (Stolle and Precup, 2002; Şimşek and Barto, 2004), and graph-theoretic measures of transition graphs, like Max-flow/Min-cut (Menache et al., 2002) and betweenness (Simşek and Barto, 2009). However, many of these methods work in discrete settings or require an upfront complete knowledge of the task. One exception is (Konidaris et al., 2010) where options are discovered automatically in a continuous space, but to solve particular tasks rather than families of them.

3.3 Offline Options

First, we introduce a simple concept of option learning that is viable when learning is not feasible at all in the online case.

3.3.1 Averaging policy

For a set of sampled MDPs $\mathcal{M}' \subseteq \mathcal{M}$, the *mean MDP* $\bar{\mathfrak{m}}$ is the process that has the same state-action space $S \times A$ as the members of $\mathcal{M}$, but has the dynamics and reward processes $\bar{T} = E_{\mathcal{M}'}[T]$, and $\bar{R} = E_{\mathcal{M}'}[R]$.

Define the *averaging policy* $\bar{\pi}$ as the optimal policy for the mean MDP $\bar{m}$. Note that the value function of this policy averages sample returns generated from the models of $\mathcal{M}'$. Depending on the variability in $\mathcal{M}$ and $\mathcal{M}'$, the performance of $\bar{\pi}$ might be quite poor, and, in general case, conservative, as it tries to choose actions that suit the full spectrum of dynamics and rewards in $\mathcal{M}'$.

3.3.2 Offline options

An offline option is a skill that targets a specific, frequently-occurring subtask in the tasks that an agent has experienced.

Definition 1 (Offline Option)**.** An offline option for a subtask is the tuple $\langle I, \pi, \beta \rangle$, with I and β specifying where the subtask is viable and where it ends, respectively, as in standard options. The policy of the option π, on the other hand, is learnt in *batch mode* across a set of previously-experienced instances in the domain for a specific pseudo-reward function that defines the goal of the subtask.

This is in contrast to the standard option learning, where options are learnt specifically for the task at hand. This is where offline options get their name from.

Because the subtask it represents is a smaller problem, we argue, the policy of an offline option will be less affected by variability compared to the averaging policy $\bar{\pi}$ of the full task.

Then, the option switching policy μ for a new task can be learnt over the SMDP induced from the set of offline options. Due to the real-time requirement, the agent might be unable to learn μ online for a new task. Instead, we would learn an averaging switching policy $\bar{\mu}$ using batch reinforcement learning from offline experience. To improve on this hierarchically-optimal policy for the mean MDP, we propose to incorporate a notion of option interruption into the process.

3.3.3 Offline interruption

The use of interruption in the options framework not only improves performance via non-hierarchical execution of hierarchical policies (Sutton et al., 1999) but also adds an element of 'reactivity' to the state of interaction. This appears to be useful for handling unknown situations. However, the interruption condition in (Sutton et al., 1999) requires knowledge of option values in the desired instance, which we do not have. We propose

a modified notion of interruption that depends only on values extracted form the offline experience.

Given $\bar{\mu}$, the offline-learnt switching policy, and its offline value function $Q^{\bar{\mu}}$, we define offline interruption.

Definition 2 (Offline interruption)**.** A running option o in the unknown MDP m may be *offline-interrupted* at state s_t, if the maximum value of o at that state in all instances under the averaging option policy, $\hat{Q}^{\bar{\mu}}(s_t, o) = \max_{m \in \mathcal{M}} Q^{m,\bar{\mu}}(s_t, o)$, is strictly less than the averaging value of selecting a new option at s_t according to the policy $\bar{\mu}$: $V^{\bar{\mu}}(s_t) = \sum_q \bar{\mu}(s_t, q) Q^{\bar{\mu}}(s_t, q)$.

The choices of $\bar{\mu}$ would be conservative and 'safe', and following $\bar{\mu}$ would be suboptimal in general. The intuition behind the definition is that when the best seen value of the currently-running option goes below that stable safety threshold upon reaching some state, it is reasonable to follow the safe choice instead.

Algorithm 1 gives a simple procedure for decision making with offline options and offline interruption.

Algorithm 1 Decision making with Offline Interruption

Require: O: option set; $\bar{\mu}$: averaging option policy; $Q^{\bar{\mu}}$: value function of $\bar{\mu}$ over O; $\hat{Q}^{\bar{\mu}}$: maximum values of the option set O under $\bar{\mu}$; s_t: current state.

$o_{run} \leftarrow \varnothing$.
for every time step t **do**
  if o_{run} is not set **then**
    $o_{run} \leftarrow \arg\max_{o \in \mathcal{O}} Q^{\bar{\mu}}(s_t, o)$.
  else
    if $\hat{Q}^{\bar{\mu}}(s_t, o_{run}) < \sum_q \bar{\mu}(s_t, q) Q^{\bar{\mu}}(s_t, q)$ **then**
      $o \sim \bar{\mu}(s_t, .)$
      $o_{run} \leftarrow o$.
    end if
  end if
  $s_t \leftarrow execute(\pi^{o_{run}}(s_t, .))$.
  if o_{run} is finished **then**
    $o_{run} \leftarrow \varnothing$.
  end if
end for

In the algorithm, $\bar{\mu}(s,.)$ is a probability distribution over options, corresponding to the option switching averaging policy, and $\pi^o(s,.)$ is a probability distribution over other options or primitive actions, corresponding to option o's policy.

3.3.4 Experiment: windy grid world

The aim of this experiment is to test offline options and offline interruption. A gridworld of 5×5 cells has an obstacle with two exits (Fig. 3.2). Wind blows immediately before the exits. It has an unknown, but fixed, direction in any instance, and there is a fixed probability throughout the episode of the wind to blow at any specific time step. If the agent was on the windy column when that happens, the wind pushes the agent one cell at a time in the direction of the wind. The goal of the agent, starting from a random cell in the leftmost column (marked with 'S'), is to pass one of the exits to the right side (marked with 'G'), moving one cell at a time with the 4 canonical actions (up, right, down, left). The agent gets -1 penalty for every action taken until the goal is reached or the episode elapsed (100 time steps). That is, the agent has an incentive to finish the task as fast as possible. Moving towards a wall does not change the location of the agent, but it will cost it the -1 penalty.

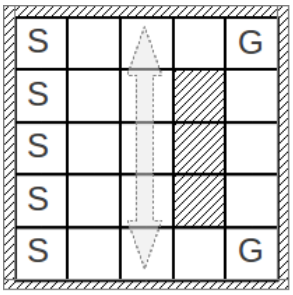


Figure 3.2: The Windy gridworld. The agent starts randomly in one of the cells marked with 'S' and is tasked with reaching any of the cells marked with 'G'. The arrow indicates the locations and possible directions of wind.

Each task instance of this MDP family is characterised with the two parameters tuple $(\mathrm{p}, \mathtt{dir})$. $\mathrm{p} \in [0,1]$ is the probability with which the wind will succeed in changing the position of the agent, while $\mathtt{dir} \in \{\mathtt{North}, \mathtt{South}\}$ is the wind direction. The agent might be pushed one cell in the direction $\mathtt{dir}$ with the probability p while in the windy cells.

The agent experiences many instances of this family, and learns in batch mode an averaging policy across all these instances. This is a policy over the primitive actions.

At the same time, we make the agent learn two options with hand-picked goals (the two 'G' states), one for reaching the goal through each of the exits. The agent learns an averaging option switching policy as well, using the offline experience and the two options in batch mode, and estimates the option values from these training instances using a Monte-Carlo procedure.

Fig. 3.3 shows 5 curves resulting from 5, 10000 uniformly-sampled task, runs. The agent is given 100 time steps in each instance, where both the flat averaging policy and the offline-interrupted option policy are evaluated. The performance criterion is the accumulated reward in the task (ranging from -100 to -5), and we report in the figure a histogram of performance difference between the two methods.

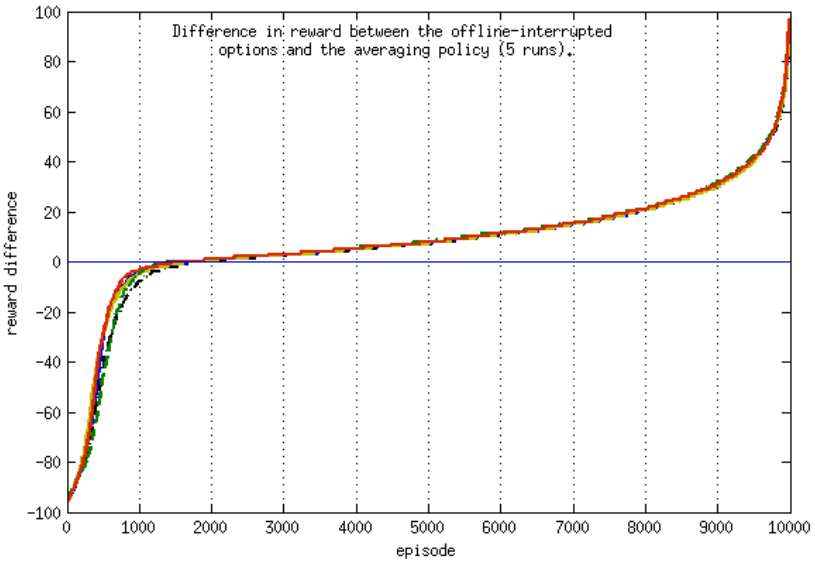


Figure 3.3: A histogram of the difference in performance between the offline-interrupted option policy and the averaging policy in 5 runs. Values above zero are tasks where the offline interruption outperforms the averaging policy.

The offline-interrupted option policy is equal or better than the flat averaging policy in almost 80% of all tasks. Also, the interruption mechanism allowed for active intervention in the control process (sensed through the change in state) in contrast to the fixed averaging policy.

3.4 ILPSS: Incremental Learning of Policy Space Structure

3.4.1 Generalised Bottlenecks

A bottleneck is a state which only successful trajectories in a task go through, so that it deserves to be treated as a subgoal. Bottlenecks classically are discrete states identified by metrics like state visitation frequencies or require complete knowledge of the domain to compute measures like centrality or betweenness, which both are unsuitable for continuous or complex domains. For example, in the soccer domain, we would anticipate that 'concepts' of game play are more useful to achieve tasks, *e.g.*, score goals, than single states.

In (Konidaris et al., 2010), continuous 'target functions' in a task are identified and posed as goals, then options are learnt to achieve them. Alternatively, we start by discovering the domains of potential skills from trajectories of previous successful trials in many tasks. Using an Expectation-Maximisation (EM) procedure, we search for a generative model for the states of the good trajectories and use the support of the components in that model in state space to define the boundaries of our options. In the soccer domain, this may mean identifying contiguous situations of play that are common in successful trials and creating skills that cover them.

The typical next stage in HRL is to learn policies to achieve the discovered subgoals. We, on the other hand, employ a process of policy reuse to populate the option domains with action policies. That is, we 'borrow' from a solved previous instance a part of its policy and use it *as is* for the new option. This makes that agent do exactly what it did in a previous instance, when facing a qualitatively similar situation.

3.4.2 Algorithm

We consider episodic, goal-oriented tasks, in which termination occurs either when the agent reaches specific *goal states* defined by $g : S \to \{0,1\}$, or when the episode elapses.

Fig. 3.4 gives a snapshot of the components and operation of ILPSS (Incremental Learning of Policy Space Structure). It comprises the following key steps:

1. Starting from a collection of $n+1$ good policies $\pi_0, \pi_1, \ldots, \pi_n$ of task instances $m_0, m_1, \ldots, m_n \in \mathcal{M}$, we sample a number of complete episode-long state traces $\tau = \{s_0, \ldots, s_{|\tau|-1}\}$ from each policy, where $s_i \in S$ are states.

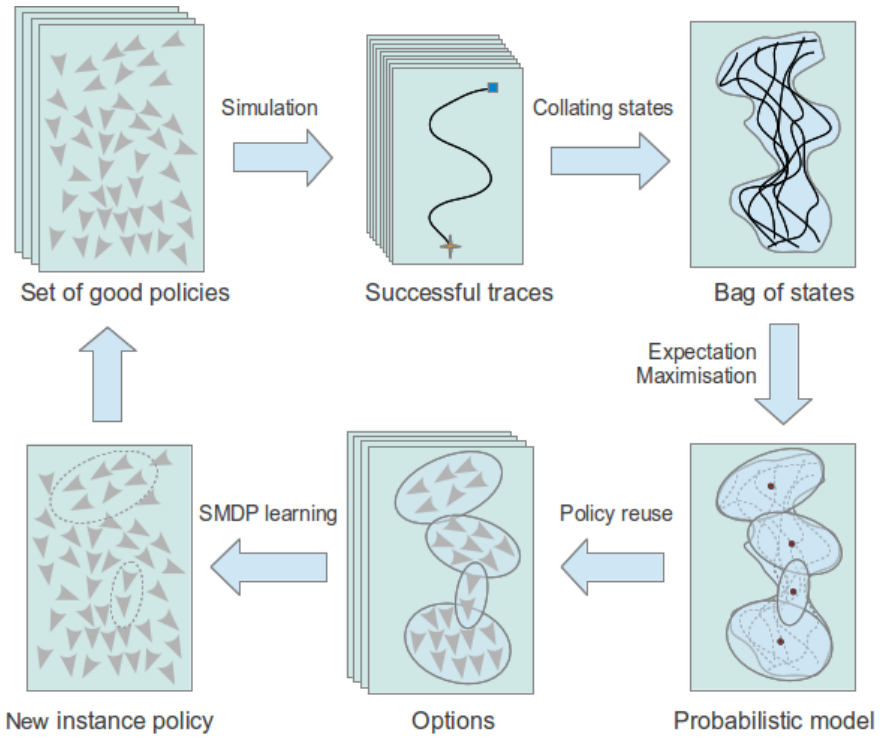


Figure 3.4: A high level overview of ILPSS. Starting from a set of policies that operate in the domain, a set of successful traces are extracted. These encode paths in the domain that have been used before and found useful. The states in the successful traces from all previous tasks are then collated into a 'bag of states'. The bag obviously excludes states that cannot be reached (*e.g.*, obstacles) or states that did not feature in solutions of any previous tasks (*e.g.*, unuseful states). Then, a probabilistic mixture model is fit to the bag of states. The purpose of this is to identify regions in the state space that are more important than others. These regions are called the generalised bottlenecks. The goal of the agent is to acquire policies for acting inside these regions. The options are acquired through *policy reuse*, where parts of the original policies are applied as is, creating multiple options at once. After identifying the options, they are added to the action set of the agent and new tasks are learnt using them. The resultant policy is added to the collection of policies and the process repeats for the next task.

2. We label the trajectories with respect to their success $(+)$ or failure $(-)$ in reaching the goal. A trajectory is successful if it terminates at the goal, while trajectories that elapses before reaching the goal are considered failures. Only the states in successful traces in the instance i, $\boldsymbol{\tau}_i^+ = \biguplus\{\tau_i \,|\, g(s_{|\tau_i|-1}) = 1\}$, are used to make the input state dataset, $\mathcal{D} = \biguplus_{i=0}^{n} \boldsymbol{\tau}_i^+$. Note that many copies of the same state

could exist in $\mathcal{D}$, which would be an indication of the importance of that state.

3. To identify our concept of bottlenecks, being contiguous regions in state space that show up frequently in successful trials, we search for a generative probabilistic mixture model of the state dataset $\mathcal{D}$. The combination of such a model with policy reuse is what defines ILPSS. In more concrete terms, we want to find a model that maximises the term

$$\log \mathrm{P}[\mathcal{D};\theta] = \sum_{i} \log \mathrm{P}[\boldsymbol{\tau}_i^+|\theta] \tag{3.4.1}$$

$$= \sum_{i} \sum_{\tau} \log \mathrm{P}[\tau_i^+;\theta], \tag{3.4.2}$$

assuming independence of traces for a specific task instance given the model, as well as independence of different instances' traces, being drawn from the same model which summarises all the correlations. Here, we are ignoring time and using the traces as *bags of states*,

$$\log \mathrm{P}[\tau;\theta] = \sum_{s\in\tau} \log \mathrm{P}[s|\theta] \tag{3.4.3}$$

The model we use here for $\mathrm{P}[s|\theta]$ is a Gaussian mixture model (GMM) with a parameter C, representing the number of the multivariate Gaussian kernels. Each kernel is described by its mean μ_k and covariance matrix Σ_k in S,

$$\log \mathrm{P}[s|\theta] = \log \sum_{k} p_k \mathcal{N}(s;\mu_k,\Sigma_k), \tag{3.4.4}$$

but any suitable alternate model can also be used. The choice of the model is task-specific and does not affect the operation of ILPSS.

For our choice of model, the parameters are the weights p_k, means μ_k and covariances Σ_k for $k = 1 \ldots C$ for the C components. We allow p_k to be *instance-specific*, *i.e.* each task instance i has it own weighting/importance of the components, $p_k = (p_k{}^i)_{i=1}^n$. On the other hand, we force μ_k and Σ_k to be *instance-independent*, *i.e.* all the task instances share the exact same components. That is, we push toward finding a common set of mixture components across the different instance policies, regardless of their relative significance in the various instances. This shared structure is what we are after for lifelong transfer.

The mixture components define kernels in the state space, $\mathcal{K}_k(.) = \mathcal{N}(.;\mu_k,\Sigma_k)$. These are an important component of our representation of the policy space in

that we assign policy fragments to the state regions defined by them, $S^k = \{s \in S : \mathcal{K}_k(s) > \Psi\}$ for some cutoff probability Ψ.

For a specific kernel and for some policy π_i, we borrow a policy fragment from π_i by restricting it to the kernel's state space region: $\pi_i^k : S^k \times A \to [0,1]$. The scale (or weight) of a component k in a task instance i (captured by ${p_k}^i$) is ignored as it is irrelevant from a structure-learning point of view. We do this for all i.

To fit the model for some desired number of kernels C, we use an adapted Expectation-Maximisation (EM) algorithm (Dempster et al., 1977) that takes our structural constraint into account. For the Expectation phase, the kernels are initialised randomly and the model responsibilities are computed accordingly, while in the Maximisation phase the parameters of the model are recomputed using the responsibilities.

4. Every kernel can spawn a set of policy fragments $\{\pi_0^k, \pi_1^k, \ldots, \pi_n^k\}$, each borrowed from a different policy. We package every policy fragment in an option, giving the option collection $\mathcal{O}_k$. The domain and the termination condition of $\mathcal{O}_k$ options are both the PDF defined by the kernel. That is, an option from $\mathcal{O}_k$ is allowed to start at a particular state s with a probability equal to the normalised kernel function $\bar{\mathcal{K}}_k(s) = \mathcal{K}_k(s)/\max \mathcal{K}_k(.)$, and will continue stochastically with the same probability, and will terminate with probability $1 - \bar{\mathcal{K}}_k(s)$. The option policies are the kernel-restricted policies $\pi_0^k, \pi_1^k, \ldots, \pi_n^k$. In other words, we are reusing policies of previous tasks in controlled regions of state space where they fared well in experience. We discuss scalability issues in the next section.

5. Next, the agent is presented with another task instance, m_{n+1}. In a similar fashion, the agent learns a new policy π_{n+1} using SMDP learning, with the new options added to the action repertoire. The resultant policy will contain, in addition to normal actions, pointers to $\pi_0, \ldots, \pi_n$. [2]

6. π_{n+1} is added to the set of input policies, and the exact same procedure is adopted (extracting traces, bag of states, probabilistic model, then options) and this continues in what can be described as a lifelong learning process.

The full procedure is summarised in Algorithm 2.

[2] Note that the use of SMDP learning allows the agent to start with no input policies, and thus learning the task at hand from scratch, then initiating the process.

Algorithm 2 ILPSS: Incremental Learning of Policy Space Structure

Require: number of components C, input policies $\pi_0, \ldots, \pi_n$, cut-off probability Ψ.

1: Initialise the repertoire $\mathbf{O} \leftarrow \varnothing$.
2: **for** every new instance **do**
3: Generate state traces $\{\tau\}$ from input policies through simulation or runtime recording.
4: Label and extract successful traces $\{\tau^+\}$.
5: Create a 'bag of states' dataset $\mathcal{D} = \biguplus_{i=1}^{n} \{\tau_i^+\}$.
6: Fit a probabilistic mixture model to $\mathcal{D}$ using EM algorithm, generating a set of kernels $K = \{\mathcal{K}_k(.)\}_1^C$ in state space.
7: Extract state regions from the kernels, $S^k = \{s \in S : \mathcal{K}_k(s) > \Psi\}$, for $k = 1, \ldots, C$.
8: Restrict input policies to kernel state regions, $\pi_i^k : S^k \times A \rightarrow [0,1]$, for all k and i.
9: Create a set of options $\mathcal{O} = \bigcup_{k=1}^{C} \mathcal{O}_k$ with $o_i^k = \langle \bar{\mathcal{K}}_k, 1 - \bar{\mathcal{K}}_k, \pi_i^k \rangle \in \mathcal{O}_k$, $i = 1 \ldots n$.
10: $\mathbf{O} \leftarrow \mathbf{O} \cup \mathcal{O}$.
11: Learn a policy π_{n+1} for the new instance using $\mathbf{O}$ via SMDP learning.
12: Add π_{n+1} to the input policies.
13: $n \leftarrow n + 1$.
14: **end for**
15: **return** Option set $\mathbf{O}$.

3.5 Scaling ILPSS

3.5.1 Managing experience

To control the number of options that are maintained and used online by the agent, a process of temporal discounting, or *forgetting*, is employed. That is, the behaviours that do not get used often will become less likely to be used afterwards and more likely to be forgotten. On the other hand, options that are used often will persist and may develop and generalise through reuse inside future options.

Discounting an option makes it less likely to survive in $\mathbf{O}$. One way to do this is by shrinking the support of the option in state space, making it less likely to be chosen later and concentrating it to less states. After crossing a threshold on effective option size Υ, the options can be pruned out of $\mathbf{O}$.

In our implementation, the initiation set of an option is defined using a Gaussian

distribution, thus we implement forgetting by multiplying the covariance of the distribution Σ^o with a scalar $\xi < 1$ in every new instance in which the option is not used. The procedure *Option Discounting*, that can be invoked after every ILPSS learning trial, is detailed in Algorithm 3.

Algorithm 3 Option Discounting

Require: options $\mathbf{O}$, newly-learnt policy π, forgetting parameter ξ, effective kernel size Υ.

for every option o in $\mathbf{O}$ **do**
  if o is not used in π **then**
    $\Sigma^o \leftarrow \xi\Sigma^o$.
    Update the option domain S^o.
    if $\int \mathbb{1}_{S^o}(s)\,ds < \Upsilon$, with $\mathbb{1}_{S^o}(.)$ being an indicator function, **then**
      $\mathbf{O} \leftarrow \mathbf{O} - \{o\}$.
    end if
  end if
end for
return Option set $\mathbf{O}$.

3.5.2 Trace sampling

Density estimation methods in general, including EM, tend to require heavy computation. This is only made worse by the incremental increase in the size of the input dataset. In ILPSS, the dataset should represent the states that are used in successful trials more often as the agent is experiencing more instances. The complexity of the computation depends on the number of involved states nN^+L, where n is the number of the included instances, N^+ is an upper bound on the number of successful traces per instance, and L is an upper bound on the length of the trajectories.

To control the size of the state dataset we employ three 'down-sampling' measures. First, we choose a subset of previous instances and their policies to be considered for making the dataset rather than using all of them, based on their similarity to the newly learnt instance. This will still uncover commonalities between the included instances and encourage generalisation and reuse. If similarity cannot be estimated, instances can be selected randomly, but metrics are then needed to ensure the usefulness of the options, *e.g.* by measuring their coverage of the input traces.

The second measure is to sample traces from the successful traces of the chosen instances. This can be a random selection, but if a cost function is defined over the traces, the more successful traces can be favoured over the less optimal ones.

The third measure is to sample states from the traces of the selected instances, up to the desired dataset size. One intuitive procedure would be to sample using the relative frequency of the state occurrence in the dataset. This way, only the states that are not very important will be removed, while the dense areas, which we are interested in, will be well-represented. Then, the continuity of the state space and the statistical-model fitting would rectify for the gaps in the input set.

Using these measures would reduce the complexity to $\hat{n}\hat{N}^{+}\hat{L}$, where $\hat{n} < n$ is the number of sampled instances, $\hat{N}^{+} < N^{+}$ is the number of sampled trances, and $\hat{L} < L$ is the number of sampled states per trace.

3.6 Experiments

3.6.1 Rooms environment

To test the lifelong learning aspect of the proposed framework, we use the static and low-dimensional domain of rooms environment in order to visualise the set of skills that can be produced and maintained after experiencing many instances of the domain. Here, the task of the agent is to navigate in a rooms environment from some initial position to some goal position, both chosen randomly in every trial. The domain is a 2D grid world of size 25×25 with walls and exits. The agent moves in the 4 canonical directions (up, right, down, left). After completing each trial, the agent uses ILPSS to generate a set of options that are used in the next trials.

In the experiment, the number of ILPSS components is set to $C = 4$, and the forgetting rate is set to $\xi = 0.8$. The agent receives -1 for each time step, and $+10$ for reaching the goal. We show the skills acquired by the agent after experiencing 25 instances of the task in Fig. 3.5. Applying option discounting allows unused options to shrink in coverage and then disappear once below a specific threshold.

In Fig. 3.6 we show a heat map of the visitation frequency of the states of the domain, where each state is coloured according to its relative visitation frequency in a set of 30 experienced instances. This is one of the metrics used to identify bottlenecks in discrete worlds in option discovery literature. The white dots on the figure are the means of ILPSS kernels extracted using the same traces. What this shows is that the

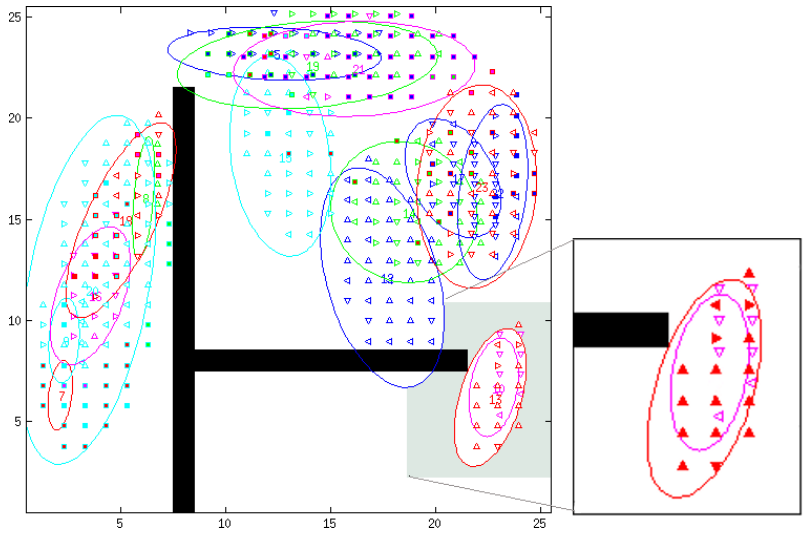


Figure 3.5: ILPSS in the rooms environment. LEFT: The acquired kernels after solving 25 instances of the task. The black lines are walls, and each ellipse shows 70% support of one of the discovered options. The arrows inside the ellipses show the option policies, which move the agent through different rooms and exits. RIGHT: The two options which appear in the highlighted bottom-right corner in the state space, zoomed-in. They allow passing in and out of the room through the exit.

framework is capable of approximating the discrete bottlenecks of visitation frequency methods.

For higher-dimensionality dynamical domains, the exposed structure will not be easily interrupted or visualised, but ILPSS will maintain the semantics of the skills in the structure as shown in this simple demonstration.

3.6.2 RoboCup simulated robotic soccer

The aim of this experiment is to test the transfer capability of the proposed framework in a continuous domain. We use the domain of simulated robotic soccer to demonstrate the improved performance of using the abstraction proposed by ILPSS. The task models a training drill for 2 vs. 2, in which the team with ball possession tries to cross the bottom line in the field *with the ball*.

The task is episodic, starting with the agents in set positions, and terminates successfully when the goal is achieved. On the other hand, the task fails when the adversaries intercept the ball, kick it out of the training region (35m$\times$35m), or when the episode elapses (100 time steps). The experiment is conducted using RoboCup 2D Simulation League Soccer Server (Noda and Matsubara, 1996) and the Keepaway extension (Stone

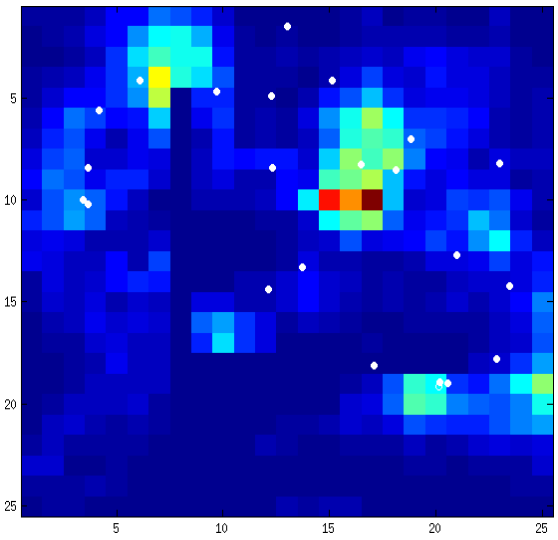


Figure 3.6: The visitation frequency of the rooms environment after 30 random instances. The darker colours are for less-visited states. The white dots superimposed on the heat map show the means of ILPSS kernels learnt using the same instances.

et al., 2005). The setup is shown in Fig. 3.7.

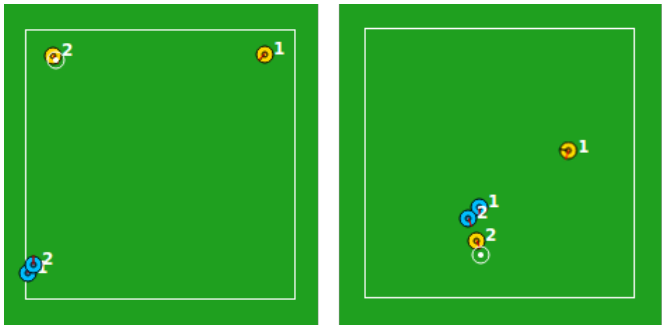


Figure 3.7: The experiment setup. LEFT: the attacking team starts from the top corners, while the defending team starts from the bottom. RIGHT: the task of the attackers is to reach the bottom line with the ball, with the middle point having the highest reward. The task of the defenders is to prevent that.

The state space is defined using 9 continuous state features describing the position and orientation of the learning agent with respect to other agents and the goal line. The features are: the distance between all the players and the goal centre point (4 features), the distance to the team mate and to the two opponents (3 features), and the distance and the angle of the team mate to its nearest opponent (2 features). The action space comprises 3 basic options: holding the ball, passing to the team mate and dribbling towards the goal line. The action set is enriched later with the discovered options. The

striker that has the ball is the only learning agent, while others use hand-tuned stochastic behaviours. When the ball is passed, the receiving player becomes the learning agent, while the other switches to the hand-tuned behaviour.

The variability in the domain comes from the different tendencies toward the ball and the goal line different opponents have. In the experiment, four opponent teams are used. The first two are only concerned about intercepting the ball, and they differ from each other by the coordination protocols between the two players. In the third type, one opponent intercepts the ball while the other protects the goal line. In the fourth type, both opponents protect the goal and only go for the ball with a small probability, making it different from the first three and harder to beat.

The opponents are presented to the learning agents sequentially, with ILPSS running after each to generate 5 options (that is, $C = 5$). After experiencing the first three opponents and learning a repertoire of 15 options, the performance against the last opponent is compared to learning the same task from scratch. The aim of this comparison is to check the usefulness of the discovered abstraction. The results are shown in Fig. 3.8 where the experiment is repeated 5 times.

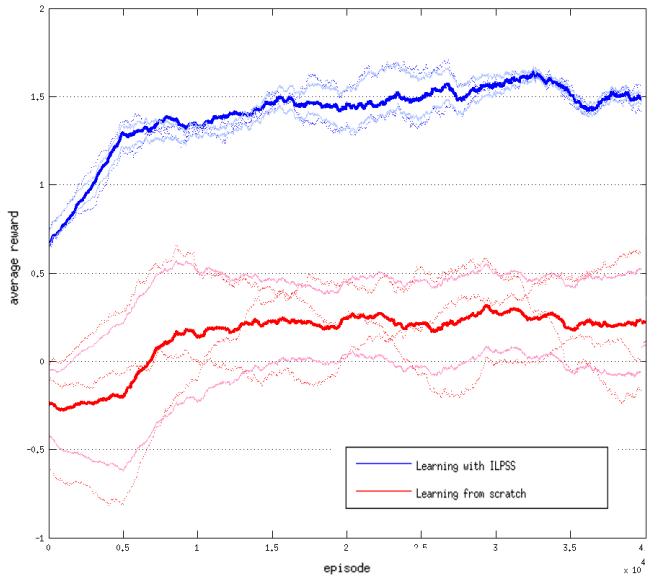


Figure 3.8: (BLUE) Average reward achieved against the fourth opponent team using options acquired by ILPSS in three other task instances, compared to learning anew (RED). The experiment is repeated 5 times.

As the results show, using the set of options discovered by ILPSS gives a head start in performance. This comes from the fact that the four tasks share significant components (*e.g.*, all the tasks require the agent to proceed toward the goal line. The

Q-Learning agent does not have this domain knowledge). This shows that ILPSS is able to produce useful abstractions in the policy space that allows faster convergence in a novel instance.

We show in Fig. 3.9 example traces generated by the most used options against the fourth opponent team. It appears that the agent developed a behaviour to approach the adversaries closer to the middle before beating them to the goal line at a short range, which is a sensible approach considering the opponent's conservative behaviour.

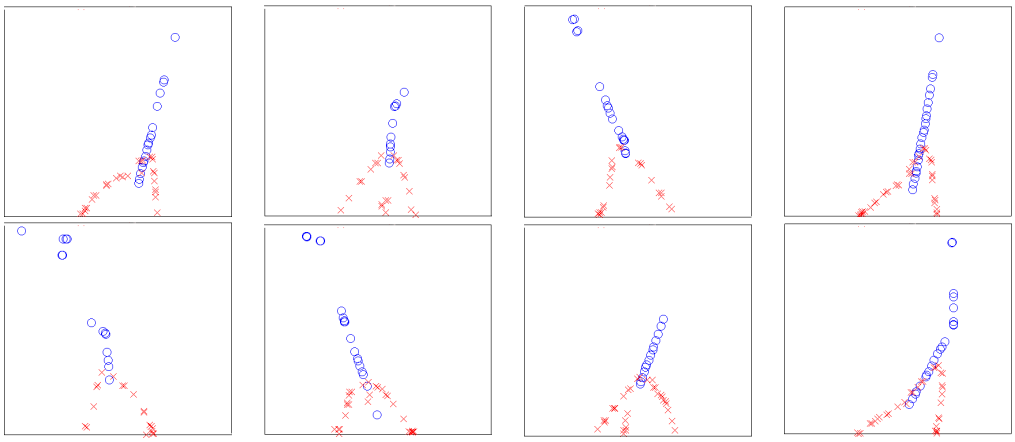


Figure 3.9: Traces from discovered options in the simulated soccer domain. The blue circles represent the location of the learning agent in the field as it moves toward the goal line. The red crosses represent the two adversaries. The figures shows traces through time of sample runs. The gaps can be justified by passing actions which change the identity of the learning agent.

3.7 Concluding Remarks

ILPSS is built around fitting a probabilistic mixture model to a set of successful trajectories of a family of tasks to uncover useful commonalities. This assumes that the task instances do indeed share common structure, that it is possible to solve some instances and generate complete traces, and that it is possible to fit a probabilistic model to the resulting state dataset. In a very high-dimensionality, complex domain, some of these assumptions may not be readily met, and thus the method would not be suitable.

On the other hand, the specific techniques and models used in this chapter (like EM and GMMs) are not essential to the operation of ILPSS, and they can be replaced with any more suitable, state-of-the art tools to tackle high dimensionality.

The intuition behind the incremental nature of the framework is in 'biasing' the evolved hierarchy to find common components that would be useful for lifelong learning. Reusing the previously-learnt policies in options while learning a new instance encourages the agent to search for a useful policy subspace (spanned by the options), before extending the search to the full policy space. If a satisfactory solution can be found early, these behavioural commonalities would be developed and maintained. We argue that this kind of structure is what is needed when facing a novel instance of a task.

On the other hand, a biased set of instances presented early on to the agent may cause a bias in the abstraction, especially when combined with random trace sampling. This could delay the convergence into a meaningful skill set sufficient for the domain. This is analogous to negative transfer in transfer learning.

Bottleneck methods work usually for single tasks, and they work best for small, discrete state spaces for which complete interaction graphs can be built, or sufficient visitation information can be collected. Also, the aim of these methods is to locate potential subgoals for a single task, leaving the policies to achieve them to be subsequently learnt. On the other hand, ILPSS generalises that concept of a bottleneck state into a continuous, probabilistic 'concept' in state space that appears often in many instances. This might make ILPSS more appropriate in large and continuous domains. Also, it immediately discovers where the *existing* policies might be useful, avoiding the two-step process of explicit discovery of subgoals followed by policy learning.

ILPSS may appear close in spirit to skill trees of (Konidaris et al., 2010). However, the assumptions about the two methods are different. ILPSS is devised to extract common structure in many task instances in a domain, rather than backchaining to solve a specific task as in skill trees.

An earlier method that defines skills with probabilistic domains is the SKILLS algorithm (Thrun and Schwartz, 1995) where the aim is to find a compact set of macro-actions that minimises the description length (DL) of the actions for a task. There, the policy of a skill is learnt in a way that balances performance loss with compactness gains. For ILPSS, policies are chosen to support an extensive family of tasks, while compactness is observed through the option discounting process.

The work in (Foster and Dayan, 2002) investigates the structure in the space of value functions of optimal policies for a family of related tasks. For that, they employ a mixture model as the generative process of value functions. The components of that model encode the discontinuity in the value function caused by inherent properties of the domain (*e.g.*, location of walls and barriers in a room environment). Then, the

model is used to accelerate learning in new instances by augmenting the state space with the discovered features.

ILPSS defines structure *implicitly* in policy space, through a probabilistic model in state space combined with reusable policy fragments. Our components in state space do not necessarily feature smooth value functions, but rather regions of stability and good performance. Also, our framework only requires sample traces from good policies, while other methods require complete explicit representations, like value functions or transition graphs, which may allow ILPSS to be used even before learning converges, especially in big worlds.

Some prior work has looked into the use of probabilistic models in hierarchical reinforcement learning. For example, (Manfredi and Mahadevan, 2005) defines a more complicated graphical model for both state and policy abstraction that is trained from sample trajectories using Expectation-Maximisation (EM) algorithm, but constrained to a single task.

Our use of Gaussian kernels to describe option domains may not be the optimal choice, as they suggest a kind of symmetry over various dimensions which may not be true. However, they are an intuitive choice from a computational point of view.

We introduce a framework for learning and refining a structural description of the space of policies for a set of qualitatively-related task instances. We employ a principled probabilistic method to decompose the state space and relate the learnt model with policy fragments through policy reuse. The resulting abstraction is maintained using a set of temporally-extended options. We note that learning continually is essential for extracting useful decompositions in policy space.

This method does not require explicit representation of the space of policies, and it does not rely on the optimality of the input policies, which allows it to scale well. Rather, only a set of trajectories of successful trials and the policies that generated them are needed to enable the agent to produce a rough solution to a novel instance, which can then be refined by learning. Surely, better policies would allow discovering better options, faster.

Testing this framework on large problems to understand its scalability is a topic of future work, as well as understanding the effects of state abstraction and dimensionality reduction on the produced action abstraction. Also, further investigation is needed on defining the boundary of a skill in state space, *e.g.*, using a mixture model to represent each option.

Chapter 4

Policy Space Abstraction using Computational Topology

4.1 Introduction

In this chapter, we develop a method of abstracting policies via the qualitative similarities of their induced behaviours. We use tools from computational topology to identify distinct classes of trajectories in a configuration space. These classes are different with respect to the constraints that the domain has, *e.g.*, the obstacles in the case of navigation.

We start with a novel construction of a simplicial complex to represent the free configuration space of a robot, then we map sample trajectories to paths on the complex and classify them using homology theory. We show how the resultant classes change when finer or coarser representation of the complex is used, and we show how all the representations can be used simultaneously using the machinery of persistent homology. Finally, we use the trajectory classes to define a small set of representative options for a domain.

Fig. 4.1 gives an overview of the approach of this chapter.

4.1.1 Contributions

The main contributions of this chapter are:

- The introduction of filtrations of simplicial complexes and persistent homology for multiscale topological robot motion classification.
- A sample-based algorithm to classify homotopy classes of paths in $\mathrm{X}_\varepsilon \simeq \mathcal{C}_\mathrm{f}$ at all scales, and a sample-based algorithm to classify homotopy classes of paths when cost is defined.
- A topological metric for option discovery from demonstrations.
- An experimental evaluation of the approach in simulation in 2 and 4 dimensions, and an evaluation with a physical robot system.

4.2 Multiscale Topological Trajectory Classification

Robot Motion Planning refers to the problem of finding continuous paths in the free subset of the robot's *configuration space* that lead from a start point to an end point. The robot's configuration space is the space of all configurations, *i.e.* possible assignments to all degrees of freedom of the robot. The *free* configuration space is the subset of

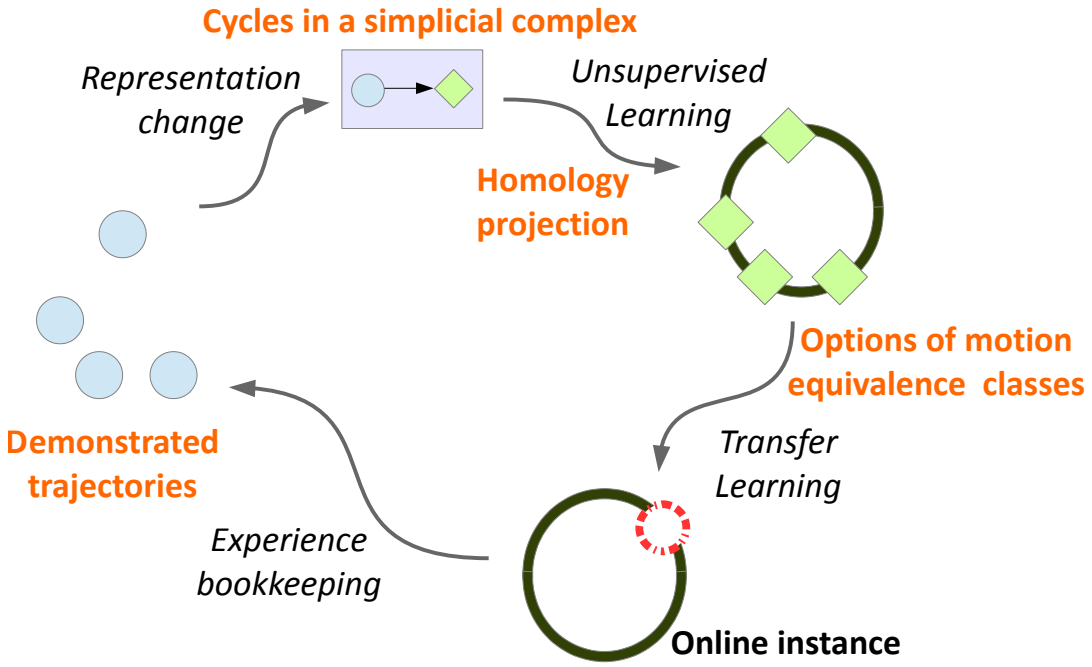


Figure 4.1: Overview of the approach in this chapter. From a set of experienced trajectories, a simplicial complex is built and each trajectory is mapped into a cycle on the complex. The structure learning happens using persistent homology which can classify cycles in a simplicial complex into equivalence classes, where equivalence of two trajectories is defined as the possibility of continuous morphing of one to the other without crossing any obstacle in the domain. These classes can then serve to make a set options that are qualitatively-different.

assignments that avoid obstacles in the robot's environment . For a holonomic mobile robot in a 2D world this could be the possible free positions in the plane that do not involve walls, while for a multi-joint robotic arm, it is the space of joint angles that are realisable and that do not collide with obstacles.

Popular motion planning algorithms, such as Rapidly Exploring Random Trees (RRT) and Probabilistic Roadmaps (PRM), start by estimating the free configuration space by a graph constructed using random samples with a collision detection mechanism (LaValle and Kuffner, 2001; LaValle, 2006; Kavraki et al., 1996). Path planning can then be performed efficiently on that graph. However, even though this representation captures path-connectedness in the domain, it fails to capture *higher-order homological and homotopical information* of the domain, such as, *e.g.*, contractability and equivalence of paths with respect to the domain constraints.

We aim to develop a representation for motion planning that captures this kind of information to be used for trajectory summarisation and abstraction extraction.

4.2.1 Approach

Consider a simple navigation task with a start state s and a goal state t as shown in Fig. 4.2.

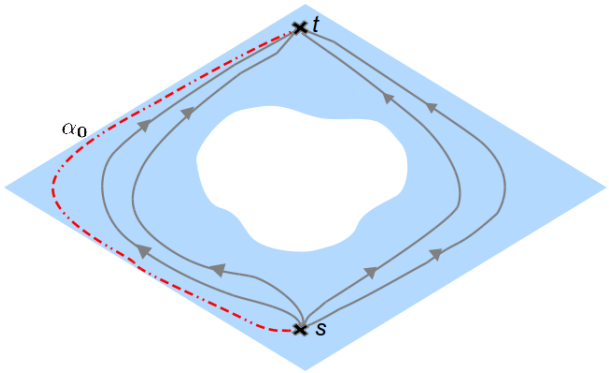


Figure 4.2: A depiction of a simple domain and a collection of trajectories.
The task is to reach from s to t. The 'hole' in the middle represents a constraint in the configuration space, *e.g.* an obstacle in a navigation domain. α_0 is the augmented edge that represents the task.

From a set of trajectories of that task (sequences of combined agent positions and actions) we build a topological construct (a simplicial complex) which is a discretisation of the agent's knowledge of the task, and in general a generalisation of a graph. This construct is useful because it enables us identifying all the *different ways* to achieve the task in a set of input trajectories using homology theory (for example, in Fig. 4.2 there are two different ways to achieve the task: east of the obstacle, or west of it). These define equivalence classes of trajectories with the property that *no path belonging to one equivalence class can be continuously deformed to any path in any of the other equivalence classes*. Moreover, we can classify any discretised trajectory on the complex to the class it comes from.

Additionally, the construction permits the utilisation of *persistent homology*. Using *filtrations* of simplicial complexes, we do not have to commit to any specific discretisation of the domain, but rather we can work on different 'resolutions' of representation at the same time, 'zooming out' and 'in' the data. Furthermore, it is possible to include a state-based cost function into the representation to study the space of paths that are capped by some state-wise cost threshold.

This construction subsumes previous methods that rely only on graphs, and allows for new perspective into qualitative and multiscale robot motion planning, taking global perspective into account when planning locally.

We now describe the algorithm for identifying and classifying trajectories into homology equivalence classes as in (Pokorny et al., 2014).

4.2.2 A simplicial complex of a configuration space

Starting from a set of samples $V = \{v_1, \ldots, v_n\}$ from the collision-free configuration space of some robot $\mathcal{C}_f \subseteq \mathcal{C} \subset \mathbb{R}^d$, we want to build an abstract representation of $\mathcal{C}_f$ using V that maintains the homotopy equivalence relation.

We use the union of ε-balls space, $X_\varepsilon = \bigcup_{v \in V} \mathbb{B}_\varepsilon(v)$ for some $\varepsilon > 0$, as a candidate. If V is sampled uniformly and $\mathcal{C}_f$ is a smooth compact submanifold of $\mathbb{R}^d$, then according to the reconstruction theorem of (Niyogi et al., 2008), a sufficiently dense sampling deformation-retracts to the actual manifold for appropriately chosen ε. Furthermore, it is known that the Alpha complex (Edelsbrunner, 1995) of the point set V as a simplicial complex is homotopy-equivalent to X_ε.

Finally, (Bauer and Edelsbrunner, 2014) have recently proven that the Delaunay-Čech complex DC_ε is homotopy-equivalent to the Alpha complex in arbitrary dimension. Then, DC_ε is homotopy-equivalent to the original configuration space under the previous sampling conditions, and thus it is possible to compute the homology groups of DC_ε at all scales simultaneously by means of the persistent homology algorithm as an alternative to computing the homology groups in the original space.

Since we are interested in classifying trajectories, which map to edge paths on the complex, we will focus particularly on the first homology group, $H_1(DC_\varepsilon)$ which only depends on the simplices of dimension 2 or less. Hence, we will build the complex DC^2_ε that only has these simplices.

For some ε, we extract the 0-,1- and 2-simplices (vertices, edges and triangles, respectively) from the ε-filtered Delaunay triangulation of V. That is, the triangulation with only the edges that are shorter or equal to ε.

- A 0-simplex (vertex) is added for each point in V.
- A 1-simplex (edge) is added for any distinct pair of two points in the ϵ-filtered Delaunay triangulation of V. Edges reflect one-step reachability in the configuration space.

- A 2-simplex (triangle) is added for every triangle in the ϵ-filtered Delaunay triangulation of V. Triangles represent covered surfaces in the configuration space that the agent knows how to span, and hence can be collapsed when considering the topological features of the configuration space.

The results of (Edelsbrunner, 1995; Bauer and Edelsbrunner, 2014) show that there exists a deformation retraction from X_ε to DC_ε, so that any continuous path in X_ε can be continuously deformed to an edge path in DC_ε. Moreover, this retraction induces an isomorphism of homotopy groups such that the resulting edge path maintains the homotopy class of the continuous path.

We will treat edge paths as undirected chains of edges. Hence, we define the group of chains to be over $\mathbb{Z}_2 = \mathbb{Z}/2\mathbb{Z} = \{0,1\}$. That is, the coefficient of an edge in a chain can only be 0 or 1 so that it can either appear once in a chain, or not appear at all. Moreover, adding two edge chains will remove all the shared edges between them, making addition and subtraction the same operation.

4.2.3 Homotopy classification of paths

Assuming a set of trajectories between two fixed points $s, t \in V$, we would like to determine the homotopy classes of these trajectories. Consider again the situation in Fig. 4.2, and consider an arbitrary trajectory α_0 that connects the start point s and the end point t (*i.e.* representing the agent's task). Any original trajectory between s and t when augmented with α_0 now represents a closed loop. Mapping the trajectories to path edges on DC_ε for some $\varepsilon > 0$, these closed loops are now 1-cycles.

This construction is particularly interesting because the space of 1-cycles, with respect to discontinuities in the complex, can be analysed using simplicial homology. That is, we can understand the space of all the ways to reach between two points (the equivalence classes) only by augmenting the input trajectories with one arbitrary trajectory between the same two points, mapping that to a simplicial complex, and applying standard tools from homology theory. The one remaining piece needed is the ability to classify any input trajectory to its equivalence class from the resulting alternatives.

Following (Pokorny et al., 2014), consider two 1-cycles from $Z_1(DC_\varepsilon)$ corresponding to the edge paths α_u, α_w, both from s to t, when augmented with α_0. If the two cycles have the same equivalence class in $H_1(DC_\epsilon)$, adding them together over $\mathbb{Z}_2$ will be 0, so that $[\alpha_u + \alpha_w] = 0$ as well. That is, the equivalence class of the sum of the two

paths is 0, *i.e.* the edge paths are homotopy equivalent. On the other hand, if the cycles happen to be from different equivalence classes, adding them up will be non-zero, and hence $[\alpha_u + \alpha_w] \neq 0$. Hence, we can use the coefficients of the cycles when expressed in some basis of the homology group to define path equivalence classes. The same argument is valid when we consider the persistent homology between filtration indices i and j, $H_1^{i,j}(DC)$.

To get a basis for the homology group $H_1^{i,j}(DC)$, we use a result from the reduction algorithm for persistence in (Carlsson, 2009; Edelsbrunner and Harer, 2010). The basis for the first homology group between filtration indices i and j is:

$$T^{i,j} = \{R_t : (s,t) \in P_R, s \leqslant i, t > j\} \cup \{V_s : s \in E_R, s \leqslant i\},$$

where R is the reduced boundary matrix using left-to-right column operations on the boundary matrix M, R_t is the t^{th} column of R, P_R is the set of *persistent pairs*, E_R is the set of *essential cycles*, V is the matrix that keeps track of the column operations, and V_s is the s^{th} column of V (see Appendix. A).

Then, by expressing any cycle (made by concatenating an edge path α, corresponding to an input trajectory, with the fixed trajectory α_0) in that basis we get a vector of coefficients that encodes the class of the trajectory. Paths homotopic to α_0 will get the 0 vector class. Two paths α_u, α_w are not homotopy-equivalent if their coefficients are different, and that test can be performed for a specific scale ε by setting $i = j = \varepsilon$, or simultaneously for a range of scales when $i < j$. Then, the paths will have the same class only if they are equivalent under all the scales in consideration.

An illustration of this is in Fig. 4.3. The displayed simplicial complex has two holes (white) at some filtration value i (top row). The red path corresponds to α_0 and we display three edge paths $\alpha_1, \alpha_2, \alpha_3$ in blue which have mutually different coordinates in $H^{i,i}$. In the bottom row, we display the complex at a filtration index $j > i$ at which the smaller hole becomes filled, and hence the class of the third edge path becomes the same as the second. The other two classes do not change at this scale, or in other words, they persist from i to j.

4.2.4 Filtrations with cost landscapes

Assume that a cost-function $c : \mathcal{C}_f \to \mathbb{R}$ is prescribed for the domain, giving each possible configuration a specific cost. Our aim now is to not only classify paths in the space $\mathcal{C}_f$, but to take into account constraints defined by a maximal threshold for the cost function. That is, we also want to discern how paths negotiate the cost landscape.

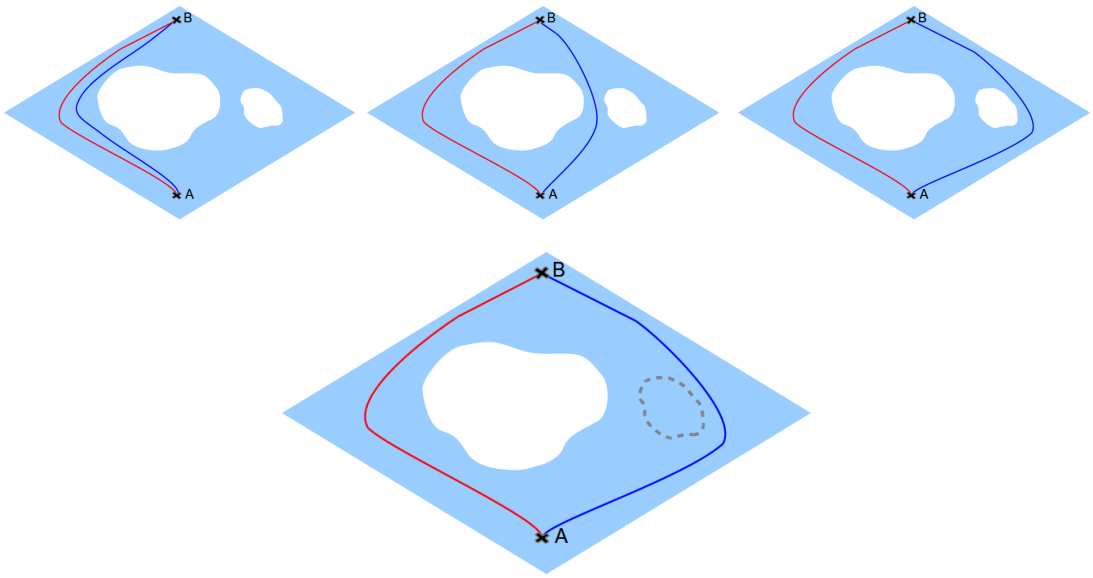


Figure 4.3: Illustration of three paths $\alpha_1, \alpha_2, \alpha_3$ between fixed points (in blue) from different homology equivalence classes at some scale. The red path is α_0. The figure in the bottom row shows the death of a topological feature (the smaller hole) when the scale is increased, and thus, the loss of an equivalence class and the change of the classification of α_3.

To map the cost function to the complex construction, we assume that the cost of a simplex $\sigma = \{v_0, \ldots, v_k\} \in DC_\varepsilon$ is simply the maximum cost among its faces, $c(\sigma) = \max(c(v_0), \ldots, c(v_k))$. This creates a different filtration of simplicial complexes $L_{\epsilon,\lambda} = c^{-1}((-\infty, \lambda])$ with the cost threshold λ. Nonetheless, the classification algorithm works precisely in the same way, regardless of the semantics of the filtration and the resolution variable.

One example is in Fig. 4.4, where cost is depicted as height in a 2D configuration space of size 250 by 500. For some $\varepsilon = 10$, the space has two holes (white regions). Moreover, we set a threshold λ for the permissible cost values. The blue regions in the figure are configurations with above-the-threshold costs, so that they are removed from the configuration space creating additional holes.

The top row of Fig. 4.4 depicts different equivalence classes for a cost threshold $\lambda = 70$, showing 100 paths of each using a fixed reference path (red). Note that both peaks in the cost landscape are truncated at this threshold. In the bottom row, throttling the threshold up to $\lambda = 90$ causes one of the cost holes to disappear, and thus makes the first two equivalence classes merge together into one, while the third class does not change.

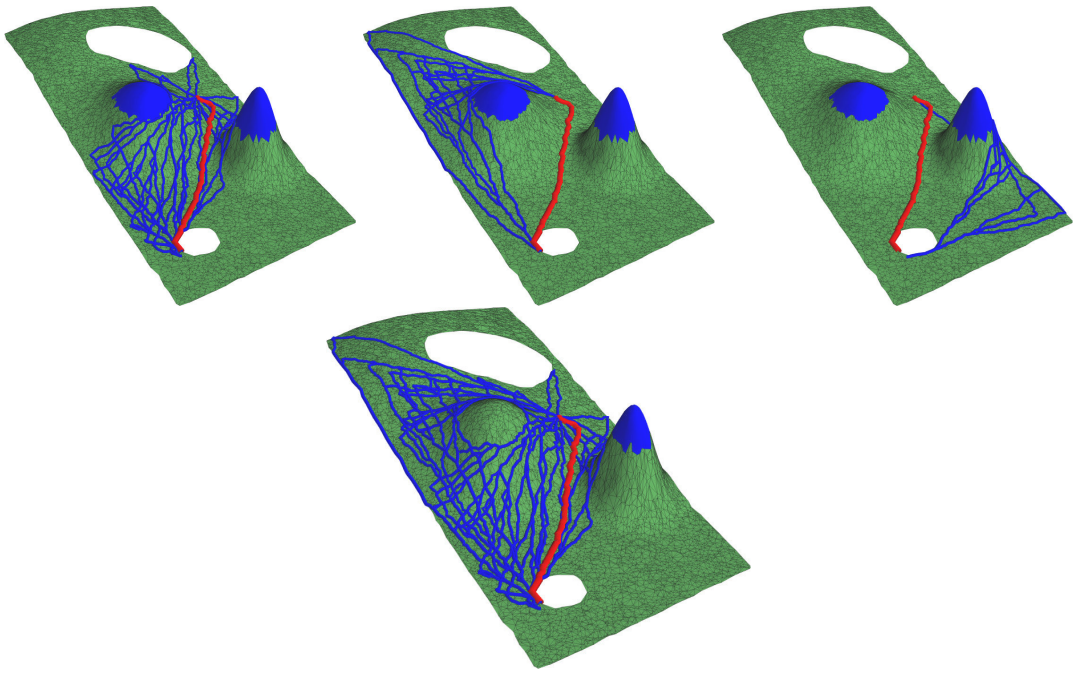


Figure 4.4: Path costs, and classes of paths depending on the cost threshold. The white areas are holes, and the blue areas have costs above the allowed threshold (*i.e.*, holes of a different kind). Each of the plots in the upper row shows a collection of paths from one equivalence class for a set cost ($\lambda = 70$) and resolution ($\varepsilon = 10$). For the more forgiving threshold in the bottom row plot ($\lambda = 90$), the first two classes, which were distinct in the top row under a stricter cost budget, merge (*Image from (Pokorny et al., 2014).*)

4.3 Topological Options

From an option discovery point of view, classifying input trajectories into equivalence classes allow creating abstractions of the policy space that build on the redundancy of equivalent trajectories. Such abstractions have two characteristics: they encode behaviour similarity with respect to the domain, and their support lies in active regions that have been visited by many demonstrations. These two global features are quite unique and different to the typical metrics used in option discovery, which usually corresponds to local features like visiting a certain bottleneck state.

4.3.1 Options from single trajectories

After we classify the input set of trajectories into a discrete set of distinct path classes under some filtration index, we can create an abstraction of the policy space by choosing one ‘landmark’ path from each homology class. For example, given a cost function that

is defined over the complete trajectories, we can select the trajectory from the class with the lowest cost to be the landmark for class.

By changing the filtration index, we get different equivalence classes, and hence different option sets. Then, we can ask about options which are persistent with the change of the persistence index, *e.g.* when the filtration is controlled by an adversarial or noise process and the persistent options are needed to achieve their task regardless of the realisable filtration.

4.3.2 Options from all trajectories

We can use all the trajectories in a class to create an option that represents the trajectory variability. We use the technique of GMM-GMR (Calinon et al., 2007) to construct a probabilistic model of the points in the support of the trajectories. For instance, a mixture of Gaussians can be used for the joint probability density function over pairs of points, predicting the next point given the current.

Using Gaussian Mixture Regression we can sample the expected behaviour for unseen points in the support of the class. This gives a probabilistic policy that imitates the class demonstrations which will be used as the class's option policy. The initiation set of the option is defined by the support of the trajectories, and the termination happens with respect to leaving that support.

4.3.3 ILPSS with Topological Options

To facilitate reuse, we integrate the topological options with the ILPSS framework (see Sec. 3.4). After we identify the topological options, We follow the procedure in (Hawasly and Ramamoorthy, 2013a,b) to identify important regions in the state space (the generalised bottlenecks), which probabilistically captures highly visited regions in the state space under different demonstrations. This step is needed to segment the trajectories and elicit their common factors. We restrict the option policies to these regions, and create an option for every class of behaviour that can be followed region-wise. Doing that for all identified regions, we create an option hierarchy for navigating the free configuration space under the current knowledge of the agent. The complete procedure is presented in Algorithm 4 for a fixed filtration value ε.

As discussed before, an alternative is to keep persistent options for a range of filtrations. Also, we can keep options at different filtration levels simultaneously. This can provide the basis for an *anytime* multiscale planning framework that, depending

Algorithm 4 ILPSS with ε-Topological Options

Require: number of components C, input trajectories $\gamma_0, \ldots, \gamma_n$, cut-off probability Ψ, filtration value ϵ, number of trajectories per instance m.

Initialise the repertoire $\mathbf{O} \leftarrow \varnothing$.
for every new instance **do**
  Classify the input trajectories with the filtration index ϵ.
  for every equivalence class $[\gamma]$ **do**
    Create a dataset of point-pair sequences from the trajectories of $[\gamma]$.
    Fit a probabilistic model $T_{[\gamma]}$ to the dataset using GMM-GMR.
    Generate a policy $\pi_{[\gamma]}$ that implements $T_{[\gamma]}$.
  end for
  Create a 'bag of states' dataset $\mathcal{D} = \biguplus_{i=1}^{n} \{\gamma_i\}$.
  Fit a probabilistic mixture model to $\mathcal{D}$ using EM algorithm, generating a set of kernels $K = \{\mathcal{K}_k(.)\}_1^C$ in state space.
  Extract state regions from the kernels, $S^k = \{s \in S : \mathcal{K}_k(s) > \Psi\}$, for $k = 1, \ldots, C$.
  Restrict policies $\pi_{[\gamma]}$ to kernel state regions, $\pi_{[\gamma]}^k : S^k \times A \rightarrow [0,1]$, for all k and i.
  Create a set of options $\mathcal{O} = \bigcup_{k=1}^{C} \mathcal{O}_k$ with $o_{[\gamma]}^k = \langle \bar{\mathcal{K}}_k, 1 - \bar{\mathcal{K}}_k, \pi_{[\gamma]}^k \rangle \in \mathcal{O}_k$, for all equivalence classes $[\gamma]$, $k = 1, \ldots, C$.
  $\mathbf{O} \leftarrow \mathbf{O} \cup \mathcal{O}$.
  Learn a new instance using $\mathbf{O}$ via SMDP learning.
  Acquire m trajectories from the newly-learnt policy and add them to the input set.
  $n \leftarrow n + m$.
end for
return Option set $\mathbf{O}$.

on available resources, can automatically choose a level of granularity for the desired behaviour in a new instance.

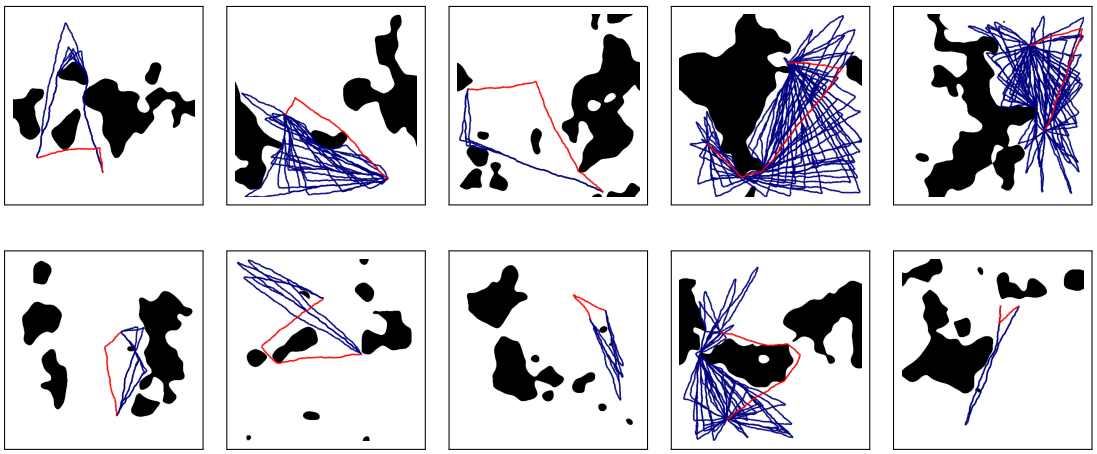


Figure 4.5: Example worlds with a single path class from $\mathrm{DC}_{\varepsilon_2}$ reconstructed from 10^5 samples. The red trajectory is α_0, and the blue trajectories are samples from an equivalence class (*Image from (Pokorny et al., 2014).*)

4.4 Experiments

4.4.1 Multiscale path classification in 2D

We generate a set of random 2D worlds $W_1, \ldots, W_{10}$ of size 512×512 by sampling Gaussian Random Fields and defining those regions above a threshold to be obstacles, as displayed in Fig 4.5. From the resulting free space $\mathcal{C}_\mathrm{f}$, we sampled $\mathrm{N} \in \{10^3, 10^4, 10^5, 10^6\}$ uniform samples. We compute the Delaunay-Čech filtration for all examples and record the time required for the computation of the Delaunay triangulation (utilising CGAL (CGAL, 2014)), the time for the construction of the filtration, as well as the time required to reduce the resulting boundary matrix to its reduced form R. In all these examples, we work with the boundary sub-matrix that sends 2-chains (triangles) to 1-chains (edges) during the reduction, since the remaining columns correspond to homology in dimensions $\mathrm{d} \neq 2$.

The Delaunay triangulation took on average 1ms, 2ms, 76ms, 810ms, as the sample size increased. The remaining construction of the filtration took 11ms, 31ms, 278ms and 3270ms respectively. Finally, the boundary matrix reduction took 14ms, 13ms, 76ms, 981ms on average as the sample size increased. All the computations were performed on an Intel i7 laptop with 8GB of RAM, and we only measured the computation time of the core algorithms, ignoring time required to load data into memory.

We investigated the filtration DC_ε at various thresholds and at a filtration value of $\varepsilon_1 = 25\sqrt{1000/\mathrm{N}}$ we found that $\mathcal{C}_\mathrm{f}$ was conservatively covered, while at $\varepsilon_2 =$

$35\sqrt{1000/N}$, the space was well covered with a minimum number of holes in collision free areas. The factor of $\sqrt{N}$ is chosen in order to ensure that asymptotically the same number of samples fall in a square of side-length ε_i as the sample size increases. See Fig. 4.6.

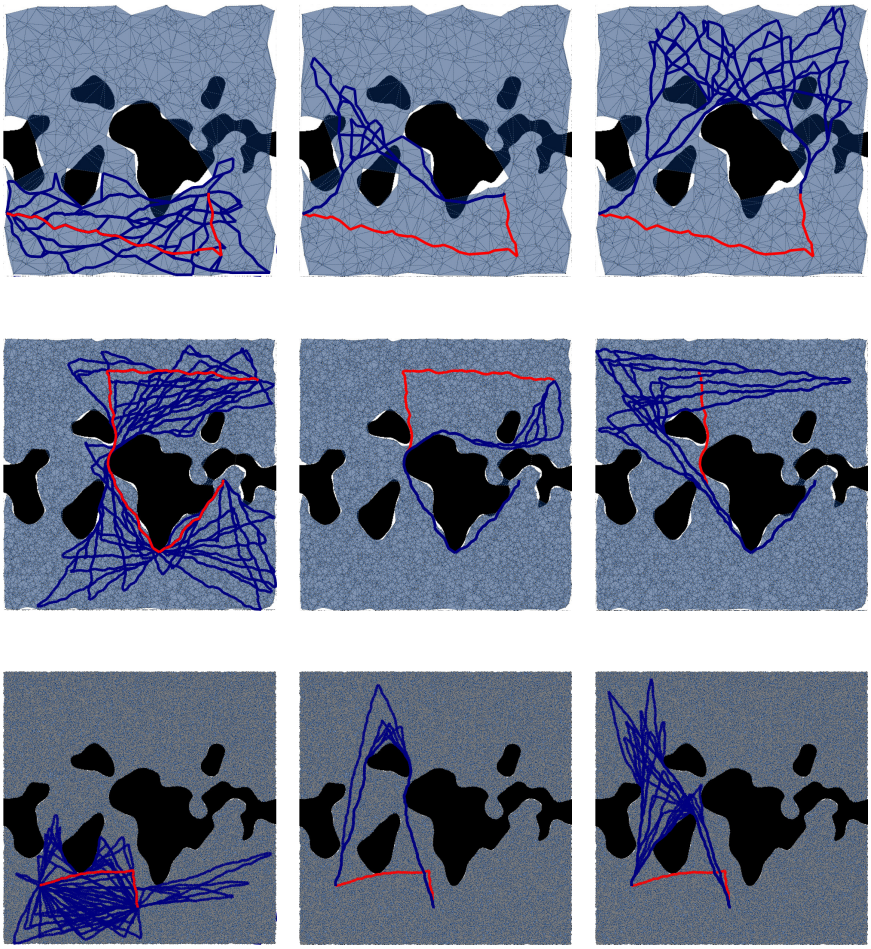


Figure 4.6: Example world W_1, with DC_ε for 10^3, 10^4 and 10^5 sample points per row. In each column, we plot paths $\alpha_1, \ldots, \alpha_s$ (in blue) which belong to some fixed homology class for the given sample size $H^{i,i}(DC(i))$. The fixed reference path α_0 is plotted in red. Two paths in different classes have a different homotopy class, while paths within a class are homotopy-equivalent, but the quality of the approximation $DC(i) \sim \mathcal{C}_f$ is only sufficient for 10^4 or more sample points as can be seen on the right in the first row. There, some 2-simplices (triangles) cover thin obstacle region in the centre-right (*Image from (Pokorny et al., 2014).*)

Determining an appropriate filtration value could be aided if additional information about $\mathcal{C}_f$ is available. For example, the persistence diagram in dimension 0 allows us to infer filtration settings at which DC_ε has a prescribed number of connected components.

In order to investigate interesting path classes, we generated a set of 1000 paths per world and sample setting as follows: In 10 trials, we selected two sample points v_1, v_2 at random and for each such setting selected another 100 random waypoints $w_1, \ldots, w_{100}$ from the sampled point-cloud. We determined shortest path from v_1 to w_i and then to v_2 utilising Dijkstra's algorithm on the 1-skeleton graph of $\mathrm{DC}_{\varepsilon_1}$.

We recorded the time required to calculate the *persistent cycle coordinates* for these paths and found that it took 1.8ms, 10ms, 115ms and 1750ms to calculate the feature for 100 query paths and for the respective sample sizes on average. These numbers encourage the use of the framework in realistic applications and in conjunction with continuous path optimisation engines.

4.4.2 Multiscale path classification in 4D

We now consider a planar robot arm fixed from the bottom and with 4 joints $\theta_1, \ldots, \theta_4$ displayed in Fig. 4.7. We constrain $\theta_2, \theta_3, \theta_4 \in [-0.9\pi, 0.9\pi]$, $\theta_1 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and furthermore disallow self-collisions of the arm and collisions with the environment. This yields a free configuration space $\mathcal{C}_f \subset \mathbb{R}^4$ which is difficult to describe explicitly. The robot has the task of moving from the start configuration displayed in blue to the goal joint configuration displayed in faint red in Fig. 4.7, *e.g.* in order to transport an object from left to right.

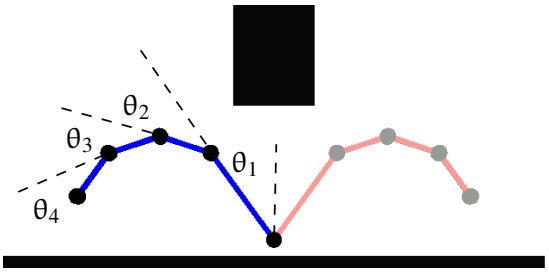


Figure 4.7: The robot arm in start configuration in blue on the left, and in goal configuration in red on the right.

We sample 10^5 poses uniformly in $\mathcal{C}_f$ using OpenRave (Diankov and Kuffner, 2008) and apply our framework, reconstructing a Delaunay-Čech simplicial complex 2-skeleton filtration which has more than 6.2 million triangles and 1.8 million edges. Fig. 4.8 displays the resulting H_1 persistence diagram which shows a single homological feature with large H_1-persistence in $\mathcal{C}_f$, *i.e.*, a single hole.

If we project the samples onto the first two angles, we obtain a 2-dimensional projection of the configuration space shown in Fig. 4.9 which empirically confirms this

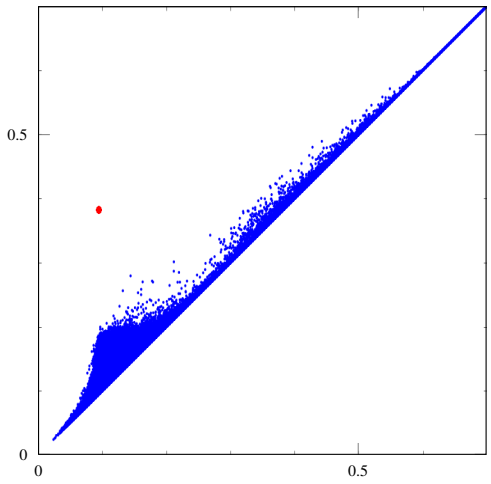


Figure 4.8: The persistence diagram for the robot arm configuration space reconstruction, with one persistent pair (red point) far above the diagonal. The horizontal axis is the birth time, and the vertical axis is the death time of homology features (*Image from (Pokorny et al., 2014).*)

observation since the it has a single hole.

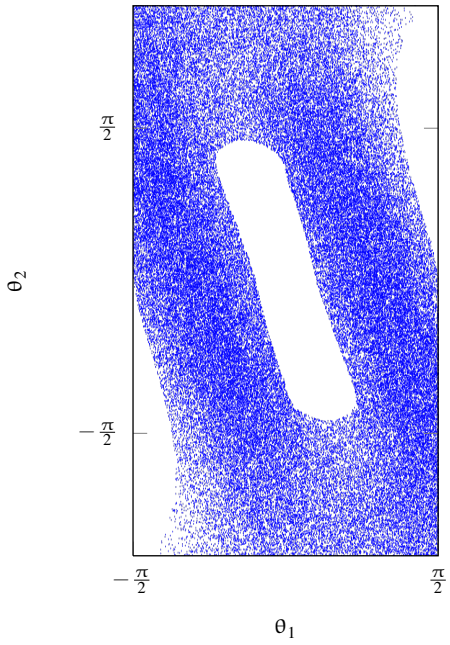


Figure 4.9: Projection of collision free samples onto θ_1, θ_2 (*Image from (Pokorny et al., 2014).*)

We computed 1000 edge-paths in $\mathrm{DC}_{0.25}$ between the start and end-configuration using random waypoints. For filtrations $0.301 \leqslant \varepsilon \leqslant 0.382$ only two path classes existed among the paths. The reduction of the boundary matrix took 0.46s, while the persistent

cycle features for all 1000 paths were calculated in 0.55s. The Delaunay triangulation in $\mathbb{R}^4$ took 251s, partially due to the increased dimension[1].

We inspected the trajectory in each of the two homology classes and found that the paths were classified according to whether the second link was positioned to the right (like the letter 'Z') or to the left (like the letter 'S') of the base link of the arm as the arm passes the narrow passage (see Fig. 4.10). Our framework hence allows the robot to *discover* the fact that two fundamentally different trajectory classes exist which cannot be deformed into each other.

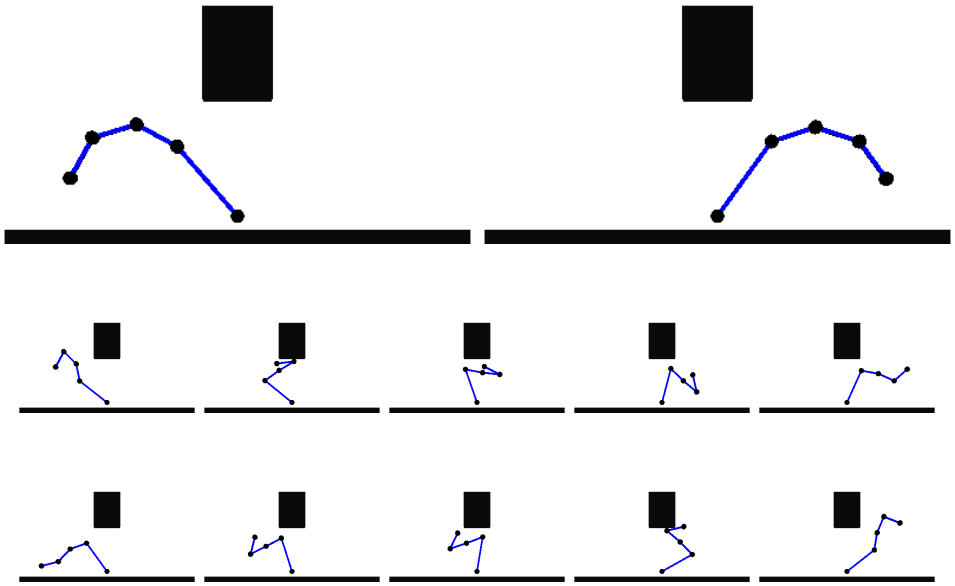


Figure 4.10: An illustration of the difference between the found topological classes. Top, the task is to move from the configuration on the left to the configuration on the right. Below, the two discovered classes of motions, class 'Z' and class 'S'.

4.4.3 Multiscale path classification for a physical robot system

We evaluate the framework on a set of trajectories generated by the end-effector of a Baxter robot (Rethink Robotics, 2012). The robot executes a specific reaching task between two points using one or both arms, *e.g.* by kinesthetic demonstration. Thus, each point in a trajectory is a 3-dimensional point (x, y, z) of the end effector coordinates for one arm, or a 6-dimensional point $(x_r, y_r, z_r, x_l, y_l, z_l)$ for the right and the left end effector position, respectively, if both are used.

[1] Not directly comparable to the 2D case, since we did not employ spatial sorting here.

(a) The skill using the right arm. (b) The skill using the left arm.

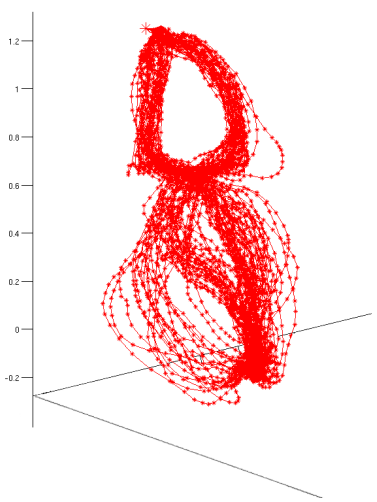
Figure 4.11: Setup of the Baxter experiment 1. The robot demonstrates the reaching skill using both arms. The movement starts above the head and ends in the bottom.

Experiment 1

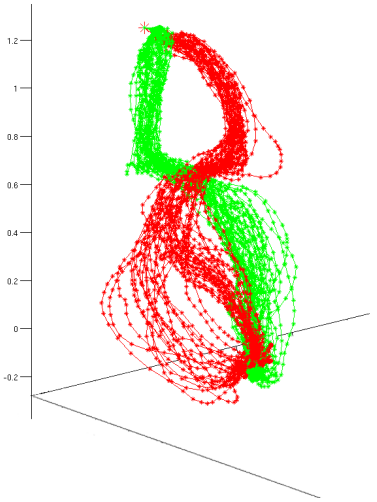
In this experiment, each of the two arms is used to reach from a point above the head of the robot to a point below its torso as in Fig. 4.11. The complete set of 96 demonstrated trajectories of the two motions are fed into the homology classification algorithm, with the results of classification shown in Fig. 4.12. The filtration values 0.5m, 0.15m, 0.075m, 0.060m resulted in a number of 1,2,4 and 8 classes, respectively.

The power of handling the filtration parameter inside the framework is that we do not have to select a setting for it beforehand, but can investigate the change in classification as its value changes. For example, the separation of the left arm trajectories from the rest is evident as it persists across a wide range of filtrations, while the emergence of the green class in Fig. 4.12c is of limited importance as it disappears quickly, making it most probably an outcome of noise in the demonstrated data. In general, the choice of classes can happen after examining the persistence of the features in the demonstrated data.

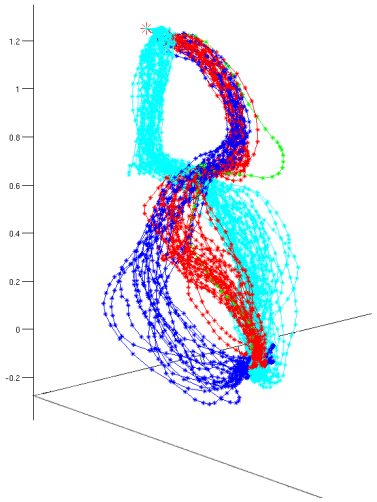
Comparison with k-means For the sake of contrast, we show the results of clustering using a k-*means* implementation using the same input trajectories. The domain of clustering is a g-dimensional space, where g is a positive integer, made by reading-off g equally-distant points from time-normalised trajectories. To tune the parameters g



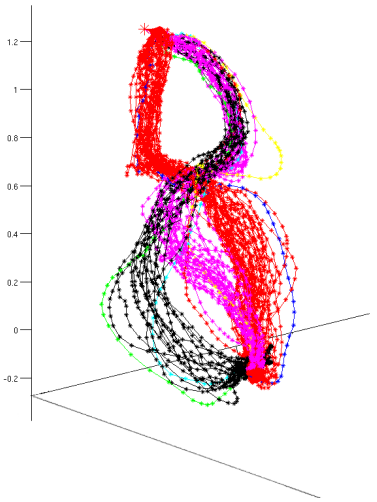
(a) filtration value $\varepsilon = 0.5\mathrm{m}$, 1 class



(b) filtration $\varepsilon = 0.15\mathrm{m}$, 2 classes

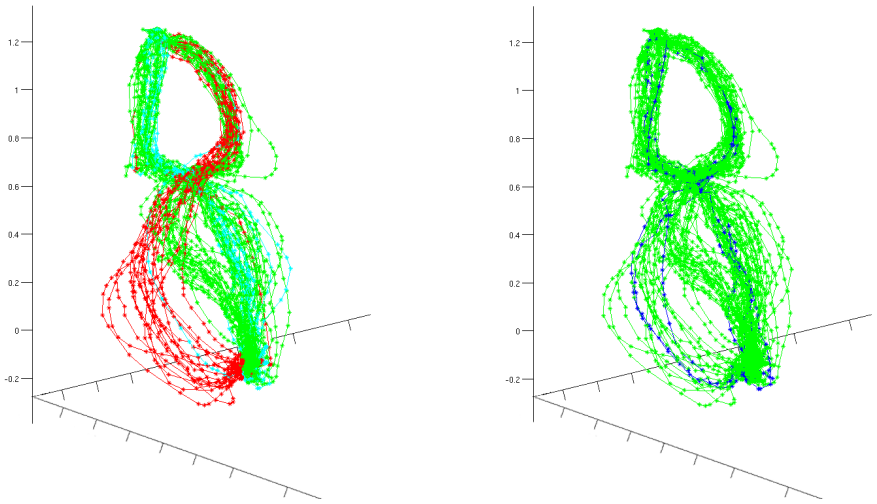


(c) filtration $\varepsilon = 0.075\mathrm{m}$, 4 classes



(d) filtration $\varepsilon = 0.06\mathrm{m}$, 8 classes

Figure 4.12: Classification of trajectories under different filtration values in the Baxter experiment 1. Each colour represents a different equivalence class. The trajectories show end-effector position for the Baxter arm performing a reaching skill. The filtration values are $0.5\mathrm{m}$ (a), $0.15\mathrm{m}$ (b), $0.075\mathrm{m}$ (c) and $0.06\mathrm{m}$ (d), and the number of discovered classes are $1, 2, 4$ and 8, respectively.



(a) hand-picked parameter setting, g=9, k=4.

(b) Automatic parameter setting.

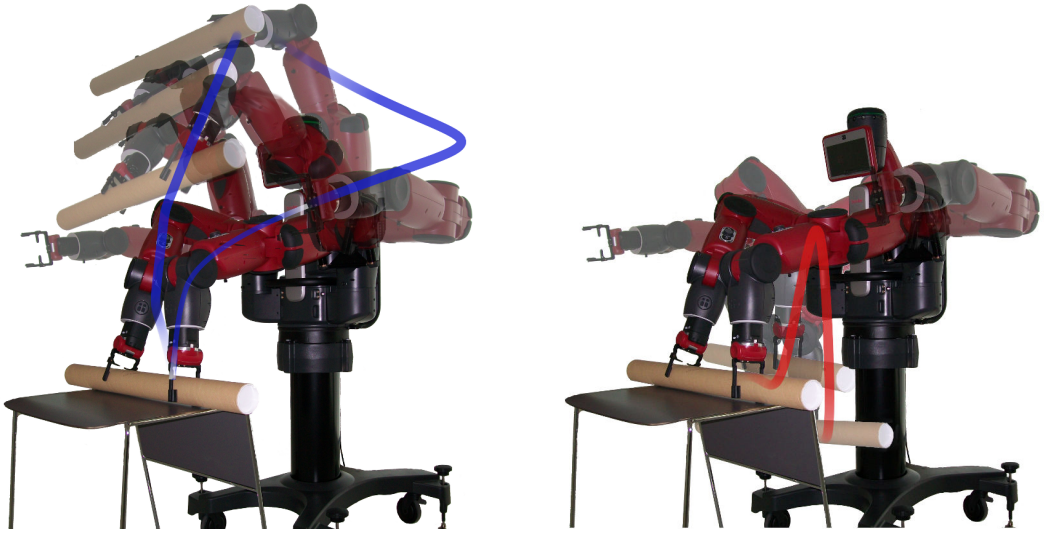
Figure 4.13: Classification of trajectories using k-means algorithm. Each colour represents a different cluster. The parameters of the plots are manually-set (left); and automatically chosen to minimise the number of label switches in an ordered input set (right).

and k, we search for the setting that minimises the number of switches of cluster labels in an ordered trajectory set, ordered by class. The results in Fig. 4.13 shows, in addition to that automatically-chosen setting, a hand-picked setting also. These parameters were chosen knowing the complexity of the trajectories and the expected number of classes. As it is clear from the figure, both settings do not manage to distinguish the classes cleanly. That is, setting the parameters even for such a simplified experiment is not straightforward task.

Experiment 2

In the second experiment (Fig. 4.14), we demonstrate a two-arm periodic skill in which the robot picks a cylindrical object on a nearby table and places it either above its head on a hypothetical shelf, or onto the ground next to the robot base, before returning to the starting pose. Also, different trajectories have different distances between the end effectors when grasping the object.

The result of classification is shown in Fig. 4.15 for a fixed filtration value $\varepsilon = 0.19\text{m}$. Each 6D point is depicted as two 3D points, one per arm. The two colours represent two different equivalence classes, which correspond to the two different input motions.



(a) Placing the object up on a hypothetical shelf.

(b) Placing the object on the ground.

Figure 4.14: Setup of the Baxter experiment 2. The robot uses both arms to place an object in two different destinations before returning to the starting pose.

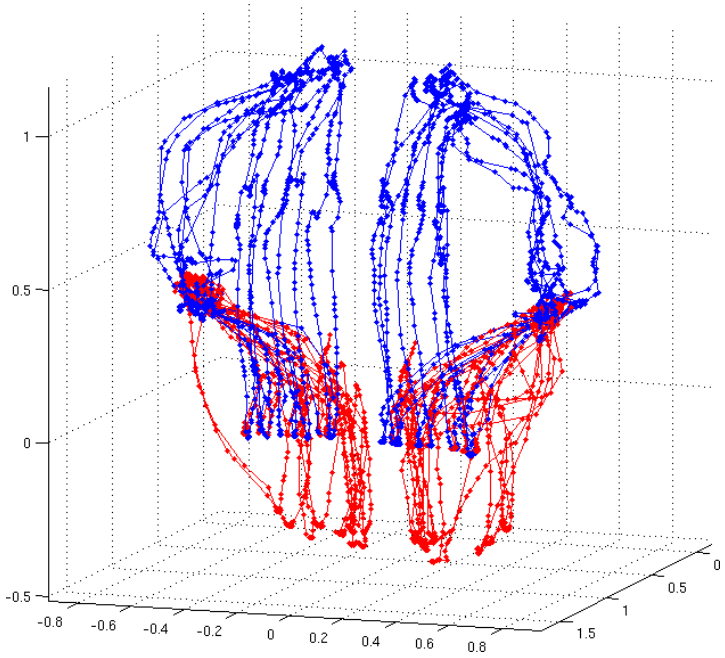


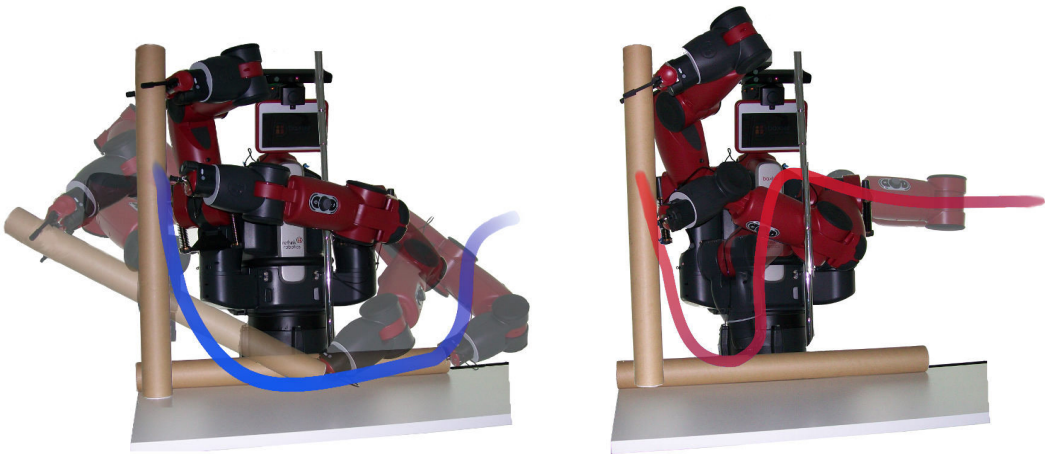
Figure 4.15: Classification results in the Baxter experiment 2 for both arms. Each 6D points is depicted as two 3D points, one per arm. Each colour represents a different equivalence class.

Experiment 3

In this experiment (Fig. 4.16), an obstacle (a metal bar) is placed in front of the robot to create extra structure in the free configuration space. The task of the robot is to pick a

cylindrical object lying horizontally in front of the obstacle and move it to a vertical configuration. The robot does that either by manoeuvrings of the left arm in front of, or behind, the obstacle.

The collection of configurations that collide with the obstacle are removed from the configuration space, creating a hole in its topological description, see Fig. 4.17 where only the trajectories of the left arm are shown, as the right arm performs the same behaviour in the input demonstrations.. The classification is performed for a fixed filtration value $\varepsilon = 0.19\mathrm{m}$. The two colours represent the two different equivalence classes, which correspond to the two different manoeuvres.



(a) Moving in front of the obstacle. (b) Moving behind the obstacle.

Figure 4.16: Setup of the Baxter experiment 3. The robot negotiates an obstacle in its environment (the metallic post) in two different ways in order to reorient a cylindrical object from horizontal to vertical position.

4.4.4 Option extraction for a physical robot system

Starting from a set of demonstrations, we would like to generate a set of options (motion primitives) that summarises the variability in the ways the task can be solved in. We fix the filtration value to $\varepsilon = 0.075\mathrm{m}$ in the first experiment (Fig. 4.11), and we use the same set of demonstrations to generate options that solve that task.

We run ILPSS with Topological Options on the resulting dataset, and we show the outcome in Fig. 4.18. In the figure, each colour represents the support of an option, while the option policy can be constructed to follow the flow of the trajectories until the support is left. Note that an option is segmented out for both being a member of a

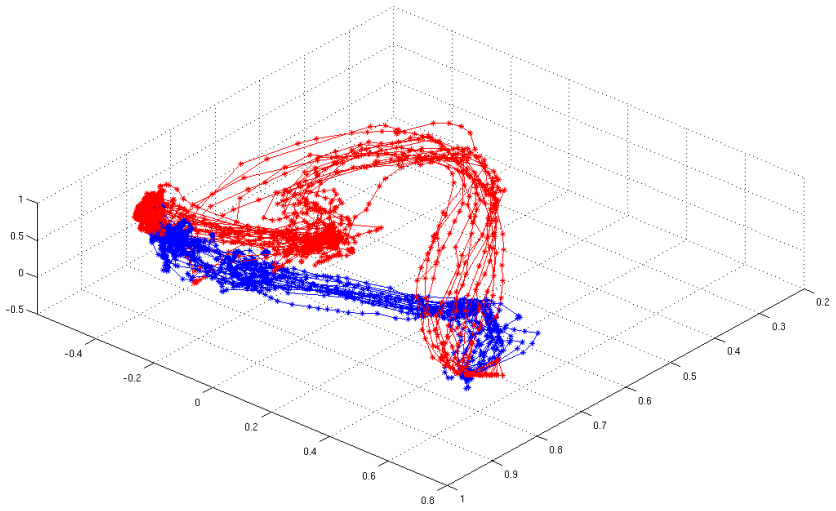


Figure 4.17: Classification results in the Baxter experiment 3 for the left arm only. Each colour is a different equivalence class. The trajectories for the right arm are very similar in the two skills, and hence they are not shown.

specific topological equivalence class, and for the importance of its support, quantified by the visit frequency of different demonstrations.

Thus, what is captured is a set of valid options/ motion primitives that were useful in the current task. Note that these options were defined in a 'snapshot' of the task and the environment. For a lifelong learning agent, the options can be refined and made more specific with more experience in the same, or new, learning tasks.

4.5 Concluding Remarks

In this chapter, we have proposed a novel *sampling-based* approach to studying homology classes of paths in robot configuration spaces of arbitrary dimension, with the aim of defining topologically-distinct motion primitives as a kind of policy abstraction. We used a novel construction of a filtration of simplicial complexes and the tools of persistent homology to achieve a classification of a set of input trajectories into equivalence classes to select suitable representative behaviours.

This opens new possibilities for integrating *local* optimisation based planning algorithms with the proposed method which extracts *global* information from the configuration space. The integrating of cost landscapes into the framework allows applications of our method in optimal class-dependent generation of dynamic motion primitives. Furthermore, we have shown how the proposed method can be used to define

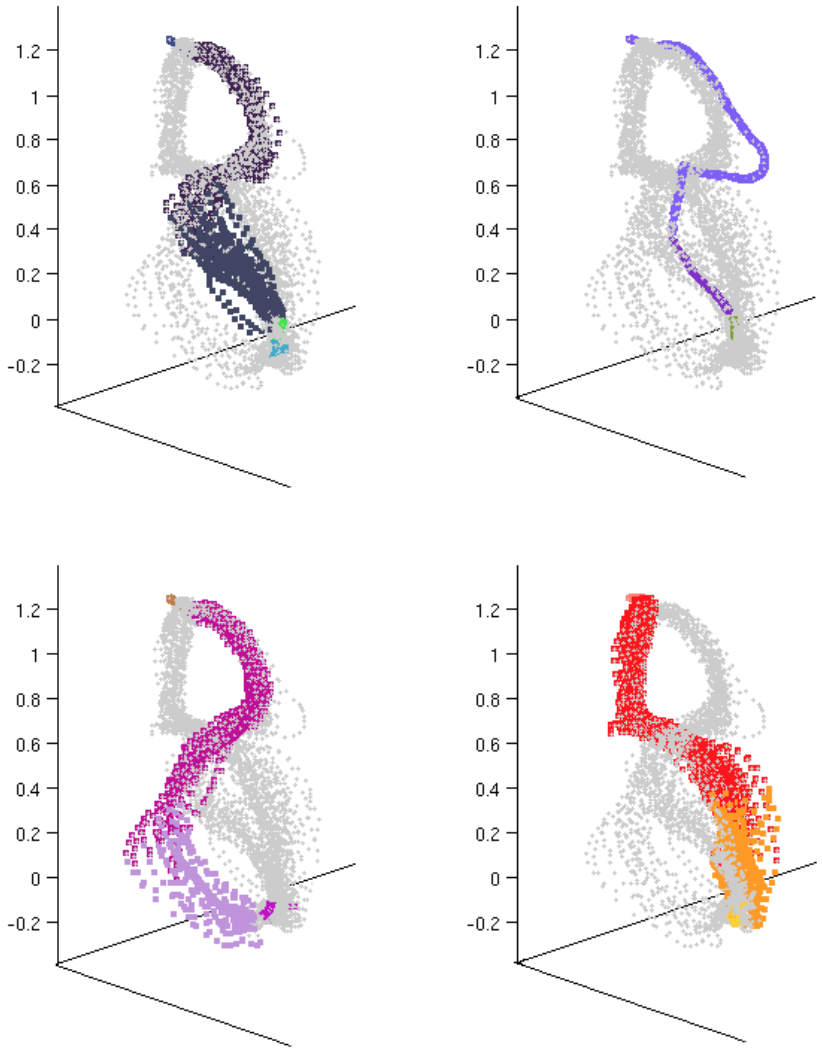


Figure 4.18: Options derived using ILPSS with Topological Options. The 3D data points represent sampled poses for the end effector of a Baxter robot performing a reaching skill. The trajectories are classified with a filtration value of $\varepsilon = 0.075\mathrm{m}$, giving the four classes shown in colour in the four panels. After that, options are acquired using ILPSS. The different colours in each panel represent different options.

a small set of options which topologically summarises the variability of the behaviours in the domain. We believe that using topology as a metric to gauge the usefulness of an abstraction is a promising approach to autonomous learning for a lifelong learning agent particularly, and to abstraction in general.

Chapter 5

Policy Space Abstraction for Policy Reuse

5.1 Introduction

In this chapter, we explore policy space abstraction using models of task space and its relation to policy space. A direct mapping between the two spaces would be useful for transfer, as solutions for new tasks could be promptly computed through that mapping, without the need to learn.

In general, modelling the task/policy space requires a measure of similarity between tasks/policies. One example of that would be the metric in the space of parameters for some parametrisable task/policy. In that case, two tasks/policies with similar parameters are similar. The direct mapping for this scenario can be a function that relates the parameters of the task to those of the policy (*e.g.*, (Silva et al., 2012)). This approach, in many interesting practical situations, is not relevant either because the policies are not parametrisable in the first place, or because the parameters of the task are not (immediately) observable, which require a more careful treatment.

A more general approach uses the return of some behavioural policies on a set of tasks to define a similarity measure, where similar tasks exhibit similar performance under the set of policies (*e.g.* (Mahmud et al., 2013)).

In this chapter we propose a model that selects a policy from a general library of policies in response to a new task (Sec. 5.2). This problem is called *Policy Reuse*. The selection is achieved through a Bayesian framework that elicits information about the task in an optimal exploration process (Sec. 5.4), using *signals* that are generated by the task and are correlated to its type (Sec. 5.3.4). Not knowing the policy space parameters is handled through building a non-parametric model of the tasks in an offline phase (Sec. 5.3).

Fig. 5.1 gives an overview of the approach in this chapter.

5.1.1 Contributions

The main contributions of this chapter are:

- We introduce Bayesian Policy Reuse (BPR) as a general Bayesian framework for solving the policy reuse problem.
- We present several specific instantiations of BPR using different policy selection mechanisms and compare them on a domain modelling a surveillance problem.
- We provide an empirical analysis of the components of our model, considering

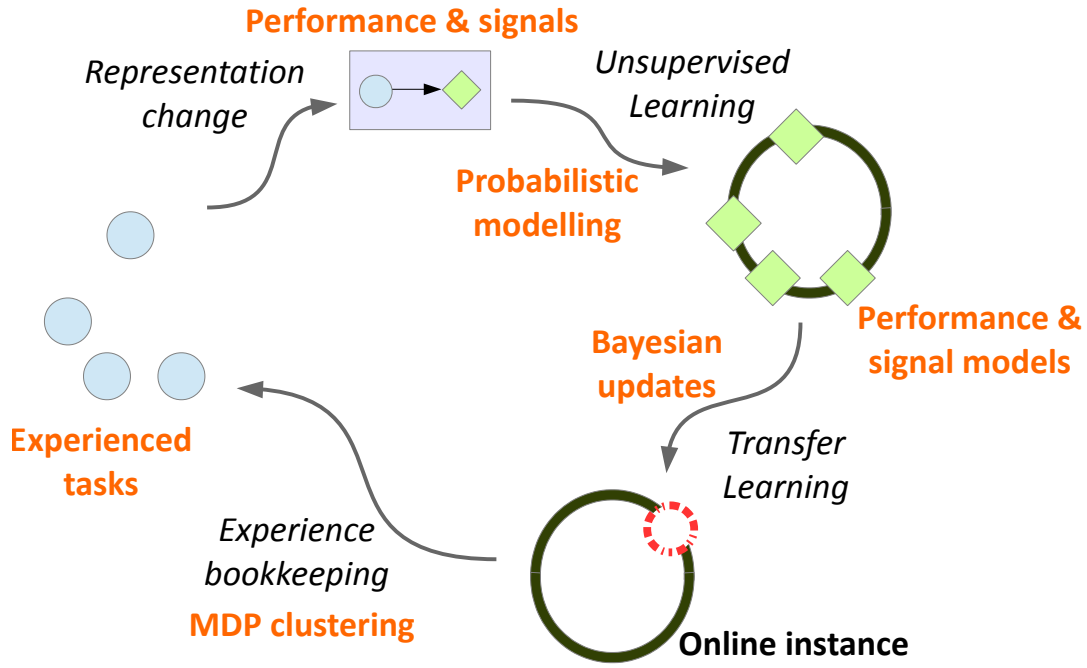


Figure 5.1: Overview of the approach in this chapter. The set of experienced tasks are abstracted using probabilistic models of performance and signals that can be observed online and which are correlated to the type of the task. Using a Bayesian update, the type of a new task can be identified from the observation signals, and a proper policy can be chosen from the previously-learnt policies to maximise performance. The tasks can be automatically clustered into representative types to reduce the storage and runtime complexities.

different classes of observation signal and the trade-off between library size and sample complexity.

5.2 Bayesian Policy Reuse (BPR)

5.2.1 Policy Reuse

The problem space considered here is characterised as a *short-term*, *interactive* adaptation to a new *situation*.

Consider *online personalisation*, which is becoming a core concept in human-computer interaction, driven largely by a proliferation of new sensors and input devices which allow for a more natural means of communicating with hardware. Consider, for

example an interactive interface which interprets the gestures of a human user, set up in a public space, such as a museum or a store. The device is required to map human movement to interpretable instructions, *e.g.*, to provide the user with information. The difficulty is that the device may be expected to interact with a wide and diverse pool of users, who differ both at the low level of interaction speeds and body sizes, and at the higher level of which gestures seem appropriate for particular commands. The device should autonomously, and quickly, calibrate to the user, where a mismatch could result in a failed interaction and lengthy calibration is likely to frustrate the user and make him/her abandon the interaction.

Similar problems appear in other circumstances, as, for example, in the case of an intelligent base station for monitoring poachers in a large wildlife reserve, using light-weight drones that can scan particular locations in the park for unusual activity. The number of drone deployments in any problem instance would be limited, as the poachers can be expected to spend a limited time stalking their target before leaving.

5.2.2 Problem statement

The key component is the need for sample-efficient decision making, as the agent is required to respond to scenarios before they have elapsed. On the other hand, the interaction is inherently short, so the agent needs to perform well in the limited time available. Then, we need solution methods that have both *low sample complexity* and *low regret* in a large space of possible tasks.

It is unreasonable to assume that any new task instance could be learnt from scratch in the constrained interaction time. More plausibly, seeding the process with a set of policies of solved instances, as a strategy for transfer learning, may be beneficial. For example, the interactive interface may ship with a set of different classes of user profiles which have been acquired offline, while the monitoring system may have a collection of pre-learnt behaviours to navigate the park when a warning is issued.

One way to build the required library of policies is described in (Mahmud et al., 2013). There, a set of experienced MDPs are clustered with respect to a distance function that is related to how the optimal policy of one MDP fares when applied to another MDP, and vice versa. Each cluster of MDPs represents a neighbourhood in performance space, so that one does not lose more than a specific amount of performance by applying the policy of a cluster's *landmark MDP* in place of the optimal policy of one of its members. The set of landmark policies of clustering experienced tasks is a

good choice for a policy library.

We term this problem of short-lived sequential policy selection for a new instance the *policy reuse* problem and we define it formally as follows. Let an agent be a decision making entity in a specific domain and let it be equipped with a policy library Π for tasks in that domain. The agent is presented with an unknown task which must be solved within a limited, and small, number of trials. At the beginning of each trial episode, the agent can select one policy from Π to execute for the full episode. The goal of the agent is thus to *select from existing policies to minimise the total regret incurred in the limited task duration with respect to the performance of the best alternative in* Π *in hindsight.*

5.2.3 Related work

Online choice from a set of alternatives for a minimal regret could be posed as a multi-armed bandit. In this framework, each arm would correspond to a pre-learnt policy, and our problem becomes a sequential optimal selection of fixed policies. In general, finite-horizon online bandit problems are intractable (Niño-Mora, 2011). Furthermore, traditional bandit approaches have to try each available arm on the new task in order to gauge its performance, which may be a very costly procedure from a sample complexity point of view, assuming a large number of policies.

Instead, one can exploit knowledge which has been acquired offline to accelerate online response times, as in transfer learning (Taylor and Stone, 2009). We propose a solution for policy reuse that exploits the natural correlation between policies, captured by testing them under canonical situations.

A version of the policy reuse problem appears in (Mahmud et al., 2013), where it is used to test a set of landmark policies retrieved through clustering of tasks in the space of MDPs. In another thread, the term 'policy reuse' was used by (Fernández and Veloso, 2006) in a different context. There, a learning agent is equipped with a library of previous policies to aid in exploration, as they enable the agent to collect relevant information quickly to accelerate learning. In our case, learning is infeasible, and policy reuse is the only way to achieve the objective of the agent.

We now pose the policy reuse transfer problem in a Bayesian framework.

5.2.4 Setting

For a set of tasks $\mathcal{X}$ and a set of policies Π, Bayesian policy reuse involves two key components.

- The *performance model*, $P(u|x,\pi)$, where $u \in \mathbb{R}$ is utility, $x \in \mathcal{X}$ is a task, and $\pi \in \Pi$ is a policy: a probability model over performance of the set of policies Π on the set of seen tasks $\mathcal{X}$. This information is acquired in an offline phase.

- The *observation model*, defined as a probability distribution $P(\sigma|x,\pi)$, where $\sigma \in \Sigma$ a signal from the space of possible observation signals which can be provided by the environment, $x \in \mathcal{X}$ is a task, and $\pi \in \Pi$ is a policy. A signal is some kind of information that is correlated to the performance and which can be observed online. If the performance information is directly observable, the observation and the performance models may be essentially the same.

Given a new task $x^* \in \mathcal{X}$, the agent is required to select the best policy $\pi^* \in \Pi$ in as few trials as possible, whilst accumulating as little regret as possible in the interim. The agent has prior knowledge only in the form of performance models for each policy in Π on a set of tasks from $\mathcal{X}$, as well as observation models.

Having an observation model is useful in that using a known policy on a new task would provide a sample observational *signal*. This can be used to update a 'similarity' measure over the seen tasks. This similarity allows for selection of a policy at the next time step to optimise expected performance. This is the core idea behind Bayesian policy reuse.

A caricature of the Bayesian policy reuse problem is in Fig. 5.2.

5.2.5 Algorithm

We present the general form of Bayesian Policy Reuse (BPR) in Algorithm 5. The algorithm maintains a probabilistic distribution, called the *belief* β, over the set of seen tasks as a representation of the similarity measure of the tasks to the new task instance. The procedure is a sequential Bayesian update of the similarity belief using the signals of interaction.

Possible implementations of step 3 are to be discussed in Sec. 5.4.

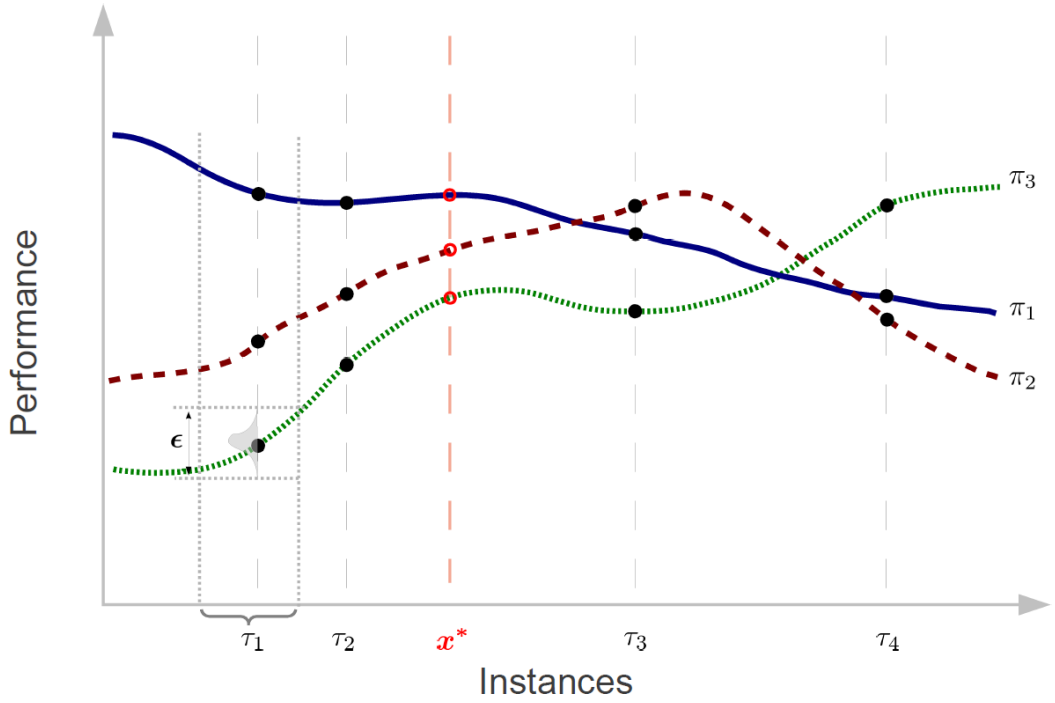


Figure 5.2: A simplified depiction of the Bayesian Policy Reuse problem. The agent has access to a library of policies (π_1, π_2 and π_3 in the figure), and has previously experienced a set of task instances ($\tau_1 \dots \tau_4$) from a continuum of tasks, and has sampled the utilities of the library policies on these instances (the dots on the curves are the means of these estimates, while the agent maintains distributions of the utility as illustrated by $P(u|\tau_1, \pi_3)$). The agent is presented with a new unknown task instance (x^*), and it is asked to select the best policy from the library (optimising between the red points). Note that it is infeasible to try every individual option (in less than 3 trials in this example), the agent has no knowledge about the complete curves, neither where the task instances occur in the problem space. The agent needs to infer this from utility similarity. The observation models are not depicted.

5.2.6 Regret

In order to evaluate the performance of our approach, we define *regret* as the policy selection metric to be optimised by Bayesian Policy Reuse.

Definition 3 (Library Regret)**.** For a library of policies Π and for a policy selection algorithm $\xi : \mathcal{X} \to \Pi$ that selects the policy $\xi(x^*) \in \Pi$ for the new task instance $x^* \in \mathcal{X}$, the *library regret* of ξ is defined as

$$\mathcal{R}(\xi) = u_{x^*}^{\pi^*} - u_{x^*}^{\xi(x^*)},$$

Algorithm 5 Bayesian Policy Reuse (BPR)

Require: Problem space $\mathcal{X}$, Policy library Π, observation space Σ, prior over the problem space $P(\mathcal{X})$, observation model $P(\sigma|x,\pi)$, performance model $P(u|x,\pi)$, number of episodes K.

1: Initialise beliefs: $\beta^0(x) \longleftarrow P(x)$ for all $x \in \mathcal{X}$
2: **for** episodes $t = 1 \dots K$ **do**
3: Select a policy $\pi^t \in \Pi$, using the current belief of the task β^{t-1}.
4: Apply π^t on the task instance.
5: Obtain an observation signal σ^t from the environment.
6: Update the belief $\beta^t(x) \propto P(\sigma^t|x,\pi^t)\beta^{t-1}(x)$, for all $x \in \mathcal{X}$.
7: **end for**

where u_x^π is the performance of using policy π on the task x; $\pi^* = \arg\max_{\pi\in\Pi} u_{x^*}^\pi$, the best policy in hindsight *in the library* for the task instance x^*.

Definition 4 (Average Library Regret)**.** For a library of policies Π and for a policy selection algorithm $\xi : \mathcal{X} \to \Pi$, the *average library regret* of ξ over K trials is defined as the average of the library regrets for the individual trials,

$$\mathcal{R}^K(\xi) \;=\; \frac{1}{K}\sum_{t=1}^{K} \mathcal{R}(\xi^t),$$

where $\mathcal{R}(\xi^t)$ is the regret for the policy selection at trial t.

The metric we choose to minimise in BPR is the average library regret $\mathcal{R}^K(\xi)$ for K trials. Thus, the goal of BPR is to not only find the right solution at the end of the K trials, possibly through expensive exploration endeavours, but also to optimise performance even when exploring in the short duration of the task.

We will refer to this metric simply as 'regret' throughout the rest of the chapter.

5.3 Problem Space, Observation Signals and Beliefs

5.3.1 Tasks

Let a task be specified by a Markov Decision Process (MDP). Denote the space of all MDPs $\mathcal{M}$. We will consider episodic tasks, *i.e.* tasks that have a bounded time

horizon. The return, or utility, generated from running the policy π on an episodic task, $u^{\pi} = E[\sum_{i=0}^{k} r_i | \pi]$, with k being the length of the episode, and r_i being the reward realised at time t. We denote a collection of policies that the agent has by Π, and refer to it as the policy library.

5.3.2 Types

When working directly in the task space $\mathcal{M}$, it is very likely that maintaining and updating the belief of BPR would require a large number of samples. In many scenarios, we may find that there is a natural notion of clustering in $\mathcal{M}$, where many tasks are similar with minor variations. Previous work has looked into finding the clustering in a space of tasks; see (Mahmud et al., 2013) for a detailed account. We do not try to discover the clustering in this work, but we assume it exists.

To this end, we propose a simple notion of task *types* as clusters of tasks that form ϵ-balls in performance space, for some $\epsilon \in \mathbb{R}$, where performance is with respect to a collection of designated policies. Other ways to define types exist, as the concept in (Mahmud et al., 2013) or the notion of task classes as probability distributions over task parameters in (Wilson et al., 2007).

Definition 5 (Type)**.** A type τ is a subset of tasks such that for any two MDPs m_i, m_j from a type τ and for all policies π in a set of designated policies Π, the difference in utility is upper-bounded by some $\epsilon \in \mathbb{R}$:

$$m_i, m_j \in \tau \;\; \Leftrightarrow \;\; |u_i^{\pi} - u_j^{\pi}| \leqslant \epsilon, \;\; \forall \pi \in \Pi,$$

where $u_i^{\pi} \in \mathbb{R}$ is the utility from executing policy π on task m_i. Then, m_i and m_j are ϵ-*equivalent* under the policies Π.

In Fig. 5.2, the ϵ-region in performance space for τ_1 is explicitly depicted with the corresponding region in the task space. We assume the existence of a finite number of task types in a domain, and we denote the space of types with $\mathcal{T}$. Assuming these types are disjoint in the domain[1], we can work with $\mathcal{T}$ as the problem space of BPR. This induces a hierarchical structure in the space $\mathcal{M}$. A type τ is drawn from a hyperprior $\tau \sim G_0$, and then a task is drawn from that type $m \sim \Delta^{\tau}(m)$, where $\Delta^{\tau}(m)$ is a probability distribution over tasks.

[1]Types do not have to be disjoint in general for our framework to work, *i.e.* there may exist tasks that belong to multiple types at once. However, this requires that the correlation between the types be explicitly captured.

By definition, the set of MDPs generated by a single type are ϵ-equivalent under Π. Hence, regret of replacing all the MDPs in τ with any one of them cannot exceed ϵ. Let us choose a single MDP per type and call it a *landmark MDP* of τ, and denote it m_τ. This would simplify the hierarchical structure, where the prior G_0 acts immediately on the set of landmark MDPs $m_\tau, \forall \tau \in \mathcal{T}$. See Fig. 5.3.

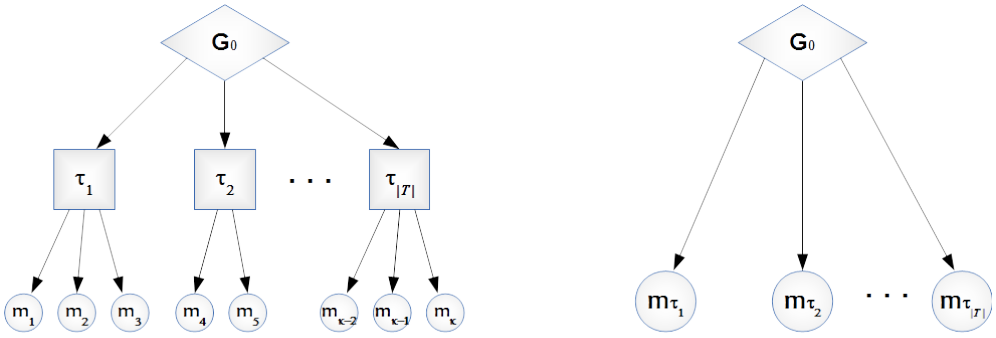


Figure 5.3: Problem space model with disjoint types. Left, tasks m are related by types τ, with a generating distribution G_0 over them. Right, a simplification of the hierarchical structure under ϵ-equivalence. The tasks of each type are represented by a single landmark task m_τ.

The benefit of this for Bayesian Policy Reuse is that each alternative now in the task space abstraction is a representative for a region in the original task space, defined by ϵ.

We will use the type space $\mathcal{T}$ as the problem space from here on, although we note that the methods we propose do not change if the full task space $\mathcal{M}$ was used as the problem space.

5.3.3 Performance model

One of the key components of BPR is the performance model of policies for seen task instances, which holds a distribution of returns. Using types, a performance model represents the variability in return for the various tasks in a type.

Definition 6 (Performance Model)**.** For a policy π and a type τ, the *performance model* $P(u|\tau,\pi)$ is a probability distribution over the utility of π when applied to all tasks $m \in \tau$.

5.3.4 Signals

Definition 7 (Signal)**.** A *signal* σ is any information that is correlated to the performance of a policy and which is available to the agent in an online execution of the policy. We call the space of observations Σ.

The goal of the signal is to allow the agent to identify the type of the task instance so that a proper response is chosen and reused. The most straightforward signal is the performance itself, but in some situations this is not always available (*e.g.* in cases where the payoff is delayed). The information content and richness of a signal determines how easily an agent can identify the type of the new task with respect to the seen types. The agent learns a model of how types, policies and signals relate and interact offline to be used in type identification online.

5.3.5 Observation model

For some choice of signal space Σ, the agent learns canonical models of the signal values expected from every type-policy pair.

Definition 8 (Observation Model)**.** For a policy π and type τ and a observation space Σ, the *observation model*, $\mathcal{F}_\pi^\tau(\sigma) = \mathrm{P}(\sigma|\tau,\pi)$, is a probability distribution over the signals $\sigma \in \Sigma$ that may result by applying π to τ.

We assume that the agent acquires observation models offline. Consider the following procedure to learn the signal models for the policy library Π:

1. The type label τ is announced.
2. A set of tasks are generated from the type τ.
3. The agent runs all the policies from the library Π on all the instances of τ, and observes the resultant signals $\sigma^{\tau,\pi}$.
4. Distributions $\mathcal{F}_\pi^\tau = \Delta(\sigma^{\tau,\pi})$ are fitted to the data.

These models relate observable signals to the latent type label of seen types. For a new instance, identifying the true type (or the most similar seen one) is sufficient to play a good policy from the policy library, which is the goal of the policy reuse problem.

5.3.6 Example signals

Here are some candidate signals that may be used in a BPR implementation.

State-Action-State tuples

The richest information signal which could be accrued by the agent is the history of all (s, a, s') tuples encountered during the execution of a policy. Thus, the observation model in this case is exactly the expected transition function of the MDPs of the type τ, or the part of that model that has been experienced.

The expressiveness of this signal comes with a drawback, that it might be expensive to learn and maintain such models for every possible type.

Instantaneous rewards

A similar form of information is the instantaneous reward $r \in \mathbb{R}$ received during the execution of a policy for some state-action pair s, a. Then, the observation model is the expected reward function for the MDPs in the type. This may provide a relatively fine-grained knowledge on the behaviour of the task in cases where intermediate rewards are informative.

Episodic returns

An example of a sparser signal is the total utility $u_\tau^\pi \in \mathbb{R}$ accrued over the full episode of using a policy in a task. This signal is useful for problems of delayed reward, where intermediate states cannot be valued easily, but the extent to which the task was successfully completed defines the return. The observation model of such a scalar signal is much more compact, and thereby easier to learn and maintain, than the previous two proposals.

In our framework, using episodic returns as signals has the additional advantage that this information is already captured in the performance model. This relieves the agent from maintaining two separate models, as in this case $P(u|\tau,\pi) = \mathcal{F}_\pi^\tau(u)$, for all π and τ.

5.3.7 Bayesian belief over types

Definition 9 (Type Belief)**.** For a set of seen types $\mathcal{T}$ and a new task x^*, the *type belief* $\beta(\cdot)$ is a probability distribution over the types of $\mathcal{T}$ that measures to what extent x^* matches the types of $\mathcal{T}$ in their observation signals.

The type belief, or *belief* for short, is a surrogate measure of similarity in type space. It approximates where a new instance may be located in relation to the known types,

which act as bases of the unknown type space.

The belief is initialised with the prior probability over the type space (called G_0 in Fig. 5.3). Then, after each execution of a policy, the environment provides an observation signal to the agent. This is used to update beliefs (line 6 in Algorithm 5), computing the posterior probability over the task space $\beta^t(\cdot)$ using Bayes' rule:

$$\beta^t(\tau) = \frac{P(\sigma^t|\tau,\pi^t)\beta^{t-1}(\tau)}{\sum_{\tau'\in\mathcal{T}} P(\sigma^t|\tau',\pi^t)\beta^{t-1}(\tau')} \tag{5.3.1}$$

$$= \frac{\mathcal{F}^{\tau}_{\pi^t}(\sigma^t)\beta^{t-1}(\tau)}{\sum_{\tau'\in\mathcal{T}} \mathcal{F}^{\tau'}_{\pi^t}(\sigma^t)\beta^{t-1}(\tau')}, \quad \forall \tau \in \mathcal{T}, \tag{5.3.2}$$

where β^{t-1} is the belief at episode $t-1$, β^t is the belief at episode t, π^t is the policy played in episode t, and σ^t is the signal received thereafter. We will use β to refer to β^t where this is not ambiguous.

5.4 Policy Selection for BPR

Given the current type belief, the agent is required to choose a policy for the next episode for two concurrent purposes: to acquire useful information about the type of the task in hand, while avoiding additional regret. Thus, the trade-off between exploration and exploitation is at the core of policy selection. We wish to gain as much information about the task as possible, so as to choose policies optimally[2] in the near future, but at the same time minimise performance losses due to sub-optimal choices.

We can define the following MDP to describe the optimal policy selection process for a type τ^*:

- The states are the continuous belief states over type, $\beta \in [0,1]^{|\mathcal{T}|-1}$.
- The actions are the available policies $\pi \in \Pi$.
- The reward process is defined by the utility, $u \sim P(u|\tau^*,\pi)$.

Under this formulation, the value of a policy choice π is given by

$$Q(\beta^t,\pi,K-t) = \int_{u\in\mathbb{R}} P(u|\tau^*,\pi)\left(u + \max_{\pi'\in\Pi} Q(\beta^u,\pi',K-t-1)\right)du, \tag{5.4.1}$$

[2]Note that we call the best policy *in the library* for a specific task the 'optimal policy', as we are not considering problems where learning is feasible.

where β^u is the new belief at $t+1$ after incorporating the observation u, $K-t$ is the number of trials still to go, and $Q(\beta^t, \pi, 1) = \int_{u \in \mathbb{R}} P(u|\tau^*, \pi)\, u\, du$. Using this formulation, the optimal policy selection at some belief β becomes the greedy optimisation, $\pi^* = \arg\max_{\pi \in \Pi} Q(\beta, \pi)$.

Computing the Q-function in Equation (5.4.1) is not feasible for two reasons. Firstly, $P(u|\tau^*, \pi)$ is not known for the unknown type τ^*. One possible work-around is to approximate it with the expected performance under the belief β, $\hat{P}(u|\tau^*, \pi) = \sum_{\tau \in \mathcal{T}} \beta(\tau) P(u|\tau, \pi)$, but this averaging would be destructive to the details in the performance models. Secondly, even if we approximate the performance model, the state in Equation (5.4.1) is continuous and hence discretisation or function approximation is needed, which does not generalise easily to different problem settings.

5.4.1 Approximate methods for policy selection

The goal of the agent is to maximise utility over the full K episodes of the task. This corresponds to minimising the cumulative, rather than instantaneous, regret. A purely *greedy* policy selection mechanism would fail to take exploratory actions needed for the belief to converge to the closest type, and may result in the agent becoming trapped in a local maximum of the utility function. On the other hand, a purely exploratory policy selection mechanism could be designed to elicit the most possible information in expectation, but this would not make an effort to improve performance. We thus require a mechanism to explore as well as exploit; find a better policy to maximise asymptotic utility, and exploit the current estimates of which are good policies to maximise myopic utility.

We now propose several policy selection mechanisms for dealing with this problem approximately:

- A simple ϵ-greedy exploration, where with probability $1-\epsilon$ we choose the policy which maximises the expected utility under the belief β,

$$\begin{aligned} \hat{\pi} &= \arg\max_{\pi \in \Pi} \sum_{\tau \in \mathcal{T}} \beta(\tau) \int_{u \in \mathbb{R}} u\, P(u|\tau, \pi) du \\ &= \arg\max_{\pi \in \Pi} \sum_{\tau \in \mathcal{T}} \beta(\tau)\, E[u|\tau, \pi] \end{aligned}$$

  and with probability ϵ we choose a policy from the policy library uniformly at random, with ϵ becoming smaller every trial, until it becomes 0 at K. This additional random exploration component perturbs the belief from local minima.

- Sampling the belief β, which involves sampling a type from the belief $\hat{\tau} \sim \beta$, and selecting the best policy for that type,

$$\hat{\pi} = \arg\max_{\pi \in \Pi} \mathrm{E}[u|\hat{\tau}, \pi].$$

  In this case, exploration is achieved through the sampling process.

- Using *exploration heuristics* that estimate a value for each policy, so as to achieve a balance between exploitation and a limited degree of look-ahead, to approximate optimal exploration.

  This is the prevalent approach in Bayesian optimisation, where, instead of directly maximising utility, a surrogate function that takes into account both the expected utility and a notion of the utility variance is maximised (see, *e.g.* (Brochu et al., 2010)). Multiple proposals have been widely considered in the multi-armed bandit (MAB) literature, ranging from early examples like the Gittins index (Gittins and Jones, 1974) for infinite horizon problems, to more recent methods such as the knowledge gradient (Powell, 2010). For the finite-horizon total-reward MAB, (Lai and Robbins, 1978) show that index-based methods achieve optimal performance asymptotically.

5.4.2 Bayesian Policy Reuse with Exploration Heuristics

By incorporating the notion of an exploration heuristic that computes an index ν_π for a policy π into Algorithm 5, we obtain the proto-algorithm Bayesian Policy Reuse with Exploration Heuristics described in Algorithm 6.

Note that we are now using G_0, the hyper-prior, as the prior in step 1 because we are using $\mathcal{T}$ as the problem space.

Next, we discuss some candidate heuristics $\mathcal{V}$ that can be used in line 3 in the algorithm, and we define four variants of the BPR-EH algorithm, as

- BPR-PI using probability of improvement,
- BPR-EI using expected improvement,
- BPR-BE using belief entropy,
- BPR-KG using knowledge gradient.

Algorithm 6 Bayesian Policy Reuse with Exploration Heuristics (BPR-EH)

Require: Type space $\mathcal{T}$, Policy library Π, observation space Σ, prior over the type space G_0, observation model $P(\sigma|\tau,\pi)$, performance model $P(u|\tau,\pi)$, number of episodes K, an exploration heuristic $\mathcal{V}$.

1: Initialise beliefs: $\beta^0 \longleftarrow G_0$.
2: **for** episodes $t = 1 \dots K$ **do**
3: Compute $\nu_\pi = \mathcal{V}(\pi, \beta^{t-1})$ for all $\pi \in \Pi$.
4: $\pi^t \longleftarrow \arg\max_{\pi\in\Pi} \nu_\pi$.
5: Apply π^t to the task instance.
6: Obtain the observation signal σ^t from the environment.
7: Update the belief β^t using σ^t by Equation (5.3.1).
8: **end for**

Probability of Improvement and Expected Improvement

One heuristic for policy selection can be the probability with which a specific policy can achieve a hypothesised increase in performance. Assume that $u^+ \in \mathbb{R}$ is some utility which is larger than the current best estimate under the current knowledge,

$$u^+ > u^t = \max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau) E[u|\tau,\pi].$$

The *probability of improvement* (PI) principle chooses the policy that maximises the term

$$\hat{\pi} = \arg\max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau) P(u^+|\tau,\pi).$$

The choice of u^+ is not straightforward, and this choice is the primary factor affecting the performance of this exploration principle. One approach to addressing this choice, is through the related idea of *expected improvement* (EI). This requires integration over all the possible values of improvement $u^t < u^+ < u^{max} \in \mathbb{R}$, then the policy with the best potential is chosen. That is,

$$\begin{aligned}
\hat{\pi} &= \arg\max_{\pi\in\Pi} \int_{u^t}^{u^{max}} \sum_{\tau\in\mathcal{T}} \beta(\tau) P(u^+|\tau,\pi) du^+ \\
&= \arg\max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau) \int_{u^t}^{u^{max}} P(u^+|\tau,\pi) du^+ \\
&= \arg\max_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau)(1 - F(u^t|\tau,\pi)) \\
&= \arg\min_{\pi\in\Pi} \sum_{\tau\in\mathcal{T}} \beta(\tau) F(u^t|\tau,\pi),
\end{aligned}$$

where $F(u|\tau,\pi) = \int_{-\infty}^{u} P(u|\tau,\pi)du$ is the cumulative distribution function of utility for π and τ.

Belief Entropy

Both PI and EI principles select a policy which has the potential to achieve higher utility. An alternative is to select the policy which will have the greatest effect in reducing the uncertainty over the type space.

The *belief entropy* (BE) approach seeks to estimate the effect of each policy in reducing uncertainty over type space, represented by the entropy of the belief. For each policy $\pi \in \Pi$, estimate the expected entropy of the belief after executing π as

$$H(\beta|\pi) \quad = \quad -\beta^{\pi} \log \beta^{\pi},$$

where β^{π} is the updated belief after incorporating the observation expected from running the policy π:

$$\begin{aligned} \beta^{\pi}(\tau) &= E_{\Sigma}\left[\frac{\mathcal{F}_{\pi}^{\tau}(\sigma)\beta(\tau)}{\sum_{\tau' \in \mathcal{T}} \mathcal{F}_{\pi}^{\tau'}(\sigma)\beta(\tau')}\right] \\ &= \int_{\sigma \in \Sigma} \mathcal{F}_{\pi}^{\tau}(\sigma)\left(\frac{\mathcal{F}_{\pi}^{\tau}(\sigma)\beta(\tau)}{\sum_{\tau' \in \mathcal{T}} \mathcal{F}_{\pi}^{\tau'}(\sigma)\beta(\tau')}\right) d\sigma. \end{aligned} \tag{5.4.2}$$

Then, selecting the policy

$$\hat{\pi} = \arg\min_{\pi \in \Pi} H(\beta|\pi)$$

which would reduce the most uncertainty in the belief in expectation.

This is in essence a purely exploratory policy, so to incorporate exploitation of the current state of knowledge, we rather select

$$\hat{\pi} = \arg\max_{\pi \in \Pi}\left(\sum_{\tau \in \mathcal{T}} \beta(\tau)\, E[u|\tau,\pi] - \kappa H(\beta|\pi)\right),$$

where $\kappa \in \mathbb{R}$ is a positive constant controlling the exploration-exploitation trade-off.

Knowledge Gradient

Another approach is to use the *knowledge gradient* (KG), which aims to balance exploration and exploitation through the optimisation of myopic return, whilst maintaining asymptotic optimality (Powell, 2010). The principle behind this approach is to estimate a one step look-ahead and select the policy which maximises utility over both the current time step, and the next in terms of the information gained.

To select a policy using the knowledge gradient, we choose the policy $\hat{\pi}$ which maximises the online knowledge gradient at time t

$$\hat{\pi} \quad = \quad \arg\max_{\pi \in \Pi} \left(\sum_{\tau \in \mathcal{T}} \beta(\tau)\, \mathrm{E}[u|\tau,\pi] + (K-t)\nu_{\pi}^{t} \right),$$

trading-off between the expected utility and ν_{π}^{t}, the offline knowledge gradient of π for a horizon of K trials, which essentially measures the performance of a one-step look-ahead in the process, given as

$$\nu_{\pi}^{t} \quad = \quad \mathrm{E}_{\beta} \left[\max_{\pi'} \sum_{\tau \in \mathcal{T}} \beta^{\pi}(\tau)\, \mathrm{E}[u|\tau,\pi'] - \max_{\pi''} \sum_{\tau \in \mathcal{T}} \beta(\tau)\, \mathrm{E}[u|\tau,\pi''] \right], \quad (5.4.3)$$

which is the difference in the expectation, with respect to β, of the best performance of any policy at $t+1$ if π was played at t, with that of the best policy at t, which may be different from π. β^{π} in the first term is the belief after incorporating the signal generated by π, as in Equation 5.4.2.

5.5 Experiments

While the first experiment shows a simplified scenario to demonstrate the approach, the second experiment highlights the different observation signals, and the third compares BPR to other methods in the literature.

5.5.1 Golf club selection

Consider the problem of a simulated robot golfer taking a shot on an unknown golf course, where it *cannot reliably estimate the distance to the hole*. The task is evaluated by how close to the hole the ball ends, and the robot is only allowed to take $K = 3$ shots from a fixed position from the hole, but it can choose any club from a set of 4 clubs (number of clubs $> K$) with the properties in Table 5.1 for some canonical stroke.

In fact, this simplified scenario is a single-shot, bandit-style, decision problem. Nonetheless, it fulfils the requirements of BPR, as a policy needs to be selected from the beginning and an evaluation signal is received in the end. BPR treats any sequential decision process as a one-shot decision process by abstracting acting away with policy selection from a fixed library.

- *Types*: the type space $\mathcal{T}$ is made of a set of previous shots, each defined by how far the target was (the agent could be told how far the hole was after the shot is

Club	Average Yardage	Standard Deviation of Yardage
$\pi_1 = 3$-wood	215	8.0
$\pi_2 = 3$-iron	180	7.2
$\pi_3 = 6$-iron	150	6.0
$\pi_4 = 9$-iron	115	4.4

Table 5.1: Statistics of the ranges (yardage) of the four clubs used in the golf club selection experiment. We choose to model the performance of each club by a Gaussian distribution with the shown parameters. We assume the robot is competent with each club, and so the standard deviation is small, but related to the distance.

performed). We assume the robot has experience with four previous holes, with distances $\tau_{110} = 110$ yds, $\tau_{150} = 150$ yds, $\tau_{170} = 170$ yds and $\tau_{220} = 220$ yds. The performance models are shown in Fig. 5.4.

- *Performance*: the metric to optimise is defined as the negative of the absolute distance from the final position of the ball to the hole, such that this quantity must be maximised.

- *Policies*: given the robot's canonical stroke, the policy is the choice of the club. For each club, the robot has a performance model of the final distances for all the previous instances.

- *Signals*: The robot cannot measure distances in the field, but for a feedback signal, it can crudely estimate a qualitative description of the result of a shot as falling into one of several categories (*e.g.*, *near* or *very far*) which is weaker than performance. The distributions over these qualitative categories (the observation models) are known to the agent for each club on each of the training types it has encountered.

When the robot faces a new hole, it overcomes its inability to judge the distance to the hole (unobservable task parameter) by using the qualitative feedback from a shot to identify the most similar previous task. This enables the robot to choose the club/clubs which would have been the best choice for the most similar previous task/tasks.

Consider, as an example, a hole 179 yards away. A coarse estimate of the distance can be incorporated as a prior over $\mathcal{T}$, otherwise an uniformed prior is used. Assume the robot is using greedy policy selection, and assume that it selects π_1 for the first shot

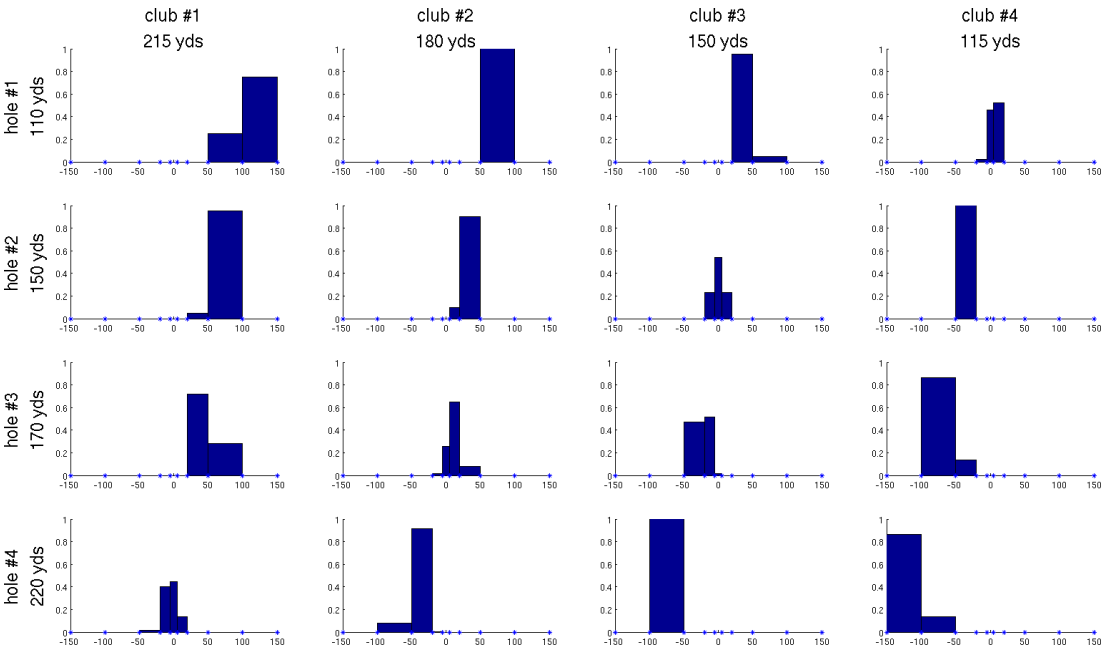


Figure 5.4: Performance models for the four clubs for four training holes with distances 110 yds, 150 yds, 170 yds and 220 yds. The signals are coarse distance category. The width of each category bin has been scaled to reflect the distance range it signifies. The x-axis is the distance to the hole, such that negative values indicate under-shooting, and positive distances over-shooting the hole.

due to a uniform prior and that this resulted in an over-shot by 35 yards. The robot cannot gauge this error more accurately than that it falls into the category *over-shooting* in the range of 20 to 50 yards. This signal will update the belief of the robot over the four types, and by Fig. 5.4, the closest type to produce such a behaviour for π_1 would be τ_{170}. The new belief and the greedy selection dictate that the best club to use for anything like τ_{170} is π_2. Using π_2, the hole is over-shot by 13 yards, corresponding to the category with the range 5 to 20 yards. With the same calculation, the most similar previous type is again τ_{170}, keeping the best club as π_2, and making the belief to converge. Indeed, given the ground truth in Table 5.1, this is the best choice for the 179 yard task. Table 5.2 describes this process over the course of 8 consecutive shots.

Fig. 5.5 shows the performance of BPR with greedy policy selection in the golf club selection task averaged over 100 unknown golf course holes, with ranges randomly selected between 120 and 220 yards. This shows that on average, by the second shot, the robot will have selected a club capable of bringing the ball within 10–15 yards of the hole. Considering that the current scenario is a multi-armed bandit decision problem,

Shot	1	2	3	4	5	6	7	8
Club	1	2	2	2	2	2	2	2
Error	35.37	13.16	4.28	6.78	2.07	11.05	8.15	2.45
Category	20–50	5–20	-5–5	5–20	-5–5	5–20	5–20	-5–5
β entropy	1.39	0.22	0.00	0.00	0.00	0.00	0.00	0.00

Table 5.2: The 179 yard example. For each of 8 consecutive shots: the choice of club, the true error in distance to the hole, the coarse category within which this error lies (the signal received by the agent), and the entropy of the belief. This shows convergence after the third shot, although the correct club was used from the second shot onwards. The oscillating error is a result of the variance in the club yardage. Although the task length was $K = 3$ strokes, we show these results for longer to illustrate convergence.

BPR better exploits the correlation between the arms, as explicitly captured by the performance and signal models, compared to standard bandit algorithms that do not.

5.5.2 Online personalisation

Consider a service offered over the phone, such as telephone banking, where the bank tries to improve the speed of answering telephonic queries by having a personalised model for understanding the speech of each user. In a traditional speech recognition system, the user may have time to train the system to her own voice, but this is not possible in this scenario. As a result, the phone service may have a number of different pre-trained language models and responses, and over the course of many interactions with the same user, try to estimate the best such model to use.

Let a user $\mathfrak{i}$ be defined by a preference in language $\lambda_\mathfrak{i} \in \{1,\dots,L\}$, where L is the number of such models. The policy π executed by the telephonic agent also corresponds to a choice of language model, *i.e.* $\pi \in \{1,\dots,L\}$. The goal of the agent is to identify the user preference $\lambda_\mathfrak{i}$, whilst minimising frustration to the user.

Assume that the transition system given in Fig. 5.6 describes the interaction. In every state, there is only one action which can be taken by the system, being to use the chosen language model. At the beginning of the call, the user is in the *start* state. We assume the system can identify the user (perhaps by caller ID) and selects a language model. If, at any point, the system can deal with the user's request, the call ends successfully. If not, we assume the user becomes gradually more irritated with the

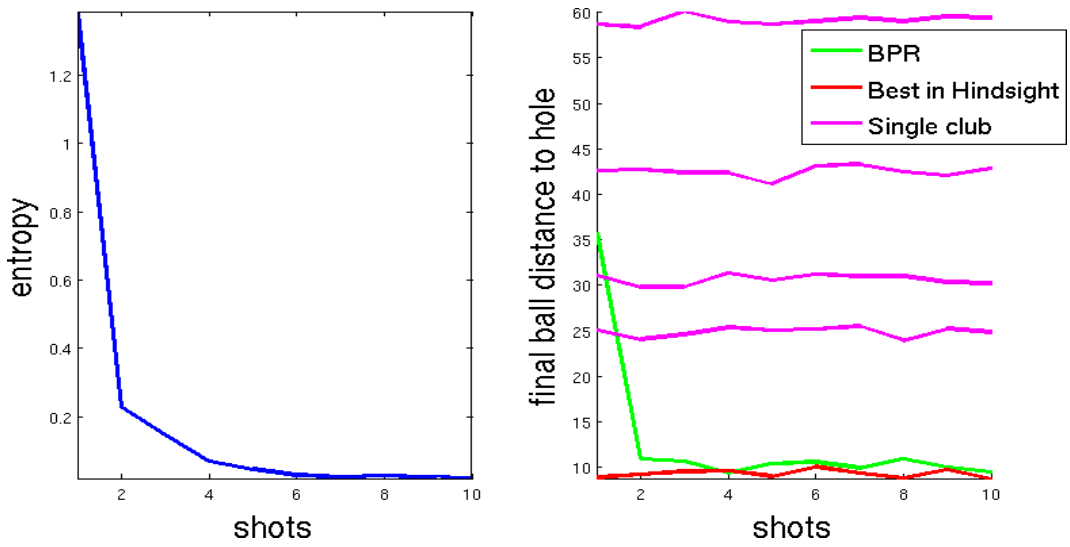


Figure 5.5: Performance of BPR on the golf club example with results averaged over 100 unknown holes, showing the decrease in entropy of the belief β and average distance to the hole (where lower scores are better). Performance of the pure four clubs, as well as the best club for each hole in retrospect, is shown for regret comparison. Although the task length was $K = 3$ strokes, we show these results for longer to illustrate convergence. Error bars have been omitted for clarity.

system, passing through states *frustrated* and *annoyed*. If the user reaches state *angry* and still has an unresolved request, she is transferred to a human. This counts as an unsuccessful interaction. Alternatively, at any point the user may hang up the call, which also terminates the interaction unsuccessfully.

The transition dynamics of this problem depend on a parameter $\rho = \frac{L-|(\pi-\lambda_i)|}{L}$ which describes how well the selected language model π can be understood by a user of type λ_i. An additional parameter η governs the trade-off between the user becoming gradually more frustrated and simply hanging up when the system doesn't respond as expected. In our experiments, $\eta = 0.3$, unless $\pi = \lambda_i$, in which case $\eta = 0$.

The aim of this experiment is to demonstrate the effect of using different observation signals to update beliefs, as described in Sec. 5.3.5. As a result, the transition dynamics and the rewards of this domain, shown in Fig. 5.6, have been selected such that only two utility signals are possible: $u = 10$ for a successful completion of the task, and $u = 3$ otherwise. Similarly, any state that transitions to the unsuccessful outcome *angry* state, receives the same reward for a transition to the unsuccessful outcome *hang up* state. Finally, all transition probabilities between the states *start*, *frustrated*, *annoyed*,

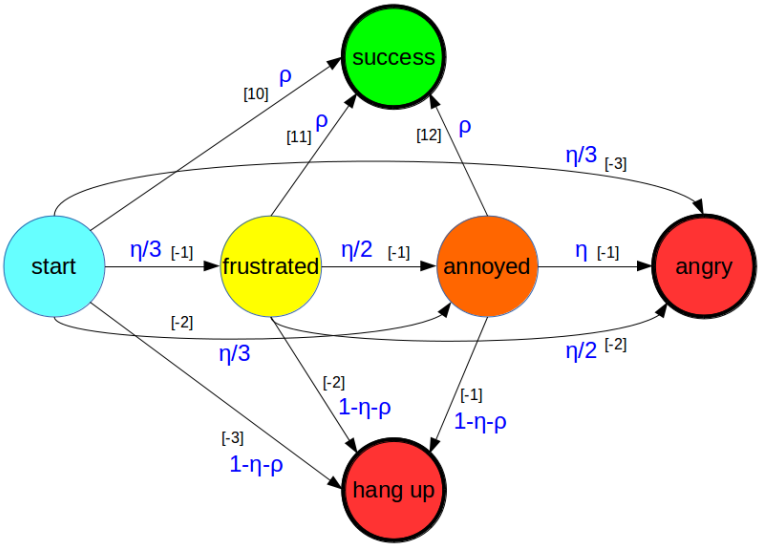


Figure 5.6: Transition system describing the online telephonic personalisation example. Circles are states, thick bordered circles are terminal states, small black edge labels in square brackets are the transition rewards, and large blue edge labels are transition probabilities. See text for a description of the parameters ρ and η.

and *angry* are independent of ρ, and thus the type. This setup mirrors the fact that in general the state sequence given by the signal (s, a, s') is more informative than the reward sequence (s, a, r), which is in turn more informative than the utility signal u. However, in many applications u may be the only of these signals available to the agent as, for example, gauging the frustration of the caller automatically may not be possible.

Fig. 5.7 shows comparative performance of BPR with these three candidate signals. Belief sampling is used as a policy selection mechanism. As expected, the lowest regret (and variance in regret) is achieved using the (s, a, s') signal, followed by the (s, a, r), and finally the u signal. We do note though that all three signals eventually converge to zero regret.

5.5.3 Surveillance

Assume a base station is tasked with monitoring a wildlife reserve spread out over some large geographical region. The reserve suffers from poaching and so the base station is required to detect and respond to poachers on the ground. The base station has a fixed location and so it monitors the region by deploying a low-flying, light-weight, semi-autonomous drone to complete particular monitoring missions. The commands

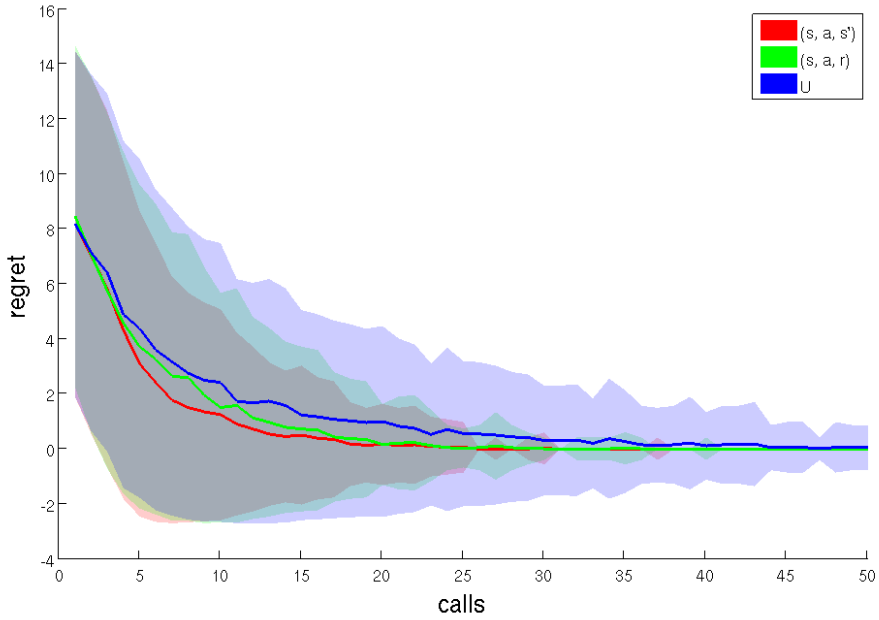


Figure 5.7: Regret, showing comparative performance of BPR on the telephone banking domain, using (s, a, s'), (s, a, r), and u as signals. The shaded areas show one standard deviation. All the methods converge at the end to zero-regret, but the richer the signal, the faster the convergence at the cost of higher storage complexity.

issued by the base station may include deploying to a specific location to scan for unusual activity, then reporting back. After each such episode the drone returns with information, *e.g.*, an indication of whether or not there was any suspicious activity in the designated region. The base station is required to use that information to better decide on the next strategy to use on the drone.

We consider a 26×26 cell grid world, which represents the wildlife reserve, and the base station is situated at a fixed location in one corner. We assume that there are 68 target locations of interest, perhaps being areas with a particularly high concentration of wildlife. These areas are arranged around four 'hills', the tops of which provide better vantage points. Fig. 5.8 depicts this setting.

At each episode, the base station deploys the drone to one of the 68 possible locations (68 policies for reaching and surveying). Poachers would be in one of these locations for the full length of the process (68 possible types). The drone is able to identify whether an intruder is at the location it visits or in an adjacent cell, providing a weak observation signal. Furthermore, the hill centres offer high vantage points, providing

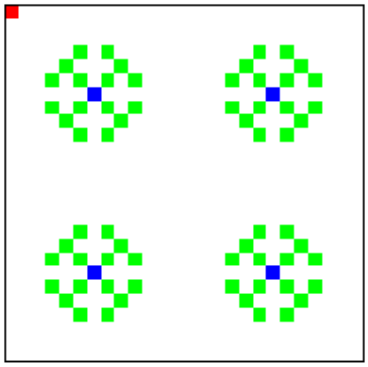


Figure 5.8: Example of the surveillance domain. The red cell is the base station, green cells correspond to surveillance locations, and blue cells are hill tops. The base station is tasked with deploying drones to find poachers which may be at one of the surveillance locations.

a signal that the intruder is somewhere around the hill. However, this signal is noisy. The reward R for a Euclidean distance d between the region surveyed and the region occupied by the poachers is

$$R \longleftarrow \begin{cases} 200 - 30d + \psi & \text{if agent surveys a hilltop and } d \leqslant 5 \\ 200 - 20d + \psi & \text{if agent surveys another location and } d \leqslant 3 \\ \psi & \text{otherwise,} \end{cases}$$

where $\psi \sim N(10, 20)$ is Gaussian noise.

Fig. 5.9 presents a comparison between six of BPR variants: BPR-KG, BPR-BE, BPR-PI, BPR-EI, in addition to belief sampling and ϵ-greedy selection with $\epsilon = 0.3$, all averaged over 10 random tasks. The standard deviations of the regret is shown in Table 5.3.

episode	ϵ-greedy	sampling	BPR-KG	BPR-BE	BPR-PI	BPR-EI
5	72.193	103.07	76.577	96.529	15.695	27.801
10	97.999	86.469	75.268	91.288	62.4	33.112
20	83.21	18.834	7.1152	17.74	72.173	16.011
50	86.172	10.897	18.489	13.813	101.69	11.142

Table 5.3: Standard deviations of the regret for the six BPR variants, after episodes 5, 10, 20 and 50.

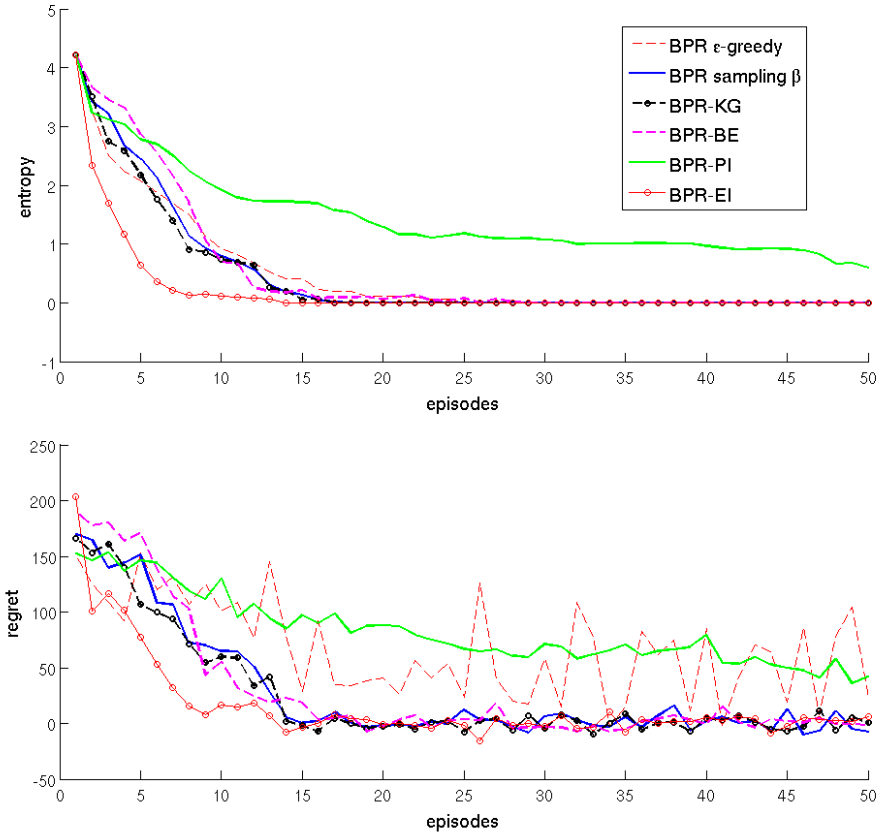


Figure 5.9: Comparison of six policy selection methods on the 68-task surveillance domain, averaged over 10 random tasks. Top, the entropy of the belief $\mathrm{H}(\beta)$. Bottom, regret $\mathcal{R}$. The standard deviations of the regret are shown in Table 5.3.

Note in Fig. 5.9 that BPR-BE, BPR-KG, and BPR with sampling β all converge in about 15 episodes, which is approximately a quarter the number that would be required by a brute force strategy which involved testing every policy in turn. Both BPR-PI and BPR with ϵ-greedy selection fail to converge within the allotted 50 episodes. BPR-EI shows the most rapid convergence.

We now compare the performance of BPR to different approaches for solving the same problem, namely multi-armed bandits for which we use UCB1 (Auer et al., 2002), and Bayesian optimisation represented by GP-UCB (Srinivas et al., 2010). We note upfront that although these are the most similar to our own in terms of the problems they solve, the assumptions they place on the problem space are different, and thus so is the information they use. These results are presented in Fig. 5.10, showing comparative performance of these approaches on the surveillance domain, averaged over 50 tasks.

We use BPR-EI in this experiment, as it was the best performing BPR variant previously.

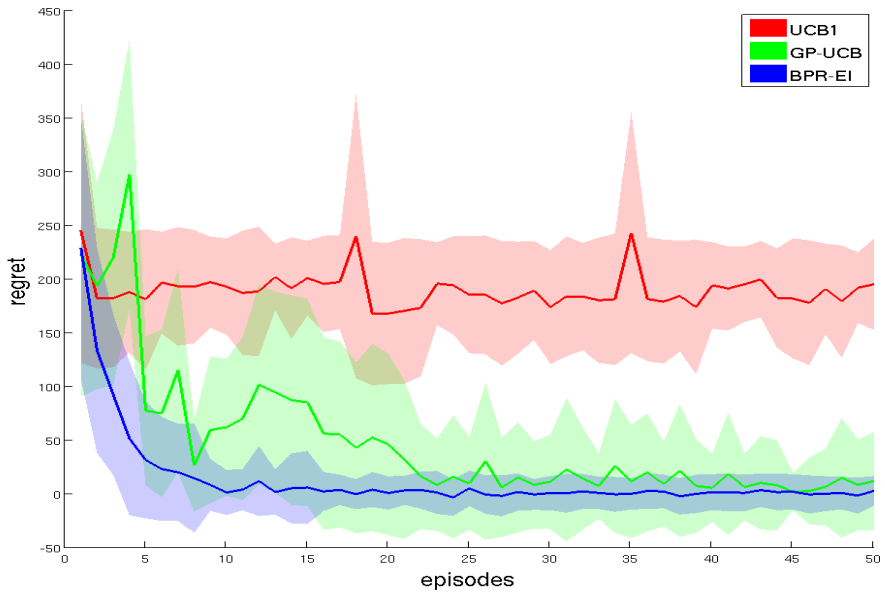


Figure 5.10: Comparison of the episodic regret of BPR-EI, a multi-armed bandit approach (UCB1), and a Bayesian optimisation approach (GP-UCB) on the 68-task surveillance domain, averaged over 50 random tasks. Shaded regions represent one standard deviation. Both BPR-EI and GP-UCB converge to zero-regret within the required time limit, while UCB1 requires more trials to converge.

A bandit approach such as UCB1 does not place a prior on the task nor assumes structure in the relationships between the arms. As a result, each arm must first be tested on each new task, meaning that the algorithm requires 68 episodes before it can begin the optimisation process. By this point, other approaches would have converged. On the other hand, an optimisation approach such as GP-UCB is better suited to this problem, as it operates with the same restriction as BPR of maintaining low sample complexity. However, unlike BPR, Bayesian optimisation requires a metric in policy space. This information is not available in this problem, but can be approximated from performance offline. As a result of this approximation, sampling a single point in GP-UCB (corresponding to executing a policy) only provides information about a local neighbourhood in policy space, whereas the same action allows BPR to update beliefs over the entire task space. In the end, all the methods converge to the correct solution (zero regret in the figure), but BPR achieves that in the least number of trials.

Further discussion of the differences between BPR and both bandits and optimisation

approaches is provided in Sect. 5.6.2 and Sec. 5.6.3 respectively.

Fig. 5.11 shows the trade-off between library size and sample complexity with respect to the regret. For each setting of library size and sample complexity pairs, we average over 200 trials. For each trial, a random subset of the full task set is used as the policy library, and the task is drawn from the full task set, meaning that this includes both seen and unseen tasks in the online phase. Intuitively and as can be seen, performance can be improved by increasing either the library size, or the time allocated (in terms of number of episodes) to complete the new task.

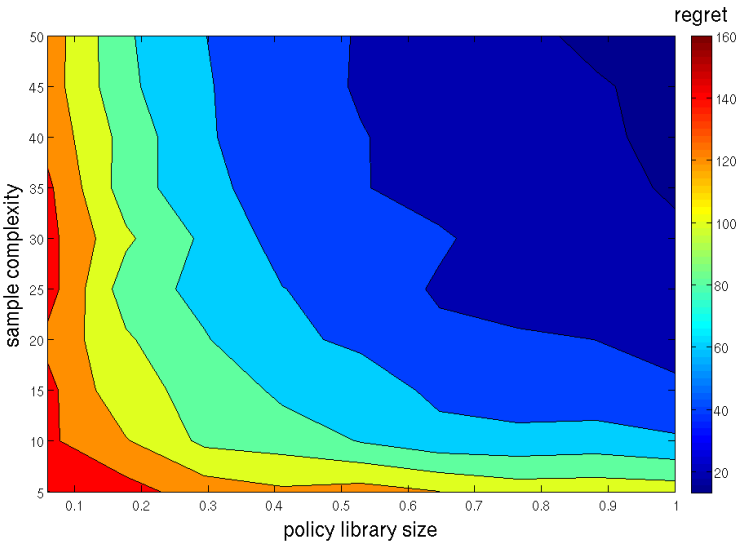


Figure 5.11: Average episodic regret for running BPR-EI on the 68 task surveillance domain, with different library sizes (as a proportion of the full task space size) and number of episodes (sample complexity), averaged over 200 random tasks.

5.6 Concluding Remarks

5.6.1 Relation to Transfer Learning

The optimal selection from a set of provided policies for a new task is in essence a transfer learning problem (see (Taylor and Stone, 2009) for a review.) Specifically, Bayesian policy reuse aims to transfer to a new, initially unknown, task a policy from a library Π which is the best for a similar seen type. One transfer approach that considers the similarity between source and target tasks is given by (Lazaric, 2008), where generated (s, a, r, s') samples from the target task are used to estimate similarity to

source tasks, measured by the average probability of the transitions happening under the task. Then, samples from the more similar source tasks are used to seed the learning of the target task, while less similar tasks are avoided to escape negative transfer. Bayesian policy reuse does not assume learning is feasible in the first place, so that it relies on transferring a useful policy immediately. Also, we use a Bayesian similarity measure which allows exploiting prior knowledge of the task space.

5.6.2 Relation to Correlated Bandits

Using a once-off signal per episode relates BPR to an instance of correlated bandits. In this problem, the decision-making agent is required to pull an arm every decision time slot from a fixed set of arms and observe its return, which will then be used to update the estimates of the values of not only the arm that was pulled, but a subset of all the arms. In the case of BPR, the arms correspond to policies and the new task instance is the bandit 'machine' that generates utilities per pull (execution of a policy).

In the correlated bandits literature, the form of correlation between the arms is known to the agent. Usually, this happens to be the functional form of the reward curve, usually called the *response surface*. The agent's task is then to identify the parameters of that curve, so that the hypothesis of the best arm moves in the parameter space of the reward curve. In response surface bandits (Ginebra and Clayton, 1995), the functional form has unknown parameters, with a prior over these parameters and with a known metric on the policy space. More recently, (Mersereau et al., 2009) present a greedy policy which takes advantage of the correlation between the arms in their reward functions, assuming a linear form with one parameter, with a known prior.

In our framework, we do not assume knowledge of the metric on policy space, and we do not specify any functional form for the response surface. Rather, we only assume continuity and smoothness of the surface. We treat the known types as a set of learnt bandit machines that share the same arms (policies). The behaviour of these machines defines local 'kernels' on the response surface, which we then approximate by a sparse kernel machine. We then track a hypothesis of which arm is the best in that space. This is similar to the Gaussian process framework, but in our case, the lack of a metric on the policy space prevents the definition of the covariance functions needed there. This point is elaborated in Sect. 5.6.3.

In another thread, dependent bandits (Pandey et al., 2007) assume that the arms in a multi-armed bandit can be clustered into different groups, the members of which have

correlated reward distribution parameters. Then, each cluster is represented with one representative arm and the algorithm proceeds in two steps: first, a cluster is chosen by a variant of UCB1 (Auer et al., 2002) applied to the set of representative arms, then the same method is used again to choose between the arms of the chosen cluster. We assume in our work that the set of seen types span and represent the space well, but we do not dwell on how they came about. Clustering is one good candidate for that and one particular example of identifying the important types in a task space can be seen in (Mahmud et al., 2013).

5.6.3 Relation to Bayesian Optimisation

If the problem of Bayesian policy reuse is treated as an instance of Bayesian optimisation, we consider the objective

$$\pi^* = \arg\max_{\pi \in \Pi} \mathrm{E}[u|x^*, \pi],$$

where $x^* \in \mathcal{X}$ is the unknown process with which the agent is interacting, and the term $\mathrm{E}[u|x^*, \pi]$ is the unknown expected performance when playing π on x^*. This optimisation involves a selection from a discrete set of alternative policies ($\pi \in \Pi$), corresponding to sampling the performance only in a discrete set of locations. Sampling this function (executing a policy for an episode) is expensive, as we have a limited number of trials and as a result the performance function must be optimised in as few samples as possible.

A Bayesian optimisation solution requires the optimised function to be modelled as a Gaussian Process (GP). There are two issues here:

1. Observations in BPR need not be the performance itself (see Sec. 5.3.4), while the GP model is appropriate only where the process that is optimised can be sampled.

2. BPR does not assume knowledge of the metric in policy space. This is however required for Bayesian optimisation, so as to define a kernel function for the Gaussian process. An exception to this case is when all policies belong to a parametrised family of behaviours, placing the metric in parameter space as a proxy for policy space.

Still, we assume smoothness and continuity of the response surface for similar tasks and policies, which is a standard assumption in Gaussian process regression. Bayesian policy reuse uses a belief that tracks the most similar seen type, and then reuses the best

policy for that type. This belief can be understood as the *mixing coefficient* in a mixture model that represents the response surface.

To see this, consider Fig. 5.12 which shows an example 2D response surface. Each type is represented by a 'band' on that surface; a set of performance curves for the tasks of the type which induced performance under any policy is contained in an ϵ-ball. That is, bands can by wide in plateau performance regions. Nonetheless, the performance curves are only precisely known for the set of known policies, in terms of means and variances. Projecting these 'bands' into policy-performance space (the panels on the left in Fig. 5.12) gives a probabilistic description of the performance of the different policies on that type, which can be modelled as a Gaussian process. Each of these projected GPs would be a component of a mixture model that represents the surface and would represent the type's contribution to the surface.

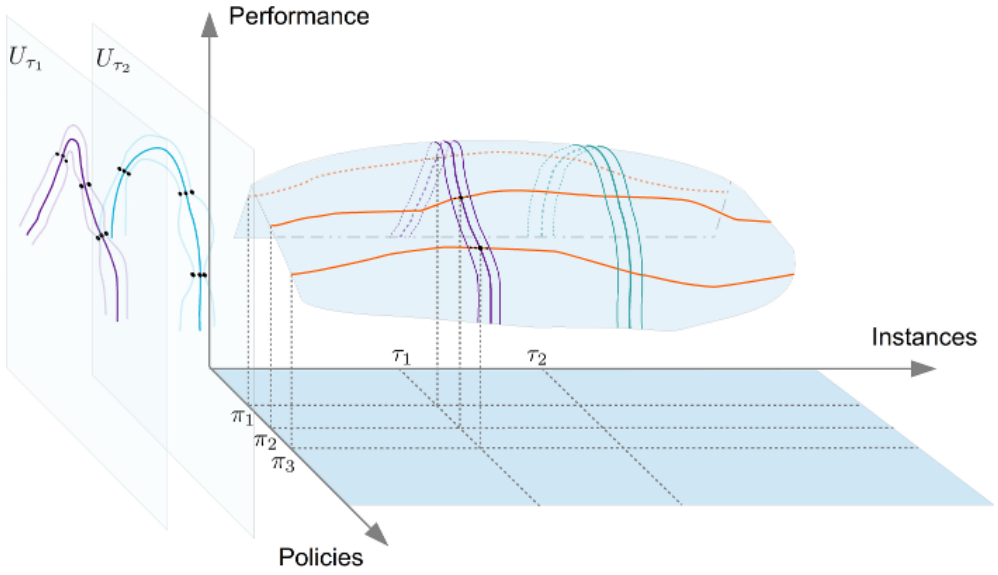


Figure 5.12: An example 2D response surface. The 'bands' on the curve show two types, and the lines that run through the curve from left to right are policy performances for all types. The agent only has access to the intersection of type bands with policy curves (the black dots). Shown on the left are performance curves of the two types τ_1 and τ_2 under all policies as projections of the bands onto the Policy-Performance plane. The projections are represented as Gaussian processes. The dots on the curves are observed values of policy performance of tasks in the corresponding type, making the variance shrink. The aim of BPR is to approximate the performance curve of the new, unknown task, so that choosing the right policy becomes trivial.

Any new task instance corresponds to an *unknown* curve on the surface. Given that

the only knowledge possessed by the agent from the surface is these type probabilistic models, Bayesian policy reuse implicitly assumes that they act as a basis that span the space of possible curves, so that the performance under any new task can be represented as a weighted average of the responses of the seen types[3]. To this extent, the performance for the new task instance is approximately identified by a vector of weights, which in our treatment of BPR we refer to as the type belief. In summary, BPR fits a probabilistic model to the performance curve of the new task by sampling and weight adjustment of an approximated mixture of Gaussian processes representing the seen types.

5.6.4 Other Bayesian approaches

A Bayesian treatment to the Policy Gradient method in reinforcement learning is introduced in (Engel and Ghavamzadeh, 2006). The gradient of some parametrised policy space is modelled as a Gaussian process and paths sampled from the MDP (completed episodes) are used to compute the posteriors and to optimise the policy by moving in the direction of the performance gradient. The use of Gaussian processes in policy space is similar to the interpretation of our approach, but it is used there to model the gradient rather than the performance itself.

When no gradient information is available to guide the search, (Wingate et al., 2011) propose to use MCMC to search in the space of policies which is endowed with a prior and various kinds of hierarchical priors that can be used to bias the search are discussed. In our work, we choose the policies using exploration heuristics based on offline-acquired performance models rather than using kernels in policy space and policy priors. Furthermore, we start with a small fixed set of policies, and search in it for the optimal selection.

5.6.5 Storage complexity

As described in Sec. 5.3.4, the use of different signals entail different observation models and hence different storage complexities. Assume that $|S|$ is the size of the state space, $|A|$ is the size of the action space, $|\Pi|$ is the size of the policy library, N is the number of seen types, $|R|$ is the size of the reward space, T is the duration of an episode, and b is

[3]Note that this assumption will create a bias in the agent's estimated model of the type space toward the types that have seen more often before. We assume that the environment is benign and that the offline phase is long enough to experience representative types.

the number of bits needed to store one probability value. For the performance signal, the storage complexity of the observation model is upper bounded by $\mathcal{SC}_{\text{perf}} = |\Pi|\,N\,|R|\,b$ for the average reward case, and $\mathcal{SC}_{\text{perf},\gamma} = |\Pi|\,N\,\frac{1-\gamma^T}{1-\gamma}\,|R|\,b$ for the discounted reward case. For the state-action-state signals, the term is $\mathcal{SC}_{s'} = |\Pi|\,N\,|R|\,|S|\,|A|\,b$, and for the immediate reward signal we get $\mathcal{SC}_r = |\Pi|\,N\,|S|^2\,|A|\,b$. In applications that have $|R| > |S|$ we get the ordering $\mathcal{SC}_{\text{perf}} < \mathcal{SC}_r < \mathcal{SC}_{s'}$.

In this chapter we discuss policy reuse as a method to policy abstraction. We address the policy reuse problem, which involves selecting between a number of different policies from a library so as to minimise regret within a short number of episodes. This kind of abstraction is most valuable when learning is infeasible, and in many application domains where tasks have short durations, such as human interaction and personalisation, as well as monitoring tasks.

We introduce Bayesian Policy Reuse as a Bayesian framework that uses that abstraction to solve the policy reuse problem. It operates through tracking a belief defined over the space of known tasks that represent the similarity in behaviour to the new task. The belief gets updated using observation signals provided by the domain, and using models of seen tasks, trained offline for each policy. Several mechanisms for selecting policies from the belief are also described, giving rise to different variants of the core algorithm.

The approach is empirically evaluated in three domains. We show comparisons between the different variants of BPR, as well as a bandit algorithm (UCB1) and a Bayesian optimisation algorithm (GP-UCB). We also show the effect of using different observation signals on the task performance, and illustrate the trade-off between the policy library size and sample complexity required to achieve a certain level of performance in a task.

Chapter 6

Conclusions

6.1 Summary

The goal of this thesis is to develop methods for autonomous learning of policy space abstractions. We present three approaches for variations of this problem: using the technique of option discovery with a collection of MDPs, using computational topology on a set of demonstrated trajectories, and using models of task space for policy reuse. The outlook is to enable adaptive and seamless integration of domain understanding to the continual process of solving new tasks.

Using the option framework, we present a method that takes in a set of policies for previous tasks, clusters the close-by states that are most used in successful runs into regions, and restricts the input policies to these regions to give options. The framework of ILPSS also allows continual adding and removing of options from the hierarchy based on the qualities of the new tasks the agent will see.

Employing computational topology, we extract from a set of demonstrated trajectories homology-equivalence classes using a simplicial complex constructed from the experienced configuration space. These classes serve as an abstraction of the full set of trajectories. We also show the generality and usefulness of the approach when we include a cost landscape into the complex construction and generate paths that do not cross a specific cost threshold.

The third method summarises the policy space using a set of policies learnt for a collection of sampled tasks in a domain. Using a non-parametric model of the performance of the policies on different tasks, it employs a Bayesian principle of policy reuse to select a candidate policy from the collection to be applied on a new task. This approach exploits correlation and similarity in performance to find a quick solution suggestion to a real-time situation.

The findings of the thesis were validated using a set of experimental domains, simulated and real. Among others, we used the Soccer Server of RoboCup 2D Simulation League to learn options in a soccer scenario for ILPSS, and we used a Baxter robot to generate the trajectory sets that we used in the topology experiments. Also, we have developed simulations of navigation domains, real-time personalisation and surveillance.

6.2 Key Findings

The hypothesis of this thesis is that *the ability to learn skill hierarchies is one essential capability of a lifelong learning agent. Proper representations of the lifelong learning problem coupled with various off-the-shelf machine learning tools can achieve better generalisations in the learning problem.*

Using hierarchical structure of policies is key to tame sample complexity, especially when the requirements of the environment are such that the agent has to respond to a spectrum of variability and within bounded-time frame. When the agent should interact with a growing collection of tasks, the common approach of fixing the hierarchy beforehand becomes quite restrictive. Learning the hierarchical structure on the go, on the other hand, gives the desired adaptivity and sensitivity to environment changes, and allows lifelong experience to accumulate efficiently.

6.2.1 Learning

To generalise from the specific methods described in this thesis to other contexts, we note the following attributes that the methods have:

- They all start from some kind of 'positive' behavioural examples, performed by the agent itself for sample tasks in the domain. *E.g.*, these examples can be policies of tasks that have been solved before (for the option discovery and task space modelling methods), or kinesthetic demonstrations of physical robot systems (for the computational topology approach). This is a typical requirement of *supervised learning*. Here, the target function to learn from these examples is the structure in policy space that explains and justifies the generated policies for the sampled tasks.

- They involve a technique of clustering that reduces the input space into a more meaningful reduced description. Candidate metrics for clustering are ones that reward distinctiveness and abstract similarity in behaviours. Considered examples include *across-task* state visitation frequency, homotopy class, and inter-task performance. The clustering operates on states to give generalised bottlenecks which encompass not only a single target state, but a region of importance where useful behaviours could be extracted. We operated on trajectories to expose homotopy-equivalent classes and we clustered MDPs to create a representative

policy library. *Unsupervised learning* is key to self-organisation and structure learning.

- They generate policies for the hierarchy components by *shaping* or *reusing* the input policies in light of the clustering boundaries. That includes the immediate population of extracted options in ILPSS with behaviours using reused policies, as well as creating the topological options from the homology equivalence classes of trajectories.

6.2.2 Planning

The second important component to achieve the motivation of this work after hierarchy learning is planning/inference in new task instances. We have shown two possibilities of methods that can be used with the policy hierarchy:

- Sequential composition: This is featured in the SMDP learning with the option discovery method and the Topological Options. Sequencing subpolicies achieve higher flexibility and allows for responding to a range of novel task instances. However, this comes at the cost of increased learning time, which is still much less than learning in the original problem without the hierarchy. We used the framework of options as the framework of choice when we dealt with hierarchical reinforcement learning mainly due to its flexibility and generality. However, any suitable framework can be used instead.

- Flat reuse: This is featured in the homology-equivalence classes and in policy reuse. Here, no deliberation is needed beyond selecting the best response, so that a behavioural policy can be quickly initialised. This is a requirement of transfer learning for real-time embodied agents in rich environments. On the other hand, reuse does not allow behaviours to extrapolate and adapt to a different new instance, bounding the realisable optimality.

The best option for a domain would be task-specific. However, it is possible to have both reasoning types in one framework and give the agent the power to choose based on the specific task requirements.

6.2.3 Tools

The choice of tools that we use to extract structure from the input shapes the possible outcomes and define their limitations. We extended the typical notion of bottlenecks

to accommodate more general and complex concepts of importance regions, and we used vector field structures and persistent homology to expose useful perspectives not possible with the typical analysis tools of trajectories, like connectivity graphs.

The usefulness of computational topology is quite pronounced in the natural example of navigation. However, we believe that this kind of methods would be applicable to many other settings and scenarios, where the topological features have similar high significance to the task. We demonstrated that with few manipulation examples with a physical robot system. We show that more knowledge about the task and the domain can be extracted from a set of demonstrations than what a graph can extract using simplicial complexes and the framework of persistent homology. Also, these tools helped us to abstract parameter tuning.

6.3 Future Work

The problem of lifelong learning agents in changing environments will only become more relevant with time. New aspects and perspectives into the problem should be explored, as well as the need to exploit the available tools to make beneficial progress.

- This thesis assumed that the environment contains other 'actors' that do not adversely or tightly affect the task of the agent. Incorporating concepts of opponent modelling and plan recognition to account for other agents in the environment is an important direction.

- We dealt mainly with tasks that can be described as episodic and goal-seeking to build our methods. There are other kinds of tasks that deserve being studied in the same framework, like, for example, periodic tasks or infinite-horizon tasks with change regimes.

- The use of topological inference into configuration spaces of robots of various kinds of tasks is an interesting next step. This includes applications like motion planning in topological coordinates or with topological constraints, real-time probabilistic recognition and classification of qualitatively-similar demonstrations, and a more-forgiving topological framework for programming by demonstration.

Appendix A

Short Tutorial in Computational Topology

We review some of the concepts of computational topology as used in this thesis. The references (Edelsbrunner and Harer, 2010; Hatcher, 2002) provide a good source for more details.

A.1 Topological Space

An *open set* is a generalisation of the real line's open interval to a general space. Then, by utilising this concept, a *topological space* is a generalisation of a metric space by that it does not require a metric. Rather, similarity to a point c is defined by the possibility of inclusion in an *open* ball $\{x \mid |x-c| < \varepsilon\}$ centred at the point. That is, open sets define a notion of a continuous function without a metric.

A.2 Homotopy

Given topological spaces X, Y, a *homotopy* between two continuous maps between the spaces $f, g : X \to Y$ is a continuous map $H : X \times [0,1] \to Y$ such that $H(x,0) = f(x)$, $H(x,1) = g(x)$ for all $x \in X$, and $H(x,r)$ for $0 < r < 1$ continuously 'interpolates' from $f(x)$ to $g(x)$. Then, f, g are called *homotopic*, denoted by $f \simeq g$. In other terms, it is possible to continuously morph one of the maps to the other.

A.2.1 Homotopy equivalence

Two topological spaces are called *homotopy equivalent* if there exists continuous maps $f : X \to Y$ and $g : Y \to X$ such that $g \circ f \simeq 1_X$ and $f \circ g \simeq 1_Y$, where $1_X, 1_Y$ denote the identity functions on X, Y respectively. That is, following one map from X to Y then the second from Y to X gets back to where it started, *i.e.*, the mapping preserves the 'shape' information. We then call X, Y *homotopy equivalent*, denoted by $X \simeq Y$. Homotopy equivalence forms an equivalence relationship among topological spaces enabling us to classify spaces into discrete classes. The example of a homotopy equivalence between a doughnut and a coffee mug provides a popular example (Fig. A.1).

Given a topological space X, we call two curves $f, g : [0,1] \to X$ with the same end-points $f(0) = g(0)$, $f(1) = g(1)$ homotopy equivalent if f can continuously be deformed to g while keeping the end-points fixed (see Fig. A.2). Homotopy equivalence classes of paths provide a classical topological approach to studying classes of paths.

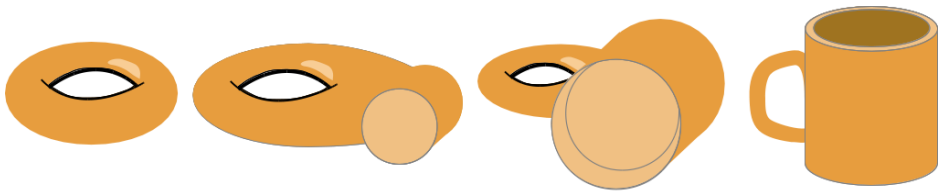


Figure A.1: Homotopy equivalence between a doughnut and a coffee mug. The doughnut can be continuously morphed into a mug.

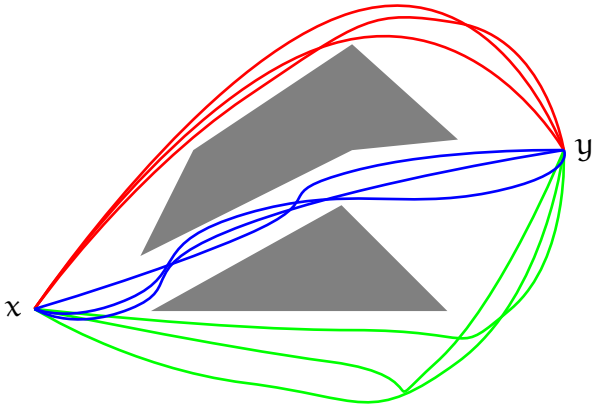


Figure A.2: Three homotopy equivalence classes of obstacle-free paths from x to y depicted in red, blue and green. None of the paths in any class can be continuously deformed into any of the other classes.

A.2.2 Fundamental group

The *first fundamental group* $\pi_1(X, x_0)$ for the point x_0 is a group whose elements consist of closed continuous curves that start and end in x_0 and lies entirely in X, and whose operation is concatenation of paths. Two closed paths $\alpha, \beta : [0,1] \to X$ through x_0 lie in the same equivalence class if there exists a homotopy between them that maps the base-point x_0 to itself. When X is *path-connected*, *i.e.* there is a path that connects any two points in X, the group $\pi_1(X, x_0)$ is independent of the chosen base-point x_0 and hence often denoted simply by $\pi_1(X)$. Furthermore, if the spaces X, Y are homotopy equivalent, $\pi_1(X)$ and $\pi_1(Y)$ are isomorphic groups, admitting a one-to-one correspondence between their elements.

Two paths γ_1, γ_2 in X with the same start point x and end point y can be deformed into each other via a homotopy if the closed curve following γ_1 from x to y and then the inverse of γ_2 from y to x is *trivial* in $\pi_1(X)$, *i.e.*, equal to the group's identity.

Hence, $\pi_1(X)$ is a natural group to consider for the purpose of motion planning and classification of paths.

Unfortunately, for a general topology space X, no general purpose method exists to compute the group structure of $\pi_1(X)$, in addition to it being non-commutative and of infinite cardinality. For this, we turn to the weaker concept of *homology* with binary coefficients, using an approximation of X by a *simplicial complex*. Both concepts will be described in the next two sections.

A.3 Simplicial Complex

One representation of a topological space that is useful to compute homological information is a simplicial complex. Simplicial complexes are constructs that generalise undirected graphs. Define a p-*simplex* σ to be the convex hull of a set of $p+1$ points. We call p the dimension of the simplex. For example, a 0-simplex is a single point, or a *vertex*. A 1-simplex is an *edge* connecting two vertices. A 2-simplex is a *triangle*, the convex hull of 3 vertices.

For example, if a 0-simplex represents a point in the free configuration space of a robot (a pose), a 1-simplex would represent the relation between two poses, *e.g.* that they are mutually accessible from each other. This is also a statement about all the configurations that happen in between these two poses.

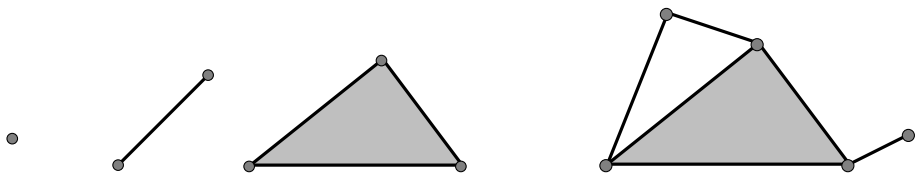


Figure A.3: Examples of a vertex, an edge and a triangle; and an example of a simplicial complex, comprising five 0-simplices (vertices), six 1-simplices (edges) and a single 2-simplex (triangle).

The convex hull of any subset of the points that make a simplex σ is a *face* of σ. An abstract simplicial complex $\mathcal{K}$ is a finite non-empty collection of simplices, such that all their intersections are members in the collection: $\sigma \cap \tau \in \mathcal{K}$ for all $\sigma, \tau \in \mathcal{K}$, and all their members are in the collection: for all $\sigma \in \mathcal{K}$ and $\varnothing \neq \tau \subseteq \sigma \in \mathcal{K}$, then $\tau \in \mathcal{K}$.

A.3.1 Delaunay-Čech complexes

Consider a set of uniformly sampled points $V = \{x_1, \ldots, x_n\} \subset X$ from a topological space $X \subset \mathbb{R}^d$ endowed with the Euclidean metric d. It is natural to approximate X by the space of the union of balls $X_\varepsilon = \bigcup_{i=1}^{n} \mathbb{B}_\varepsilon(x_i)$, where $\mathbb{B}_\varepsilon(x_i)$ is a ball of size ε centred at x_i, $\mathbb{B}_\varepsilon(x_i) = \{x \in X : d(x, x_i) < \varepsilon\}$. $\varepsilon > 0$ depends on the sample density.

We can ask about the homology groups of X as approximated by X_ε. To compute homology, we can represent X_ε by any simplicial complex $\mathcal{F}_\varepsilon$ which is homotopy equivalent to the topological space X_ε and hence carries the same topological information.

The *Čech-complex* $C_\varepsilon = \{\sigma \subseteq V : \cap_{x \in \sigma} \mathbb{B}_\varepsilon(x) \neq \varnothing\}$ is an abstract simplicial complex which is homotopy equivalent to X_ε. A simplex is added to C_ε for any subset of balls with a non-empty intersection. However, C_ε has two drawbacks:

1. as ε increases, the number of simplices increases dramatically, until eventually every subset of V is contained in C_ε, resulting in a top-dimensional simplex of dimension $|V| - 1$. This is undesirable when V contains millions of points.

2. C_ε is abstract in that simplices do not directly correspond to geometric subsets of X.

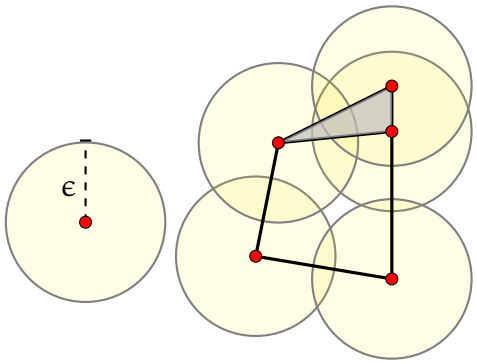


Figure A.4: An example of a Cech complex

Given V, we instead consider the *Delaunay triangulation* D of V. The Delaunay triangulation of a set of points is the dual graph of the *Voronoi diagram* of the points, in which each point x of V is assigned a cell of points $V_x \subset X$ closer to x than to any other point in the set V. [1] One example is in Fig. A.5.

[1] D is well-defined in arbitrary dimension d if $(d+1)$-tuples of points in V are in general position, which occurs with probability one for uniformly sampled points. Furthermore, any configuration not in general position can be perturbed by a small amount to satisfy this criterion.

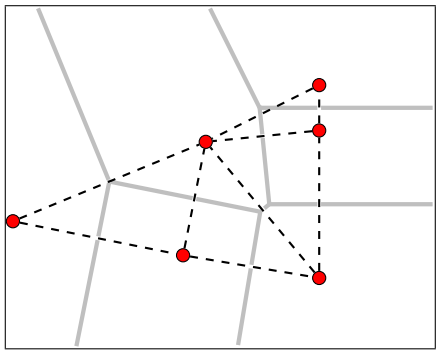


Figure A.5: An example of a Delaunay triangulation (black, dashed) and Voronoi diagram (grey, solid) for a set of points.

Considering the union of all convex hulls of $d+1$ points from $V \subset \mathbb{R}^d$ in the Delaunay triangulation, along with their faces, we can consider D as a simplicial complex.

Given V, the Delaunay complex is a geometric simplicial complex, where a p-simplex $\sigma = \{x_0, \ldots, x_p\}$ corresponds to the convex hull $\mathrm{Conv}(x_0, \ldots, x_p) \subset X$, and it can be defined by

$$DC = \{\sigma \subseteq V : \cap_{x \in \sigma} V_x \neq \varnothing\},$$

where V_x denotes the Voronoi cell containing x (the grey cells in Fig. A.5). The union of all simplices of DC is the convex hull $\mathrm{Conv}(V)$.

A.4 Simplicial Homology with $\mathbb{Z}_2$ Coefficients

For a simplicial complex $\mathcal{K}$, a p-*chain* c is a formal sum of p-simplices $\{\sigma_i\}_{i=1}^k$, $c = \sum_{i=1}^k \lambda_i \sigma_i$ with $\lambda_i \in \mathbb{Z}_2 = \{0, 1\}$, so that chains have only binary coefficients 0 or 1. $C_p(\mathcal{K})$ is the vector space of all p-chains.

For every p-simplex σ let $\partial\sigma$ be the $p-1$-chain formed by the formal sum of all $p-1$ dimensional faces of σ. We call this operator the *boundary*. This operator acts as a a linear map on $C_p(\mathcal{K})$, mapping p-chains to $p-1$-chains. if $c = \partial\omega$ for some $\omega \in C_{p+1}(\mathcal{K})$, c is intuitively called a p-*boundary*. If $\partial c = 0$, a chain whose boundary is zero (*i.e.* closed), then c is called a p-*cycle*.

The set of p-boundaries and p-cycles are denoted by $B_p(\mathcal{K})$ and $Z_p(\mathcal{K})$ respectively. Because $\partial\partial = 0$, every boundary is a cycle, and thus we have $B_p(\mathcal{K}) \subseteq Z_p(\mathcal{K})$.

Figure A.6: An example of a 1-boundary; and a 1-cycle which is not a 1-boundary

A.4.1 Homology group

The *quotient* of some group G is another group under the operation of G, made of partitioning G into equivalence classes. The quotient is represented by G/H, where H is the *normal subgroup* of G, which the equivalence relation is defined in respect to. Elements of G that are equivalent when operating on H are aggregated in the equivalence classes of the quotient.

The quotient $H_p(\mathcal{K}) = Z_p(\mathcal{K})/B_p(\mathcal{K})$ is called the p^{th} *homology group* of $\mathcal{K}$, and it contains the classes of p-cycles that are equivalent on adding/removing boundaries.

The dimension of this group, $b_p(\mathcal{K}) = \dim(H_p(\mathcal{K}))$ is of a special interest and is called the p^{th} *Betti-number* of $\mathcal{K}$. $b_0(\mathcal{K})$ is equal to the number of connected components of $\mathcal{K}$. $b_1(\mathcal{K})$ counts the number of 'holes', and $b_2(\mathcal{K})$ counts the number of 'voids' in $\mathcal{K}$.

We are particularly interested in $H_1(\mathcal{K})$. A 1-cycle is simply a set of edges forming zero or more disjoint closed edge paths. Two 1-cycles c_1, c_2 are equivalent in $H_1(\mathcal{K})$ if $c_1 - c_2 = \partial\omega$ for some 2-cycle ω, *i.e.* they differ by the boundary of some 2-chain, made of a union of 2-simplices.

A.4.2 Homology generators using Smith Normal Form decomposition

We want to extract a basis of $H_1(\mathcal{K})$ so that any 1-cycle can be described as a $\mathbb{Z}_2$-weighted sum of the generators. This is possible by a Smith Normal Form decomposition of the *boundary matrices* of the simplicial complex in $\mathbb{Z}_2$, *i.e.* by considering $\mod \mathbb{Z}_2$ after every operation. The pth-boundary matrix Λ has a row for every $p-1$-simplex, and a column for every p-simplex in the complex. The entry would be 1 if the $p-1$-simplex is a face of the p-simplex, and 0 otherwise.

A Smith Normal Form (SNF) of a matrix is a reduced matrix of the same size that only has non-zero entries in the first portion of the diagonal (ones in $\mathbb{Z}_2$), and 0 everywhere else. Denote with n the number of ones in the diagonal. An SNF

decomposition of Λ uses row and column operations to give

$$R = Q^{-1}\Lambda V, \tag{A.4.1}$$

where R is the reduced form, V encodes the column operations and Q^{-1} encodes the row operations. The benefit of this decomposition is that it allows reading off the generators for $Z_p(\mathcal{K})$ and $B_p(\mathcal{K})$ easily. The generators of Z_p are the last $N_p - n$ columns in V, where N_p is the number of p-simplices in the complex, while the generators of B_p are the first n columns of Q.

For our case, after finding the generators of $Z_1(\mathcal{K})$ and $B_1(\mathcal{K})$, finding the generators of $H_1(\mathcal{K})$ requires another decomposition. First, the basis of $B_1(\mathcal{K})$ is expressed in the basis of $Z_1(\mathcal{K})$ by solving the linear equations $B_1 = Z_1 Y$, where Y are the coefficients of the projection. Then, a second decomposition of Y gives $Y = q r v^{-1}$. By combining the two equations we get:

$$B_1 = Z_1 q r v^{-1}. \tag{A.4.2}$$

$$B_1 v = Z_1 q r. \tag{A.4.3}$$

The generators for $H_1(\mathcal{K})$ can be read from the columns of $Z_1 q$ that correspond to 0-columns in r. $Z_1 q$ is a new basis of Z_1 after applying the column operations in q. Similarly, $B_1 v$ is a new basis of B_1. Thus, for every 1 in the diagonal of the reduced r, the corresponding columns in $Z_1 q$ will map to a boundary. On the other hand, the 0-columns in r correspond to columns in $Z_1 q$ that represent the null space of the boundary group, and thus they represent generators of H_1.

A.4.3 Defining the equivalence classes

After finding the basis of H_1, it is straightforward to define the equivalence classes. Projecting any 1-chain into the basis made of the generators of $H_1(\mathcal{K})$ and $B_1(\mathcal{K})$ will give the homology class of the cycle as a vector of $\mathbb{Z}_2$ coefficients.

A.5 Persistent Homology

following (Edelsbrunner and Harer, 2008; Carlsson, 2009; Edelsbrunner and Harer, 2010; Bauer et al., 2014), we now discuss the persistent homology framework.

A.5.1 Filtrations

A *filtration* of a simplicial complex $\mathcal{K}$ and for some index set $I \subset \mathbb{R}$ is a sequence of simplicial complexes $\{\mathcal{K}_\varepsilon \subseteq \mathcal{K} : \varepsilon \in I\}$ such that $\varepsilon \leqslant \varepsilon'$ implies $\mathcal{K}_\varepsilon \subseteq \mathcal{K}_{\varepsilon'}$, $\varepsilon, \varepsilon' \in I$. A *simplex-wise filtration* of a simplicial complex $\mathcal{K}$ is a sequence of simplicial complexes $\varnothing = \mathcal{K}_0 \subset \mathcal{K}_1 \subset \ldots \subset \mathcal{K}_n = \mathcal{K}$ such that $\mathcal{K}_i = \mathcal{K}_{i-1} \cup \{\sigma_i\}$, for a simplex σ_i of $\mathcal{K}$. The inclusion $\alpha_i^j : \mathcal{K}_i \to \mathcal{K}_j$ for $i \leqslant j$ yields a linear map on the homology groups $h_i^j : H_p(\mathcal{K}_i) \to H_p(\mathcal{K}_j)$.

Let us consider the case of the Delaunay-Čech complex DC_ε which yields a natural filtration with respect to the function $f(\sigma) = \min\{\varepsilon : \cap_{x \in \sigma} \mathbb{B}_\varepsilon(x) \neq \varnothing\}$ for $\sigma \in DC$, which is the minimal filtration value at which σ comes into existence. We have $DC_\varepsilon \subseteq DC_{\varepsilon'}$ for all $0 \leqslant \varepsilon \leqslant \varepsilon'$ and there exists a maximal ε_{max} such that $DC_{\varepsilon \geqslant \varepsilon_{max}} = D$, where D denotes the full Delaunay triangulation of the underlying points. We can obtain a simplex-wise filtration $DC(i) = \cup_{j \leqslant i}\{\sigma_j\}$ respecting the partial order defined by f. Figure A.7 shows an example of a Delaunay-Čech complex filtration.

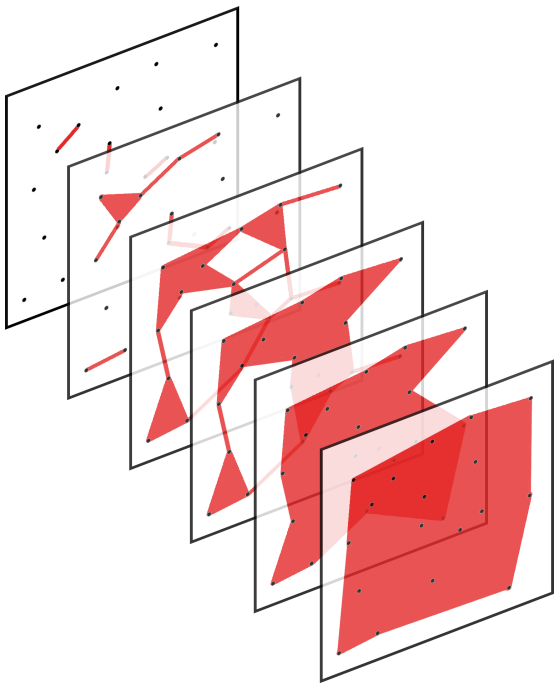


Figure A.7: A Delaunay-Čech filtration DC_ε for different ε values increasing towards the bottom, constructed from sampled points indicated by the black dots.

A.5.2 Persistence

We say that some homology class $\alpha \in H_p(\mathcal{K}_i)$ is *born at index* i if $\alpha \notin \mathrm{im}(h_{i-1}^{i})$; that is, α has no precedent in $\mathcal{K}_{i-1}$. A class $\alpha \in H_p(\mathcal{K}_i)$ born at index i is said to *die at index* j if $h_i^{j-1}(\alpha) \notin \mathrm{im}(h_{i-1}^{j-1})$ but $h_i^{j}(\alpha) \in \mathrm{im}(h_{i-1}^{j})$, so that α born at i maps to itself (persists) until time $j-1$ then disappears at time j and maps to something that existed before. The pair (i,j) is called the *persistence pair* of α, in that the homological feature survives in the filtration for a duration $j-i$, also called the *persistence*. Features that are born at i but do not die are called *essential* features, denoted by the infinite persistence (i,∞).

For $i \leqslant j$, the p-th *persistent homology group* is defined to be

$$H_p^{i,j} = Z_p(\mathcal{K}_i)/(B_p(\mathcal{K}_j) \cap Z_p(\mathcal{K}_i)).$$

Intuitively, this captures the p-dimensional homological features that exist at filtration i and which persist in the filtration between i and j. These features are the cycles that persist with adding/removing boundaries that existed before i or are created up to j. When $i = j$, the usual notion of homology is recovered: $H_p^{i,i} = H_p(\mathcal{K}_i) = Z_p(\mathcal{K}_i)/B_p(\mathcal{K}_i)$. Fig. A.8 illustrates the birth and death of a cycle in DC_ε for an example point cloud.

A graphical representation of this information is obtained by the *persistence diagram*. This diagram contains a point (i,j), $i \leqslant j$, for each class born at index i and which dies at index j. The distance to the diagonal indicates how long the feature persists. See Fig. A.9 for an example persistence diagram for the previous point cloud.

A.5.3 Computation via matrix reduction

The boundary operator $\partial : \bigoplus_p C_p(\mathcal{K}) \to \bigoplus_p C_p(\mathcal{K})$ is a linear map in the vector space of chains of $\mathcal{K}$. We express this map in a matrix form, using the ordered basis $\sigma_1, \ldots, \sigma_n$, yielding a $n \times n$ matrix M with $\mathbb{Z}_2$ entries. An entry (i,j) would be 1 if σ_i is a face of σ_j. Note that this matrix is upper triangular. We denote by M_j the j^{th} column, and by m_{ij} the (i,j)-entry. We let $\mathrm{low}(M_j) = \max\{i : m_{ij} \neq 0\}$, and note that low can be undefined for some columns $M_j = 0$. A column addition is called a *reducing left-to-right addition* if it decreases $\mathrm{low}(M_j)$. M_j is called *reduced* if $\mathrm{low}(M_j)$ cannot be decreased by applying any sequence of left-to-right additions, and the matrix M is reduced if all columns are reduced.

The standard persistence algorithm (Edelsbrunner and Harer, 2010) applies left-to-right column additions until M is reduced, yielding the reduced matrix R. We can

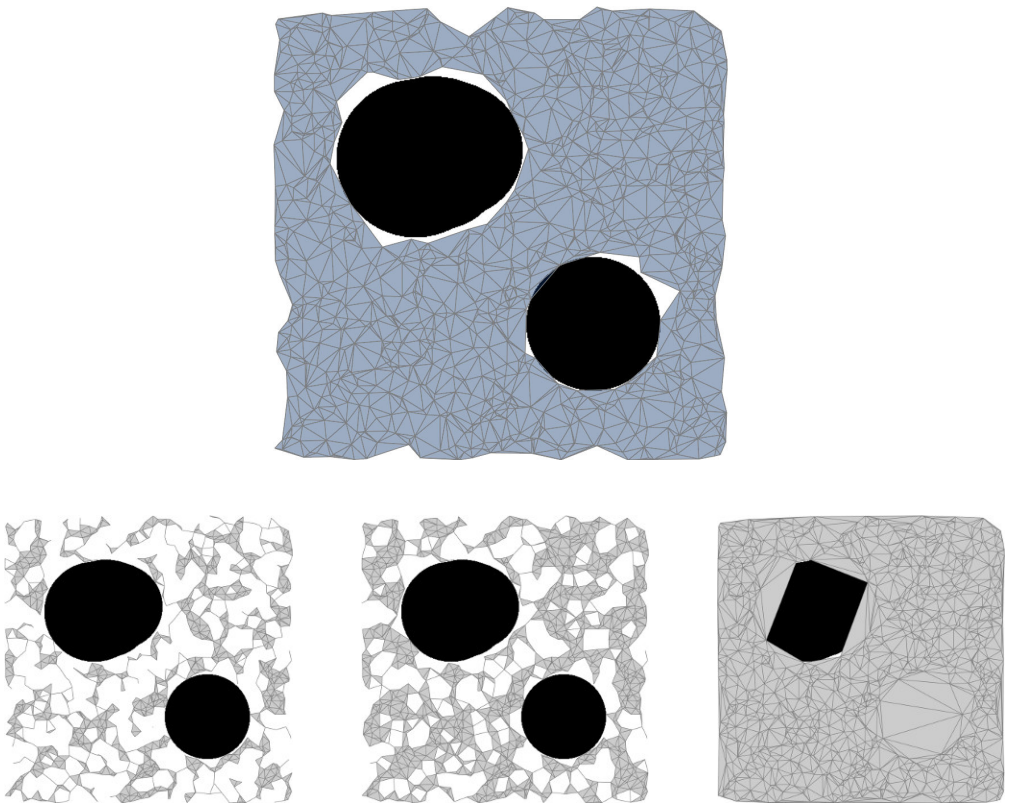


Figure A.8: A reconstruction of a configuration space from 1000 samples on a square of side-length 500. DC_{25} is displayed in the top which yields a good approximation to the original space. The bottom row displays $\mathrm{DC}_{10.58}$, $\mathrm{DC}_{12.96}$, $\mathrm{DC}_{74.0}$ which corresponds, respectively, to the birth of the smaller hole (the first time it is enclosed by edges and triangles), the birth of the larger hole, and finally the death of the smaller hole where it gets covered at $\varepsilon = 74.0$ (*Image from (Pokorny et al., 2014).*)

furthermore keep track of the associated change of basis by recording the column operations, by defining $V = I_n$ to be an identity matrix of size n, and initialising the algorithm with $R = MV$. For each left-to right column addition $M_j \leftarrow M_j + M_i$ in R for $i < j$, we perform the corresponding column addition $V_j \leftarrow V_j + V_i$. This algorithm terminates when $R = MV$ is reduced, and we have V as the change of basis matrix relating R to its unreduced version M. Using the reduced form, one then defines (Chen and Kerber, 2011)

$$P_R = \{(i,j) : R_j \neq 0 \text{ and } i = low(R_j)\},$$
$$E_R = \{i : R_i = 0 \text{ and } low(R_j) \neq i \text{ for all } j \in \{1,\dots,n\}\},$$

where P_R is the set of *persistent pairs*, such that each $(i,j) \in P_R$ corresponds to the persistence interval $(f(\sigma_i), f(\sigma_j))$ which is generated by the cycle R_j and 'killed' with

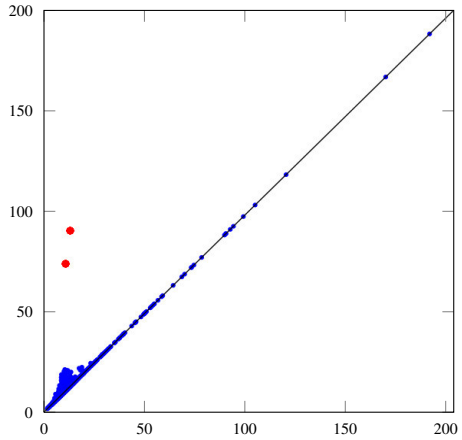


Figure A.9: The persistence diagram of $H_1(DC_\varepsilon)$. The two marked red points $p_1 = (10.58, 74.0)$, $p_2 = (12.96, 90.38)$ with large persistence, measured by the distance to the diagonal, correspond to the two holes in the original space in Fig. A.8 and their lifetime as ε changes. The other blue points are less important features (noise) that are created then killed quickly (*Image from (Pokorny et al., 2014).*)

the introduction of the simplex σ_i. On the other hand, E_R is the set of *essential cycles*, such that each $i \in E_R$ corresponds to a cycle V_i which was not killed by any simplex σ_j for all $j \in \{1, \ldots, n\}$, and hence, is keeps non-trivial in the final filtration $\mathcal{K}_n$. Proofs can be found in (Edelsbrunner and Harer, 2010).

Appendix B

RoboCup 2D Simulation League

B.1 RoboCup

RoboCup (Kitano et al., 1997) is a standard benchmark and an international competition for robot research and artificial intelligence. Started in 1997 with a simulated soccer competition, it evolved through the years to comprise six different soccer leagues, ranging from simulation to adult-size humanoids, in addition to RoboCup@Home for service robotics and RoboCup Rescue for rescue robotics, among others (RoboCup).

Seen as the grand challenge for the field of robotics, RoboCup aims to take the research to a stage where a robot soccer team can beat the human champions in their game in 2050 (Asada and Kitano, 1999).

B.2 RoboCup 2D Simulation League

RoboCup 2D Simulation League was the first league devised in RoboCup, in which two teams of 11 simulated 'dots' compete in a 2-dimensional football match through a central simulator called the Soccer Server (Noda and Matsubara, 1996) over a network.

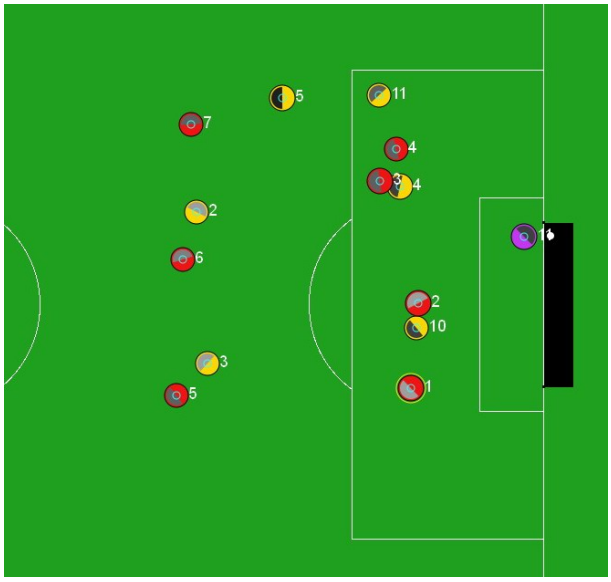


Figure B.1: RoboCup 2D Simulation League.

Each agent is an autonomous entity, connected to the server through a separate socket. The agent receives relative sensory information from the server (visual, acoustic and physical), all corrupted with noise. Then, the agent sends commands (like 'dash', 'turn', and 'kick') to the server with appropriate parameters. The server collects all input from all agents and updates the game.

The importance of this league is its focus on decision making which packs many interesting features that arise in many real world applications, while abstracting away complications from issues related to physical robots, like dexterity and balance. These features include multi-agent cooperative interaction, learning within uncertain and dynamic environments, and layered decision making from strategic thinking and planning to real-time response and reactive action.[1]

Extensions and specialisations of the Soccer Server appeared later to research techniques for more specific contexts than the full game (*e.g.* reinforcement learning in the *Keepaway* framework (Stone et al., 2005)).

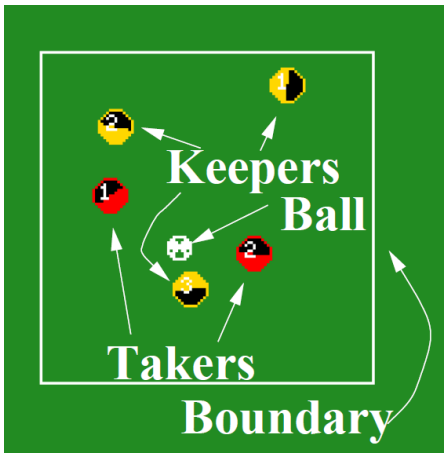


Figure B.2: Keepaway is built on the Soccer Server to research reinforcement learning (Image from (Stone and Sutton, 2001))

[1]The author participated in the 2D Simulation League in RoboCup 2011, Istanbul, Turkey, with team *Edinferno-2D* (Hawasly and Ramamoorthy, 2011).

Bibliography

P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*. ACM, 2004.

B.D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

M. Asada and H. Kitano. The RoboCup challenge. *Robotics and Autonomous Systems*, 29(1):3–12, 1999.

M. Asadi and M. Huber. Autonomous subgoal discovery and hierarchical abstraction for reinforcement learning using Monte Carlo method. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1588, 2005.

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

A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

U. Bauer and H. Edelsbrunner. The morse theory of Čech and delaunay filtrations. In *Proceedings of the 30th Annual Symposium on Computational Geometry*, pages 484:484–484:490. ACM, 2014.

U. Bauer, M. Kerber, and J. Reininghaus. Distributed computation of persistent homology. *Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 31–38, 2014.

R. Bellman. A Markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

D.C. Bentivegna. *Learning from observation using primitives*. PhD thesis, Georgia Institute of Technology, 2004.

S. Bhattacharya, V. Kumar, and M. Likhachev. Search-based path planning with homotopy class constraints. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, Atlanta, Georgia, 2010.

S. Bhattacharya, M. Likhachev, and V. Kumar. Identification and representation of homotopy classes of trajectories for search-based path planning in 3D. In *Proceedings of Robotics: Science and Systems (R:SS)*, 2011.

S. Bhattacharya, D. Lipsky, R. Ghrist, and V. Kumar. Invariants for homology classes with application to optimal search and planning problem in robotics. *Annals of Mathematics and Artificial Intelligence*, 67(3-4):251–281, 2013.

A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Survey: Robot programming by demonstration. Handbook of Robotics, Chapter 59, 2008.

S. Bitzer, I. Havoutis, and S. Vijayakumar. Synthesising novel movements through latent space modulation of scalable control policies. In *From Animals to Animats 10*, Lecture Notes in Computer Science, pages 199–209. Springer, 2008.

J.A. Boyan. Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pages 49–56, 1999.

R.I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 3:213–231, 2003.

E. Brochu, V.M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

O. Brock and O. Khatib. Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 550–555. IEEE, 2000.

R.A. Brooks. How to build complete creatures rather than isolated cognitive simulators. *Architectures for intelligence*, pages 225–239, 1991.

R.R. Burridge, A.A. Rizzi, and D.E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research (IJRR)*, 18(6):534–555, 1999.

S. Calinon, F. Guenter, and A. Billard. On learning, representing and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B. Special Issue on Robot Learning by Observation, Demonstration and Imitation*, 37(2):286–298, 2007.

G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2): 255–308, 2009.

P. Castro and D. Precup. Automatic construction of temporally extended actions for MDPs using bisimulation metrics. *Recent Advances in Reinforcement Learning, Lecture Notes in Computer Science*, pages 140–152, 2012.

CGAL. Computational Geometry Algorithms Library. `http://www.cgal.org`, 2014.

C. Chen and M. Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, 2011.

F. Chen, S. Chen, Y. Gao, and Z. Ma. Connect-based subgoal discovery for options in hierarchical reinforcement learning. In *Proceedings of the 3rd International Conference on Natural Computation (ICNC)*, volume 4, pages 698–702. IEEE, 2007.

T. Chen, M. Ciocarlie, S. Cousins, P.M. Grice, K. Hawkins, K. Hsiao, C. Kemp, C-H. King, D. Lazewatsky, A.E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama. Robots for humanity: A case study in assistive mobile manipulation. *IEEE Robotics & Automation Magazine, Special issue on Assistive Robotics*, 20, 2013.

T. Croonenborghs, K. Driessens, and M. Bruynooghe. Learning relational options for inductive transfer in relational reinforcement learning. In *Inductive Logic Programming*, pages 88–97. Springer, 2008.

A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

R. Diankov and J. Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, 2008.

T.G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13(1), 1999.

B.L. Digney. Learning hierarchical control structures for multiple tasks and changing environments. In *Proceedings of the 5th international Conference on Simulation of Adaptive Behavior*, volume 5, pages 321–330, 1998.

F. Dörfler, J.W. Simpson-Porco, and F. Bullo. Breaking the hierarchy: Distributed control and economic optimality in microgrids. *arXiv preprint arXiv:1401.1767*, 2014.

C. Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research (JAIR)*, 16: 59–104, 2002.

H. Edelsbrunner. The union of balls and its dual shape. *Discrete and Computational Geometry*, 13(1):415–440, 1995.

H. Edelsbrunner and J. Harer. Persistent homology - a survey. *Contemporary Mathematics*, 453:257–282, 2008.

H. Edelsbrunner and J.L. Harer. *Computational topology: an introduction*. AMS Bookstore, 2010.

Y. Engel and M. Ghavamzadeh. Bayesian policy gradient algorithms. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*, volume 19, page 457. MIT Press, 2006.

D. Ernst, P. Geurts, L. Wehenkel, and M.L. Littman. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 6(4), 2005.

F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 720–727. ACM, 2006.

D. Foster and P. Dayan. Structure in the space of value functions. *Machine Learning (MLJ)*, 49(2):325–346, 2002.

J. Ginebra and M.K. Clayton. Response surface bandits. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 771–784, 1995.

S. Girgin, F. Polat, and R. Alhajj. Improving reinforcement learning by using sequence trees. *Machine Learning (MLJ)*, 81(3):283–331, 2010.

J.C. Gittins and D. Jones. A dynamic allocation index for the discounted multiarmed bandit problem. *Progress in Statistics*, pages 241–266, 1974.

S. Goel and M. Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference*. AAAI, 2003.

A. Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.

M. Hawasly and S. Ramamoorthy. Edinferno. 2D team description paper for RoboCup 2011 2D soccer simulation league, 2011.

M. Hawasly and S. Ramamoorthy. Task variability in autonomous robots: Offline learning for online performance. In *5th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS)*, 2012.

M. Hawasly and S. Ramamoorthy. Lifelong learning of structure in the space of policies. In *AAAI Spring Symposium on Lifelong Machine Learning*, 2013a.

M. Hawasly and S. Ramamoorthy. Lifelong transfer learning with an option hierarchy. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1341–1346. IEEE, 2013b.

B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pages 243–250, 2002.

R.A. Howard. *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*, volume 2. Courier Dover Publications, 1971.

L. Jaillet and T. Simon. Path deformation roadmaps. In *Algorithmic Foundation of Robotics VII*, volume 47, pages 19–34. Springer Berlin Heidelberg, 2008.

L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research (JAIR)*, 1996.

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research (IJRR)*, 30(7):846–894, 2011.

L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

S. Kazemitabar and H. Beigy. Automatic discovery of subgoals in reinforcement learning using strongly connected components. In *Proceedings of the conference on Advances in Neural Information Processing (NIPS)*, pages 829–834. Springer Berlin Heidelberg, 2009.

S. Kim, K. Sreenath, S. Bhattacharya, and V. Kumar. Optimal trajectory generation under homology class constraints. In *IEEE Conference on Decision and Control*, 2012.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The robot world cup initiative. In *Proceedings of the first International Conference on Autonomous Agents*, pages 340–347. ACM, 1997.

J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379, 2012.

G. Konidaris and A.G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 895–900, 2007.

G. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in Neural Information Processing Systems (NIPS)*, 22:1015–1023, 2009.

G. Konidaris, S. Kuindersma, A.G Barto, and R. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trajectories. *Advances in Neural Information Processing Systems (NIPS)*, 23:1162–1170, 2010.

G. Konidaris, S. Kuindersma, R. Grupen, and A.G Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research (IJRR)*, 31(3):360–375, 2012a.

G. Konidaris, I. Scheidwasser, and A.G. Barto. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research (JMLR)*, 13:1333–1371, 2012b.

O. Kozlova, O. Sigaud, and C. Meyer. Automated discovery of options in factored reinforcement learning. In *Proceedings of the ICML/UAI/COLT Workshop on Abstraction in Reinforcement Learning*, 2009.

R.M. Kretchmar, T. Feil, and R. Bansal. Improved automatic discovery of subgoals for options in hierarchical reinforcement learning. *Journal of Computer Science & Technology*, 3, 2003.

H. Kretzschmar, M. Kuderer, and W. Burgard. Learning to predict trajectories of cooperatively navigating agents. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

M. Kuderer, C. Sprunk, H. Kretzschmar, and W. Burgard. Online generation of homotopically distinct navigation paths. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

T.L. Lai and H. Robbins. Adaptive design in regression and control. *Proceedings of the National Academy of Sciences*, 75(2):586–587, 1978.

S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, pages 45–73. Springer, 2012.

S.M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

S.M. LaValle and J.J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

A. Lazaric. *Knowledge transfer in reinforcement learning*. PhD thesis, Politecnico di Milano, 2008.

S.R. Lindemann and S.M. LaValle. Current issues in sampling-based motion planning. In *Robotics Research*, pages 36–54. Springer, 2005.

S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence (AIJ)*, 55(23):311 – 365, 1992.

M.M.H. Mahmud, M. Hawasly, B. Rosman, and S. Ramamoorthy. Clustering Markov decision processes for continual transfer. *arXiv preprint arXiv:1311.3959*, 2013.

V. Manfredi and S. Mahadevan. Hierarchical reinforcement learning using graphical models. In *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, pages 39–44, 2005.

S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, page 71. ACM, 2004.

O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 570–576, 1998.

E. Masehian and D. Sedighizadeh. Classic and heuristic approaches in robot motion planning - a chronological review. *World Academy of Science, Engineering and Technology*, 23:101–106, 2007.

A. McGovern. acquire-macros: An algorithm for automatically learning macro-actions. In *NIPS98 Workshop on Abstraction and Hierarchy in Reinforcement Learning*, 1998.

A. McGovern and A.G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *Computer Science Department Faculty Publication Series*, page 8, 2001.

N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning (MLJ)*, 73(3):289–312, 2008a.

N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 648–655. ACM, 2008b.

I. Menache, S. Mannor, and N. Shimkin. Q-cut: dynamic discovery of sub-goals in reinforcement learning. *European Conference on Machine Learning (ECML)*, pages 187–195, 2002.

A.J. Mersereau, P. Rusmevichientong, and J.N. Tsitsiklis. A structured multiarmed bandit problem and the greedy policy. *IEEE Transactions on Automatic Control*, 54 (12):2787–2802, 2009.

A. Nilim and L.E. Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, pages 780–798, 2005.

J. Niño-Mora. Computing a classic index for finite-horizon bandits. *INFORMS Journal on Computing*, 23(2):254–267, 2011.

P. Niyogi, S. Smale, and S. Weinberger. Finding the homology of submanifolds with high confidence from random samples. *Discrete and Computational Geometry*, 39 (1-3):419–441, 2008.

I. Noda and H. Matsubara. Soccer server and researches on multi-agent systems. In *Proceedings of the IROS-96 Workshop on RoboCup*, 1996.

S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research (IJRR)*, 29(8):1053–1068, 2010.

S. Pandey, D. Chakrabarti, and D. Agarwal. Multi-armed bandit problems with dependent arms. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 721–728. ACM, 2007.

R.E. Parr. *Hierarchical control and learning for Markov decision processes*. PhD thesis, University of California, 1998.

R.E. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems (NIPS)*, pages 1043–1049, 1998.

T.J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. *Technical Report, University of Massachusetts, Amherst, MA, USA*, 1999.

J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS International Conference on Humanoid Robots*, pages 1–20, 2003.

J. Peters, S. Vijayakumar, and S. Schaal. Natural actor-critic. In *Machine Learning (MLJ)*, pages 280–291. Springer, 2005.

M. Pickett and A.G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pages 506–513, 2002.

F.T. Pokorny, M. Hawasly, and S. Ramamoorthy. Multiscale topological trajectory classification with persistent homology. In *Proceedings of Robotics: Science and Systems (R:SS)*, 2014.

W.B. Powell. The knowledge gradient for optimal learning. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

A.A. Rad, M. Hasler, and P. Moradi. Automatic skill acquisition in reinforcement learning using connection graph stability centrality. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 697–700. IEEE, 2010.

K. Regan and C. Boutilier. Robust policy computation in reward-uncertain mdps using nondominated policies. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, 2010.

Rethink Robotics. R.A. Brooks. `http://www.rethinkrobotics.com/products/baxter/`, 2012.

RoboCup. The RoboCup Federation. `http://www.robocup.org/`.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Robotics Research*, pages 561–572. Springer, 2005.

S. Schaal, P. Mohajerian, and A. Ijspeert. Dynamics systems vs. optimal control - a unifying view. *Progress in Brain Research*, 165:425–445, 2007.

B. Da Silva, G. Konidaris, and A.G. Barto. Learning parameterized skills. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1679–1686. Omnipress, 2012.

Ö. Şimşek and A.G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, page 95. ACM, 2004.

Ö. Simşek and A.G. Barto. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

Ö. Şimşek, A.P. Wolfe, and A.G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 816–823. ACM, 2005.

S. Singh, A.G. Barto, and N. Chentanez. *Intrinsically motivated reinforcement learning*. Defense Technical Information Center, 2005.

M. Snel and S. Whiteson. Multi-task reinforcement learning: shaping and feature selection. *Recent Advances in Reinforcement Learning, Lecture Notes in Computer Science*, pages 237–248, 2012.

V. Soni and S. Singh. Reinforcement learning of hierarchical skills on the Sony Aibo robot. In *Proceedings of the Fifth International Conference on Development and Learning (ICDL)*, 2006.

N. Srinivas, A. Krause, S.M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

M. Stolle and D. Precup. Learning options in reinforcement learning. *Abstraction, Reformulation, and Approximation, Lecture Notes in Computer Science*, 2371:212–223, 2002.

P. Stone. *Layered learning in multiagent systems: A winning approach to robotic soccer*. MIT Press, 2000.

P. Stone and R.S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

P. Stone, R.S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

M. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 943–950, 2000.

F.T. Sunmola and J.L. Wyatt. Model transfer for Markov decision tasks via parameter matching. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, 2006.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Cambridge Univ Press, 1998.

R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence (AIJ)*, 112 (1):181–211, 1999.

M. Tambe, A.X. Jiang, B. An, and M. Jain. Computational game theory for security: Progress and challenges. In *AAAI Spring Symposium on Applied Computational Game Theory*, 2013.

F. Tanaka and M. Yamamura. Multitask reinforcement learning on the distribution of MDPs. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 3, pages 1108–1113. IEEE, 2003.

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

R.L. Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. *Robotics: Science and Systems (R:SS)*, 2009.

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 385–392, 1995.

C. Urmson, J. Anhalt, J.A. Bagnell, C.R. Baker , R.E. Bittner, J.M. Dolan, D. Duggins, D. Ferguson , T. Galatali, H. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, A. Kelly, D. Kohanbash, M. Likhachev, N. Miller, K. Peterson, R. Rajkumar, P. Rybski, B. Salesky, S. Scherer, Y. Seo, R. Simmons, S. Singh, J.M. Snider, A. Stentz, W.L. Whittaker, and J. Ziglar. Tartan racing: A multi-modal approach to the DARPA Urban Challenge. Technical Report CMU-RI-TR-, Robotics Institute, 2007.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning (MLJ)*, 8(3-4):279–292, 1992.

A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: a hierarchical Bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 1015–1022. ACM, 2007.

A. Wilson, A. Fern, and P. Tadepalli. Transfer learning in sequential decision problems: A hierarchical Bayesian approach. In *ICML 2011 Unsupervised and Transfer Learning Workshop. JMLR: Workshop and Conference Proceedings*, 2012.

D. Wingate, N.D. Goodman, D.M. Roy, L.P. Kaelbling, and J.B. Tenenbaum. Bayesian policy search with policy priors. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1565–1570. AAAI Press, 2011.

P. Zang, P. Zhou, D. Minnen, and C. Isbell. Discovering options from example trajectories. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 1217–1224. ACM, 2009.