Learning the Structure of Continuous Markov Decision Processes

von Jan Hendrik Metzen

Dissertation

zur Erlangung des Grades eines Doktors der Naturwissenschaften
- Dr. rer. nat.-

Vorgelegt im Fachbereich 3
(Mathematik & Informatik)
der Universität Bremen
im Dezember 2013

Datum des Promotionskolloquiums: 21. Februar 2014

Gutachter: Prof. Dr. Frank Kirchner (Universität Bremen)
Prof. Dr. Hans-Jörg Kreowski (Universität Bremen)

Dedicated to my family

Zusammenfassung

Künstliche, intelligente Agenten, die über längere Zeitspannen autonom in komplexen Umgebungen agieren und eine Vielzahl unterschiedlicher Aufgaben lösen können, sind von großem Interesse. Solche Agenten werden während ihrer „Lebensspanne“ häufig mit Problemstellungen konfrontiert sein, die zum Zeitpunkt ihrer Entwicklung nicht absehbar waren. Aufgrund dessen ist die Fähigkeit, lebenslang neue Verhaltensweisen erlernen zu können, eine wichtige Voraussetzung für diese Art von Agenten, da sie es ihnen ermöglicht, mit unvorhergesehenen Situationen umzugehen.

Es wäre allerdings für einen Agenten sehr aufwändig, jede komplexe Verhaltensweise von Grund auf neu zu erlernen. Es ist naheliegender, Verhalten als modular aufzufassen und den Agenten eine Reihe grundlegender Fähigkeiten erlernen zu lassen, die als wiederverwendbare Verhaltensbausteine dienen können. Solche Fähigkeiten können, nachdem sie erworben wurden, ein schnelleres Erlernen von Verhalten und Adaptieren an neue Situationen ermöglichen. Der Schwerpunkt dieser Arbeit liegt auf algorithmischen Ansätzen für den Erwerb von Fähigkeiten, insbesondere darauf, welche Fähigkeiten erworben werden sollen und wie der Erwerb dieser Fähigkeiten vonstatten gehen kann. Das Erstere wird als „Entdeckung von Fähigkeiten“ und das Letzere als „Erlernen von Fähigkeiten“ bezeichnet.

Der Hauptbeitrag dieser Arbeit ist ein neuer, inkrementeller Ansatz für die Entdeckung von Fähigkeiten, der für lebenslanges Lernen geeignet ist. In diesem Ansatz lernt der Agent zunächst inkrementell eine Repräsentation seiner Umgebung in Form eines Graphen und nutzt dann gewisse Eigenschaften dieses Graphen wie dessen „Flaschenhälse“ als Grundlage für die Entdeckung von Fähigkeiten. Diese Arbeit schlägt einen neuen Ansatz für das Erlernen solcher Graph-basierter Repräsentationen vor, der auf einem probabilistischen, generativen Modell basiert. Des Weiteren wird ein neuer inkrementeller Clustering-Ansatz für die Identifizierung von Flaschenhälsen in solchen Graphen vorgestellt.

Darauf aufbauend wird in der Arbeit ein neues intrinsisches Motivationssystem vorgeschlagen, das es einem Agent erlaubt, seine Zeit dynamisch zwischen der Entdeckung und dem Erlernen von Fähigkeiten einzuteilen. Dieses Motivationssystem zielt auf Szenarien ab, in welchen die Handlungsfreiheit des Agenten nicht durch die Pflicht zur Erfüllung externer Aufgaben eingeschränkt wird. Die Ergebnisse der Arbeit zeigen, dass der resultierende Ansatz für den Erwerb von Fähigkeiten für kontinuierliche Domänen geeignet ist und mit Stochastizität und variierendem explorativem Verhalten eines Agenten umgehen kann. Die erworbenen Fähigkeiten sind wiederverwendbar und vielseitig und können zum zeitgleichen Lösen verschiedener Aufgaben und für lebenslanges Lernen genutzt werden.

Abstract

There is growing interest in artificial, intelligent agents which can operate autonomously for an extended period of time in complex environments and fulfill a variety of different tasks. Such agents will face different problems during their lifetime which may not be foreseeable at the time of their deployment. Thus, the capacity for lifelong learning of new behaviors is an essential prerequisite for this kind of agents as it enables them to deal with unforeseen situations.

However, learning every complex behavior anew from scratch would be cumbersome for the agent. It is more plausible to consider behavior to be modular and let the agent acquire a set of reusable building blocks for behavior, the so-called skills. These skills might, once acquired, facilitate fast learning and adaptation of behavior to new situations. This work focuses on computational approaches for skill acquisition, namely which kind of skills shall be acquired and how to acquire them. The former is commonly denoted as "skill discovery" and the latter as "skill learning".

The main contribution of this thesis is a novel incremental skill acquisition approach which is suited for lifelong learning. In this approach, the agent learns incrementally a graph-based representation of a domain and exploits certain properties of this graph such as its bottlenecks for skill discovery. This thesis proposes a novel approach for learning a graph-based representation of continuous domains based on formalizing the problem as a probabilistic generative model. Furthermore, a new incremental agglomerative clustering approach for identifying bottlenecks of such graphs is presented.

Thereupon, the thesis proposes a novel intrinsic motivation system which enables an agent to intelligently allocate time between skill discovery and skill learning in developmental settings, where the agent is not constrained by external tasks. The results of this thesis show that the resulting skill acquisition approach is suited for continuous domains and can deal with domain stochasticity and different explorative behavior of the agent. The acquired skills are reusable and versatile and can be used in multi-task and lifelong learning settings in high-dimensional problems.

Acknowledgements

Writing a PhD thesis is—at least nowadays—not an endeavor which an author can accomplish in isolation; rather it is a work conducted by the author but promoted by its environment, both its professional and its private one. For this, I would like to express my gratitude to all the people that have supported me.

First and foremost, I would like to thank my adviser Prof. Dr. Frank Kirchner, who gave me the opportunity of working in his group in the last couple of years, both as part of the DFKI RIC and the Robotics Group of the University Bremen. Without his continuous encouragement for thinking and rethinking my ideas, for reaching beyond the borders of a specific field in order to see the overall picture, and for aiming at general approaches rather than specific solutions, this thesis would not have become what it is now. Equally important is the vivid and ambitious but at the same time constructive atmosphere his lab has always exhibited during the years of writing this thesis. The annual PhD retreats of the group have been immensely helpful and I am glad that my adviser took each year a whole week time for this.

I am also grateful for all the help, comments, and criticisms expressed by my colleagues during the last years. In particular, I would like to thank my dear colleague Dr. Yohannes Kassahun, who has been a continuous source of inspiration during my years in Bremen, both in scientific as well as philosophical topics. He has supported me ever since my first day in Bremen, from writing the first scientific papers to finishing this thesis. Reading this work entirely and giving uncounted helpful comments and suggestions for refining it has been an invaluable help. I would also like to thank my colleagues from the project BesMan, in particular Elsa Andrea Kirchner, for several discussions on the biological principles of skill acquisition and for shedding light onto the different usages of the term skill in biology, psychology, and reinforcement learning. I also gratefully acknowledge the partial funding of this work as a part of the project BesMan under a grant of the German Federal Ministry of Economics and Technology (BMWi, FKZ 50 RA 1217).

The progress of this thesis has been presented five times on the annual PhD retreats of the DFKI RIC in the years 2009-2013 on Spiekeroog, Wangerooge, and Juist. I would like to thank the other PhD students of the group and the postdoctoral volunteers for their helpful feedback during these retreats. In particular, I would like to thank Dr. José de Gea Fernández and Dr. Jan Albiez for their feedback. Equally important was the feedback of the external scientific community. I would like to express my gratitude to all reviewers for their constructive and mostly balanced reviews of my

papers and to the journal editors and conference organizers for all their efforts.

Quite important for this work has also been the software stack provided by the open source community. The algorithms proposed in this thesis have been implemented in the Python programming language, based on the packages NumPy, SciPy, scikit-learn, and NetworkX. The figures of this thesis have been created using matplotlib and Inkscape. The open source frameworks pySPACE and MMLF, which are developed by the Robotics Group of the University Bremen and the DFKI RIC, have been of great value for automating the empirical evaluation of the proposed approaches. Using LaTeX for typesetting has simplified writing this thesis and the corresponding papers immensely. GNU/Linux has served as a stable operating system that did not "get in the way" too frequently.

Without the help of all these people (and many further ones which cannot all be mentioned here explicitly), this thesis would not have been possible. Since many of their suggestions, comments, and criticisms have contributed to the approaches presented in this thesis, may it be directly or indirectly, I found it appropriate to use the plural "we" rather than the singular "I" during the presentation of the thesis' main contributions.

Last but not least I would to express my gratitude to my family for their support. I would like to thank my wife Silke for her aid during the last years and for never losing patience when the undertaking of writing this thesis became ever longer than expected over the years. My parents and my sister have always supported and encouraged me, starting from my years in school over my diploma studies in Münster to my time in Bremen. The same is true for my grandfather and Gerda, who were always interested in the content and progress of my work and encouraged me to finish it. Without such a balanced and supportive private environment, writing this thesis would have been much harder.

Jan Hendrik Metzen Bremen, December 2013

Contents

Page

CHAPTER 1 INTRODUCTION **1**
- 1.1 Motivation . . . 1
- 1.2 Goal . . . 5
- 1.3 Structure . . . 6
- 1.4 Contributions . . . 8

CHAPTER 2 BACKGROUND **11**
- 2.1 Introduction . . . 11
- 2.2 Decision Making and Optimality . . . 11
  - 2.2.1 Sequential Decision Problems . . . 11
  - 2.2.2 Markov Decision Processes . . . 13
  - 2.2.3 Optimal Behavior . . . 14
  - 2.2.4 Value Function and Bellman Equations . . . 15
- 2.3 Reinforcement Learning . . . 16
  - 2.3.1 Temporal Difference Learning . . . 17
  - 2.3.2 Function Approximation . . . 19
  - 2.3.3 Direct Policy Search . . . 21
  - 2.3.4 Hierarchical Reinforcement Learning . . . 21
- 2.4 Graph Theory . . . 26
  - 2.4.1 Basics . . . 26
  - 2.4.2 Graph Cuts . . . 27
  - 2.4.3 Graph Clustering . . . 28

CHAPTER 3 SKILL DISCOVERY **33**
- 3.1 Introduction . . . 33
  - 3.1.1 Illustration: Towers of Hanoi . . . 34
- 3.2 Considerations . . . 35
  - 3.2.1 Potential Benefits . . . 35
  - 3.2.2 Challenges . . . 40
  - 3.2.3 Desirable Properties . . . 42
- 3.3 Related Works . . . 43
  - 3.3.1 Solution-Based Heuristics . . . 43
  - 3.3.2 Factoring-Based Heuristics . . . 46

3.3.3 Frequency-Based Heuristics . . . 47
3.3.4 Graph-Based Heuristics . . . 47
3.3.5 Meta Heuristics . . . 50
3.3.6 Summary and Discussion . . . 50
3.4 Performance Evaluation . . . 53
3.4.1 Performance Metrics . . . 53
3.4.2 Baselines . . . 55

CHAPTER 4 INCREMENTAL GRAPH-BASED SKILL DISCOVERY **57**
4.1 Introduction . . . 57
4.2 Methods . . . 58
4.2.1 Transition Graph Generation . . . 58
4.2.2 Linkage Criteria and Bottlenecks . . . 60
4.2.3 Incremental Graph Clustering . . . 61
4.2.4 Skill Prototype Generation . . . 67
4.3 Results . . . 69
4.3.1 Graph Clustering . . . 69
4.3.2 Bottleneck Criterion . . . 71
4.3.3 Graph Smoothing . . . 72
4.3.4 Edge Weights . . . 74
4.3.5 Cluster Accordance Analysis . . . 75
4.3.6 Multi-task Learning . . . 78
4.4 Discussion . . . 82

CHAPTER 5 LEARNING GRAPH-BASED REPRESENTATIONS **85**
5.1 Introduction . . . 85
5.2 Graph-Based Skill Discovery in Continuous Domains . . . 86
5.2.1 Prior Work . . . 87
5.3 Methods . . . 88
5.3.1 Likelihood of Transition Graph . . . 89
5.3.2 `FIGE`: Force-Based Iterative Graph Estimation . . . 90
5.3.3 Skill Prototype Generation . . . 94
5.4 Results . . . 95
5.4.1 Graph Likelihood . . . 95
5.4.2 Skill Discovery . . . 96
5.5 Excursus: Representation Learning . . . 101
5.5.1 Method . . . 102
5.5.2 Illustration . . . 103
5.5.3 Evaluation . . . 104
5.6 Discussion . . . 108

CHAPTER 6 LIFELONG LEARNING AND INTRINSIC MOTIVATION **111**
6.1 Introduction . . . 111
6.2 Lifelong Learning and Intrinsic Motivation . . . 112

6.2.1 Lifelong Learning and Shaping . . . 112
6.2.2 Intrinsic Motivation . . . 113
6.2.3 Related Work . . . 114
6.3 Methods . . . 116
6.3.1 Agent Architecture . . . 116
6.3.2 Incremental Graph-Based Skill Discovery . . . 118
6.3.3 Intrinsic Motivation . . . 121
6.4 Results . . . 123
6.4.1 2D Multi-Valley . . . 123
6.4.2 Octopus Arm . . . 128
6.5 Discussion . . . 131

CHAPTER 7 CONCLUSION AND OUTLOOK **133**
7.1 Summary . . . 133
7.2 Insights . . . 134
7.3 Outlook . . . 136
7.4 Closing Words . . . 138

APPENDIX A DERIVATIONS **139**
A.1 Derivation of Transition Graph Weights . . . 139
A.2 Derivation of FIGE's Update Equations . . . 140

LIST OF FIGURES **143**

LIST OF ALGORITHMS **145**

LIST OF SYMBOLS **147**

BIBLIOGRAPHY **151**

1 Introduction

"The beginning is perhaps more difficult than anything else, but keep heart, it will turn out all right."

Vincent van Gogh, letter to Theo van Gogh, 1873

1.1 Motivation

EMBODIED agents like robots are used in increasingly complex, real-world domains, such as domestic environments (Iocchi et al., 2012), health care (Okamura et al., 2010), and extraterrestrial settings (Grotzinger et al., 2012). These environments are often unstructured, populated by humans, and changing over time. At the same time, robots are becoming increasingly sophisticated, both in terms of their hardware and their control software, see, e.g., Lemburg et al. (2011) and Bartsch et al. (2012). A simple, reactive control approach (Brooks, 1986) is not sufficient for these systems as it lacks the ability to predict and control the environment on larger scales of time and space. For this, agents must be able to build up both procedural and declarative knowledge[1] about the world and store this knowledge in a convenient way so that it can be reused and adapted easily. This requires robotic control architectures which allow learning, utilization, combination, integration, and adaptation of procedural and declarative knowledge. A multitude of robot control architectures has been proposed over the last years (see Murphy (2000) for a discussion and an overview). Figure 1.1 presents one example of a 3-layered, "hybrid" control architecture.

The focus of this work is one specific aspect of such a robotic control architecture: the learning and generation of complex behavior. Complex behavior is considered to be goal-directed, hierarchically organized, and based on generalization, transfer, and analogy (Oudeyer et al., 2007). One building block for complex behavior is a set of versatile, reusable *skills*. Skills are *procedural knowledge* of an agent (de Jong and Ferguson-Hessler, 1996), i.e., they are a kind of knowledge that can be utilized directly

[1] While declarative knowledge refers to knowledge about facts in the world, i.e., *that* something is the case, procedural knowledge denotes knowledge of *how* to perform some task in a close-to-optimal way.

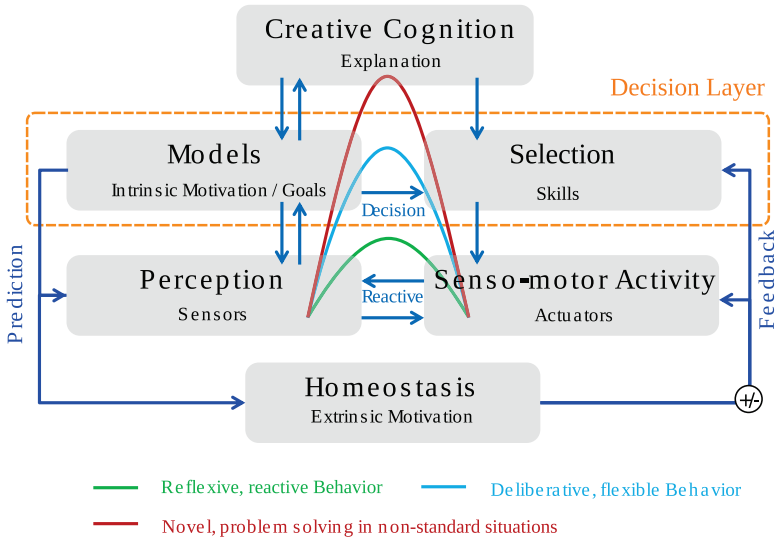


Figure 1.1 – *A 3-layer robot control architecture.* The lowest, "reactive" layer consists of hard-wired connections between sensory stimuli and motor commands. This layer allows fast and direct reaction to sensory input. The middle, "decision" layer contains learned predictive models of senso-motoric relations, which are used amongst other for internal generation of goals. Based on these self-generated goals and intrinsic motivation mechanisms, more complex, adaptive behavior is generated based on a hierarchy of acquired skills. The upper, "creative" layer of the architecture becomes important in novel situations, where the agent discovers new correlations in its sensory input, or when internal models are wrong. In this case, the creative layer may allow compensating for the errors by adapting predictive models and modifying behavior generation. Moreover, extrinsic motivation is continuously generated based on homeostatic need regulation and prediction of fitness-enhancing events and provided as feedback to the reactive and decision layer, which may use this feedback to modify behavior generation. Please refer to Köhler et al. (2012) for more details.

for the performance of some task with a pre-determined result in a close-to-optimal way. Skills are not inborn but acquired during the lifetime of an individual. Skill acquisition (Speelman and Kirsner, 2005, Chapter 2) is a process that consists of the *discovery* of novel, useful skills, *learning* to perform these skills in an efficient and reliable manner, and being able to *utilize* the learned skills for the acquisition of more complex skills or for solving external tasks.

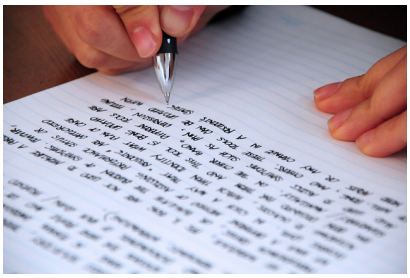
How can a set of reusable skills be acquired autonomously? Taking inspiration from humans, which can acquire skills of astounding sophistication (see Figure 1.2), one can observe that *humans develop in an autonomous open-ended manner through lifelong learning* (Oudeyer et al., 2007). Accordingly, open-ended, lifelong learning is also considered to be a key prerequisite for employing artificial systems like robots in complex, changing environments (Thrun and Mitchell, 1995; Silver et al., 2013). Yet, to date no artificial system is (by far) equipped with similar capacities as human beings in this regard. However, there is growing interest in several areas, most notably in the field of developmental robotics (Weng et al., 2001; Lungarella et al., 2003; Asada

(a) Girl learning to ride a bicycle.
Shaine Mata, CC BY-NC-ND 2.0 license

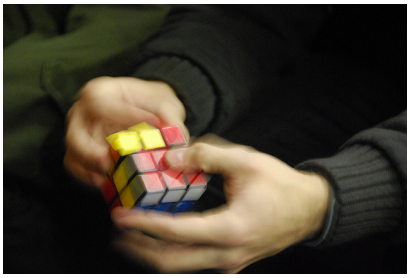
(b) 2.5 months old baby learns to grab objects.
`www.flickr.com/people/_-o-_/`
CC BY 2.0 license

(c) Woman making pottery.
Owen Lin, CC BY-NC-ND 2.0 license

(d) Man juggles.
`www.flickr.com/people/Dvortygirl`
CC BY-SA 2.0 license



(e) Handwriting a text.
Jeffrey J. Pacres, CC BY-NC-ND 2.0 license



(f) Rubik's cube is a game where experts players can use acquired skills to simplify a hard search problem considerably.
Steve Rhodes, CC BY-ND 2.0 license

Figure 1.2 – *Examples of acquired skills.* Acquired motor skills may enrich the scope of the learner's behavior (a,b), may serve as basis for an occupation (c,d), and may be the basis for more abstract, cognitive tasks (e,f). Examples are inspired by Figure 1.1 of Konidaris (2011).

et al., 2009), of providing artificial systems with means for development and lifelong learning. According to Oudeyer et al. (2007), development should be progressive and incremental as well as autonomous and active. Development being *progressive and incremental* means that first simple skills are acquired and, later on, successively more complex skills are learned based on the more simple skills that have already been acquired ("cumulative learning"). The development process is *autonomous and active* if the system decides on its own which kind of skill it tries to acquire at a certain stage of development rather than getting an externally shaped sequence of tasks.

For employing a developmental approach to skill acquisition in artificial agents, one needs to address how skill acquisition can be implemented in a way which exhibits these four properties identified by Oudeyer et al. (2007). For *autonomous* skill acquisition, the agent needs to decide on its own which skills shall be acquired, i.e., which capabilities are useful for it. This capacity is denoted as *skill discovery*. Dietterich (2000a) has concluded that the discovery of hierarchical structures such as skills is the biggest open problem in hierarchical reinforcement learning; it is also considered to be essential for lifelong learning (Hengst, 2002). While a multitude of computational skill discovery approaches has been proposed, most of these approaches do not allow progressive and incremental development, are tailored to small and discrete problem domains, or cannot be employed in a developmental setting[1] (see Section 3.3).

How can the acquisition of skills exhibit the desirable property of being *active*? This question is studied in the field of *intrinsic motivation* systems. The term "intrinsically motivated" stems from biology and one of its first appearances was in a paper by Harlow (1950) on the manipulation behavior of rhesus monkeys. According to Baldassarre (2011) "extrinsic motivations guide learning of behaviors that directly increase [evolutionary] fitness" while "intrinsic motivations drive the acquisition of knowledge and skills that contribute to produce behaviors that increase fitness only in a later stage." Intrinsic motivations contribute to learning not as a learning mechanism per se, but rather as a guiding mechanism which guides learning mechanisms to acquire behaviors that increase fitness. According to Baldassarre "[intrinsic motivations] drive organisms to continue to engage in a certain activity if their competence in achieving some interesting outcomes is improving, or if their capacity to predict, abstract, or recognise percepts is not yet good or is improving...". Thus, intrinsic motivation can be seen as an integral part of skill acquisition that determines which of the discovered skills shall be learned at a certain stage of development.

How can skills be acquired in an *incremental and progressive* way? The central idea investigated in this thesis is that *identifying and exploiting structure* of a problem domain allows the discovery of reusable, versatile skills in such an incremental and progressive manner in a developmental setting. This idea is motivated by a multitude of works which have shown that effective learning is only possible by exploiting structure and regularities of a problem. For instance, Ashby (1956) recognized that

[1] In a developmental setting, an artificial agent is granted a developmental period, in which it can act freely, before it is confronted with externally imposed tasks. In this developmental period, the agent may acquire a collection of skills that it can reuse later on for learning to solve external tasks more efficiently.

learning is worthwhile only when the environment shows some type of constraint such as reoccurring sub-structures. The reason why structure is also considered to be important for incremental and progressive skill discovery in developmental settings is that (a) structure itself can be identified incrementally and be refined later on and (b) the structure of a domain is independent of the task imposed onto an agent and can thus also be discovered in a developmental setting.

Besides being interesting on their own, artificial systems with the capacity for intrinsically motivated, lifelong learning of skills could also have a considerable practical impact. For instance, a future *domestic robot* endowed with these capacities might learn new skills during phases in which no explicit task is assigned to it. These novel skills might enable the robot to solve future tasks more efficiently or even to solve tasks that have been impossible to it originally. Similarly, robots employed in remote domains such as *extra-terrestrial exploration* missions could benefit considerable from the ability of lifelong learning: if such a remote robot would be confronted with a situation that was not foreseen by its engineers it could learn autonomously how to deal with it rather than requiring human mission operators to intervene. This can be crucial as manual intervention is already cumbersome for nowadays missions to, e.g., the planet Mars, but might become impractical for even more ambitious future missions that exhibit limited communication bandwidth and high delay between mission operators and robots.

1.2 GOAL

Section 1.1 has motivated the interest in artificial agents with the capacity for autonomous acquisition of novel skills. As discussed, skill acquisition in a lifelong learning agent should be progressive and incremental as well as autonomous and active. Moreover, such a skill acquisition module should be applicable in a broad range of domains, in particular in domains that are continuous and stochastic. To the author's best knowledge, no skill acquisition approach proposed in prior work satisfies all of these demands at the same time (see Section 3.3). The central idea of this thesis is that identifying and exploiting structure of a problem domain can give rise to such a skill acquisition approach. Accordingly, the goal of this thesis can be summarized as follows:

> **Goal:** *Develop an incremental, self-motivated approach for skill acquisition which is based on identifying and exploiting the structure of a problem and which can be used in developmental and lifelong learning settings in continuous and stochastic domains.*

Based on a study of prior works (see Section 3.3), graph-based approaches for skill discovery have been identified as promising candidates for achieving this goal since graph-based representations allow naturally to capture structure that is present in a problem. However, typical graph-based skill discovery approaches are neither incremental nor suited for continuous domains. Furthermore, many existing skill acquisition

approaches require that an external reward signal is provided and can thus not be employed in developmental and lifelong learning settings. Accordingly, for achieving the stated goal, three subgoals have been identified:

SUBGOAL S1 Develop an incremental, graph-based skill discovery approach that can identify skills at any time and allows an agent to acquire a collection of skills which increases in both size and sophistication over time.

SUBGOAL S2 Extend graph-based approaches for skill discovery to continuous domains.

SUBGOAL S3 Develop an intrinsic motivation mechanism that allows incremental skill acquisition in the absence of external reward signals, e.g., in developmental or lifelong learning settings.

This thesis restricts itself to domains which can be modeled as Markov Decision Processes (see Section 2.2.2). This allows defining, studying, and comparing methods in a solid and clearly defined theoretical framework.

1.3 STRUCTURE

Figure 1.3 shows the structure of this thesis. Chapter 2 and 3 provide the *fundamentals* of this thesis. Chapter 2 defines the problem class addressed in this thesis, defines what is considered as optimal behavior, and provides the required background in reinforcement learning and graph theory. Chapter 3 gives a motivation for learning skills and autonomous skill discovery, discusses advantages and disadvantages, and provides a review of related works. Moreover, baselines and metrics used throughout this thesis are defined.

Chapter 4, 5, and 6 contain the main *contributions* of this thesis. Chapter 4 addresses subgoal S1 by proposing `OGAHC`, a novel incremental skill discovery method which is based on learning a transition graph and is suited for discrete domains. Chapter 5 focuses on subgoal S2. For this, the new method `FIGE` is proposed which allows estimating transition graphs in continuous domains that capture the domain's dynamics well. A derivation of `FIGE` based on maximizing the likelihood of a set of observed transitions is given in Appendix A.2. Chapter 6 provides an incremental version of `FIGE` and an extension of `OGAHC` to continuous domains. Furthermore, subgoal S3 is addressed by proposing means for intrinsic motivation which allow an agent to trade-off skill discovery and learning in developmental or lifelong learning settings. Chapter 7 provides a *conclusion* of this thesis by summarizing its main contributions and insights, and a *discussion* of open problems and future work.

An overview over all figures of the thesis is given on page 143 and a summary of all algorithms on page 145. A list of the symbols used throughout this thesis is provided on page 147.

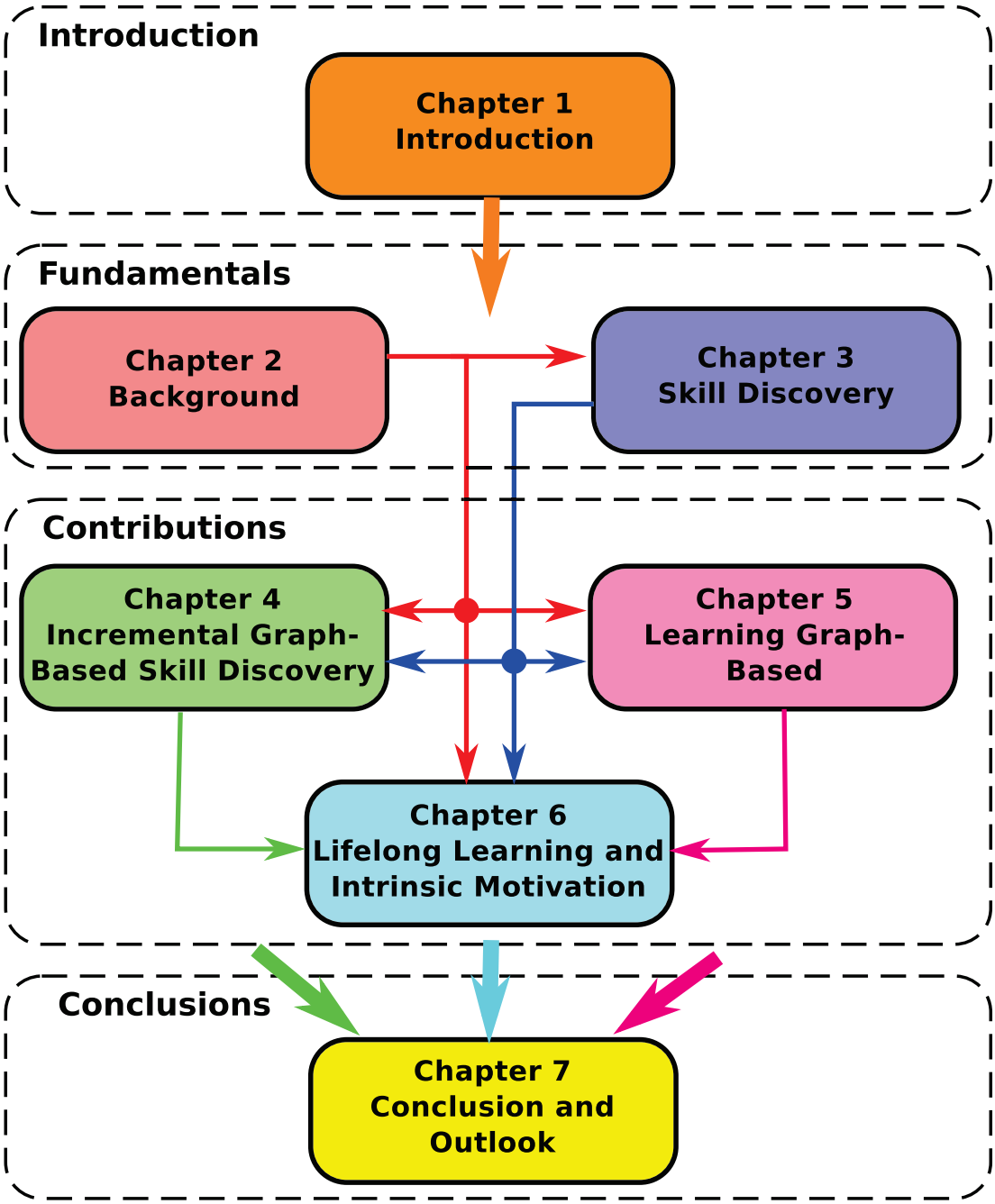


Figure 1.3 – *Schematic illustration of the thesis structure.* Arrows from one chapter to another one indicate that the former chapter provides background for the later one.

1.4 CONTRIBUTIONS

This thesis consists of three main contributions, which address the identified subgoals. These contributions are supported by publications in peer-reviewed conferences and journals.

The *first contribution* is the development of a novel graph-based skill discovery method entitled `OGAHC`, which achieves subgoal S1: `OGAHC` is incremental and can identify skills at any time. Thus, it allows an agent to acquire a collection of skills which increases in both size and sophistication over time. `OGAHC` discovers reusable and versatile skills based on identifying bottlenecks of the domain. It is robust with regard to stochasticity of the domain and the explorative behavior of the agent. Moreover, artificial agents using `OGAHC` for skill discovery can considerably outperform other approaches for learning solutions for externally imposed tasks in terms of sample-efficiency. This contribution has been presented at the 10th European Workshop on Reinforcement Learning (EWRL) in Edinburgh and has been published in a special issue of the Journal of Machine Learning Research (JMLR):

- J. H. Metzen (2012b). "Online Skill Discovery using Graph-based Clustering." In: *Journal of Machine Learning Research* W&CP 24. Ed. by M. P. Deisenroth, C. Szepesvári, J. Peters, pp. 77–88

Chapter 4 provides an extended and revised version of this work.

The *second contribution* addresses the identified subgoal S2: graph-based approaches for skill discovery are extended to continuous domains by a novel method entitled `FIGE`. `FIGE` is based on formalizing the problem as a probabilistic generative model and using a maximum likelihood-based approach for generating graphs that capture the dynamics of a continuous domains. Using `FIGE`, graph-based skill discovery and representation learning approaches developed for discrete domains can be extended to continuous domains. The resulting methods are superior compared to prior work and scale to high-dimensional problems. Furthermore, `FIGE`-based skill discovery outperforms a baseline which learns flat task solutions without acquiring skills. This contribution has been presented at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) in Prague and has been published in the corresponding proceedings:

- J. H. Metzen (2013). "Learning Graph-based Representations for Continuous Reinforcement Learning Domains." In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*. Ed. by H. Blockeel, K. Kersting, S. Nijssen, F. Zelezny. Springer Berlin Heidelberg, pp. 81–96

The copyright of this paper is held by Springer Verlag; the paper is available under `http://link.springer.com/chapter/10.1007%2F978-3-642-40988-2_6#`. Chapter 5 provides an extended and revised version of this work.

The *third contribution* corresponding to subgoal S3 consists of the development of an intrinsic motivation mechanism that allows incremental skill acquisition in the absence of external reward signals, e.g., in developmental or lifelong learning settings. The developed intrinsic motivation mechanism governs the behavior of an agent during skill acquisition such that the right amount of time is devoted to both skill discovery and skill learning. By integrating this mechanism with `OGAHC` and `FIGE`, an incremental skill acquisition approach is obtained that achieves the goal stated in Section 1.2. The resulting approach outperforms learning a flat policy in two continuous, high-dimensional control problems. This contribution has been presented at the International Workshop on Intrinsic Motivations and Open-Ended Development in Animals, Humans, and Robots (IMOD-2013) and has been published in the Frontiers of Neurorobotics journal:

- J. H. Metzen and F. Kirchner (2013). "Incremental Learning of Skill Collections based on Intrinsic Motivation." In: *Frontiers in Neurorobotics* 7.11, pp. 1–12

Chapter 6 provides an extended and revised version of this work.

Additionally, the author has published several works in the area of skill learning based on evolutionary policy search (Metzen et al., 2008a; Metzen et al., 2008b), the combination of evolutionary approaches and model learning (Metzen and Kirchner, 2010; Metzen, 2012a), and generalizing and adapting learned skills to new tasks (Metzen and Fabisch, 2013; Metzen et al., 2014):

- J. H. Metzen, M. Edgington, Y. Kassahun, F. Kirchner (2008a). "Analysis of an evolutionary reinforcement learning method in a multiagent domain." In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Richland, SC, pp. 291–298
- J. H. Metzen, M. Edgington, Y. Kassahun, F. Kirchner (2008b). "Evolving Neural Networks for Online Reinforcement Learning." In: *Proceedings of the 10th Conference on Parallel Problem Solving from Nature (PPSN X)*, pp. 518–527
- J. H. Metzen and F. Kirchner (2010). "Model-based direct policy search." In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Toronto, Canada, pp. 1589–1590
- J. H. Metzen (2012a). "Model-based Evolutionary Policy Search for Skill Learning in Continuous Domains." In: *10th European Workshop on Reinforcement Learning (EWRL)*. Edinburgh, Scotland
- J. H. Metzen and A. Fabisch (2013). "Learning Skill Templates for Parameterized Tasks." In: *11th European Workshop on Reinforcement Learning (EWRL)*. Dagstuhl, Germany
- J. H. Metzen, A. Fabisch, L. Senger, J. de Gea Fernandez, E. A. Kirchner (2014). "Towards Learning of Generic Skills for Robotic Manipulation." In: *German Journal of Artificial Intelligence (Künstliche Intelligenz)* Special Issue "Transfer Learning". Accepted

2 Background

"I am always doing that which I cannot do, in order that I may learn how to do it."

Pablo Picasso

2.1 Introduction

This chapter introduces the problem class addressed in this thesis, namely Markov Decision Processes. Moreover, it defines what is considered as optimal behavior, and provides the required background in reinforcement learning, graph theory, and graph clustering. A discussion of the basic concepts of reinforcement learning is provided since it is used throughout the thesis as the basic means for learning of behavior and skills. The background in graph theory is important for this work since graphs are used as primary means for representing structure and the presented graph clustering approaches form the basis for some of the proposed methods that aim at exploiting this structure.

2.2 Decision Making and Optimality

This section introduces Sequential Decision Problems and Markov Decision Processes, and defines formally which behavior is considered as optimal.

2.2.1 Sequential Decision Problems

Sequential Decision Problems (SDPs) are a way to formalize problems in which an autonomous *agent* aims at choosing a sequence of actions such that an external *reward* is maximized.[1] At each time step t, the agent perceives an observation o_t of its *environment* which forms the basis for his choice of the action a_t. The agent chooses its actions a_t

[1] In this work, reward is considered to be a real-valued scalar. This abstraction is used in most related works and is typically sufficient despite the fact that biological models of reward systems are usually more complex.

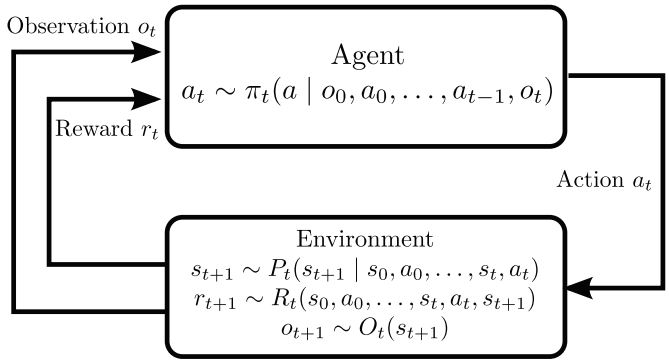


Figure 2.1 – *Agent-environment interface in an SDP*: at each time step t, the agent receives an observation o_t of the environment's current state s_t and a reward signal r_t. It chooses an action a_t based on the entire history of observations $o_0, \ldots, o_t$ and its prior actions $a_0, \ldots, a_{t-1}$ according to its current policy π_t. This action causes the environment to change its state in the next time step to s_{t+1} and yields reward r_{t+1}. Both s_{t+1} and r_{t+1} may depend on the entire history of states and actions. Note that sensors and actuators of embodied agents are typically considered as parts of the environment; because of this, the mapping $\mathcal{O}$ of states onto observations is modeled as a part of the environment.

according to its current behavior *policy* π_t, which maps observations to actions. The agent's actions cause a change of the environment's state s_t to s_{t+1}. The observations the agent perceives from the environment depend on this state of the environment. Furthermore, the environment generates at any time step a scalar reward signal r_t for the agent. In the most general case, the state transitions P_t, the generation of rewards R_t, the generation of observations O_t, and the agent's policy π_t can be stochastic, may depend on the entire history of state, actions, and observations, and can change over time (thus the subscript t). This process is depicted in Figure 2.1.

In the specific case in which the successor state s_{t+1} and the reward r_{t+1} are completely determined by the history of state and actions until time t, i.e., are a function of $s_0, a_0, \ldots, s_t, a_t$, the environment is called *deterministic*. Otherwise, the environment is called *stochastic*. Likewise, the agent is called deterministic if its policy is a function of the history of observations and actions, and stochastic otherwise. In general, sequential decision problems are open-ended, i.e., the agent might act in its environment indefinitely. However, many SDPs have a natural termination situation; e.g., when a certain state of the environment is reached or when a certain number of time steps has elapsed. These problems are called *episodic* and the time step in which the SDP terminates is denoted by T. Problems which are not episodic are denoted as *continuing*. In general, one is interested in agents that *learn*, i.e., agents that can improve their behavior policy π_t over time such that more external reward is received.

Many challenging problems from the field of robotics and artificial intelligence can be framed as SDPs, for instance: path planning, movement generation, object manipulation, control, board and computer games, trading, and operations research (see Section 2.3 for some examples).

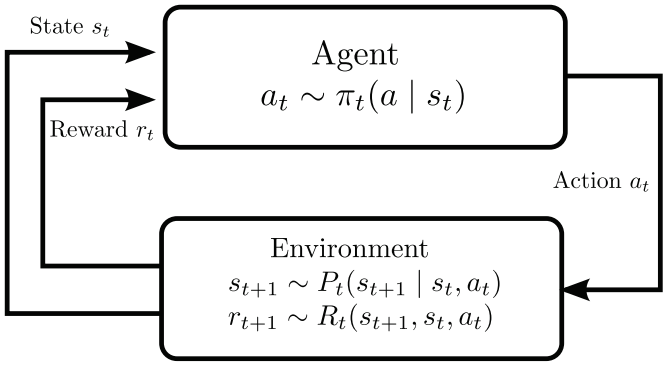


Figure 2.2 – *Agent-environment interface in an MDP according to Sutton and Barto (1998)*: in each time step t, the agent receives the environment's current state s_t and a reward signal r_t and chooses an action a_t based on s_t according to its current policy π_t. This action causes the environment to change its state in the next time step to s_{t+1} and yields reward r_{t+1}.

2.2.2 Markov Decision Processes

An important subclass of SDPs are *Markov Decision Processes* (MDPs) (Puterman, 1994). MDPs are based on three assumptions: (a) *Full Observability*: The agent's observations o_t at any time reveal a full description of the environment's state s_t such that at any point in time the agent can determine the current state of the environment. Thus, there is not necessity to distinguish between observations and states and one can simplify notation by assuming that the agent directly observe the environment's state, i.e., $o_t = s_t$. (b) *Markov property*: An action's outcome depends solely on the current state but not on the history of states and actions, i.e. $P_t(s_{t+1}|s_0, a_0, \dots s_t, a_t) = P_t(s_{t+1}|s_t, a_t)$ and $R_t(s_0, a_0, \dots s_t, a_t, s_{t+1}) = R_t(s_t, a_t, s_{t+1})$. (c) *Time invariance*: The state transition probabilities P_t and the expected immediate rewards R_t are constant over time, i.e., $P_t = P_{t'}$ and $R_t = R_{t'}$ for all t, t'. One can thus drop the subscript t and simply speak of the state transition probability P and the expected immediate reward function R. Common shorthand notations for $P(s_{t+1} = s'|s_t = s, a_t = a)$ and $R(s_{t+1} = s', s_t = s, a_t = a)$ are $P^a_{ss'}$ and $R^a_{ss'}$.

In summary, one can specify an MDP $\mathcal{M}$ as a 6-tuple[1] $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P^a_{ss'}, R^a_{ss'}, S_0, S_T)$, where $\mathcal{S}$ is the set of states of the task (the state space), and $\mathcal{A}$ is the set of actions available for the agent (the action space). The state transition probability distribution $P^a_{ss'}$ determines how likely a transition from state $s \in \mathcal{S}$ to successor state $s' \in \mathcal{S}$ is when executing action $a \in \mathcal{A}$. The immediate reward expectation $R^a_{ss'}$ specifies the expected immediate reward $r \in \mathbb{R}$ when transitioning from state s to successor state s' under executing action a. The start state probability distribution $S_0 : \mathcal{S} \to [0, 1]$ with $\int_{\mathcal{S}} S_0(s)\mathrm{d}s = 1$ determines how likely an episode starts in a particular state s and the terminal state probability distribution $S_T : \mathcal{S} \to [0, 1]$ specifies how likely an episode

[1] Note that the specification of an MDP is not standardized: many authors omit S_0 and S_T since S_0 has no effect onto the optimal policy and value functions and S_T is specific for episodic environments. Moreover, the discount factor γ (see Section 2.2.3) is sometimes included in the MDP's specification.

terminates[1] when the agent reaches a state s. Note that in deterministic environments $P^a_{ss'}$, $R^a_{ss'}$, S_0, and/or S_T might also be considered as functions and written as, e.g., $P(s,a) = s'$ if executing action a in state s always leads to the successor state s' (and analogical for $R^a_{ss'}$, S_0, and S_T). An MDP is called *continuous* if either $\mathcal{S}$ or $\mathcal{A}$ are continuous, otherwise it is called *discrete* or finite. In continuous MDPs, state and action space are often real-valued vector spaces, i.e., $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ and $\mathcal{A} \subseteq \mathbb{R}^{n_a}$. . In this case, n_s is called the *dimensionality* of the state space and n_a the dimensionality of the action space.

MDPs can be seen as an extension of Markov chains where the addition of actions allows the agent to choose and the addition of rewards provides motivation to the agent. Accordingly, if there would only be one action available in each state and all rewards were zero, an MDP would reduce to a Markov chain.

2.2.3 Optimal Behavior

The behavior of an agent is specified by its internal behavior policy π. This policy $\pi(s,a) = \pi(a_t = a|s_t = s)$ specifies the probability of executing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. If an agent uses a deterministic policy, i.e., chooses always the same action in the same state, its policy can also be considered as a function mapping states to actions and written as $\pi(s) = a$. In general, one is interested in an *optimal* policy for a given MDP. This requires specifying which behavior is considered to be optimal, for instance: is choosing actions greedily such that the immediate reward is maximized optimal?

Different models of optimal behavior have been proposed (Kaelbling et al., 1996): in the *finite-horizon* model, the agent should choose actions such that the expected reward $\mathbb{E}_\pi[\sum_{k=0}^{h-1} r_{t+k+1}]$ for the next h steps is maximized under its policy π. This model has the disadvantage that the agent might ignore the long-term consequences of its behavior, i.e., events beyond the horizon h are not taken into account, which is typically not desirable. This issue is addressed by the *infinite-horizon discounted* model, in which the agent should act such that the expected future reward $\mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}]$ is maximized. For $\gamma > 0$, there is no horizon and all future reward is taken into account and thus, the agent has to consider all long-term consequences of its behavior. The parameter $\gamma \in [0,1)$ is a so-called *discount factor* which ensures that the expected future reward is bounded and controls whether the agent focuses more on the short-term ($\gamma \ll 1$) or the long-term rewards ($\gamma \approx 1$). A third model is the *average-reward* model, in which the agent should act such that the expected long-term average reward $\lim_{h\to\infty} \mathbb{E}_\pi[h^{-1} \sum_{k=0}^{h-1} r_{t+k+1}]$ is maximized. This work focuses on the infinite-horizon discounted model since it is the most widely used model.

Related to the models of optimal behavior is the so-called *temporal credit assignment* problem (Minsky, 1961): if the agent obtains a specific reward r_t at time t, to which extent are the actions it has taken so far eligible for this reward? Different models address this issue differently: the finite-horizon model considers all actions $a_{t-h}, \ldots, a_{t-1}$ eligible equally and all prior actions not eligible at all, while the infinite-

[1] Note that this notation also supports continuing environments by setting $S_T(s) = 0 \; \forall s$.

horizon discounted model considers actions taken longer ago less eligible than recent actions, i.e., action a_{t-k} is considered to be eligible to extent γ^{k-1}.

The aggregate of the future rewards defined by a model of optimal behavior is denoted as *return*, e.g., for the infinite-horizon discounted model the return is defined as $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. Thus, optimal behavior can be seen as the behavior which maximizes the expected future return. One can formally define the *optimal policies* for state s at time t as

$$\pi_s^* = \arg\max_{\pi} \mathbb{E}_\pi[R_t | s_t = s],$$

i.e., as the policies which maximizes the expected future return starting from state s. Due to the property that MDPs are memory-less, there are policies π which are not only optimal for some start states s but for arbitrary start states; one can thus denote these policies as π^*. Note that there may be more than one optimal policy.

2.2.4 Value Function and Bellman Equations

One alternative to searching for an optimal policy in the space of all policies directly is to learn an *(action) value function*. Value functions estimate how good it is for an agent to be in a state. Analogously, an action value function estimates how good it is for an agent to execute a specific action in a state. Formally, the value of a state (a state-action pair) is the expected future return starting from this state (state-action pair) under the respective model of optimal behavior when following a particular policy π. Formally, the state value function for the infinite-horizon discounted model is defined as

$$V^\pi(s) = \mathbb{E}_\pi[R_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] \tag{2.1}$$

and the action value function as

$$Q^\pi(s,a) = \mathbb{E}_\pi[R_t | s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right]. \tag{2.2}$$

The value functions are specific for a particular policy since future rewards depend on future actions which are assumed to be chosen according to the policy.[1] One can show that all optimal policies share the same (action) value function and that this action value function is $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. For all value functions, recursive equations exist which are due to Bellman (1957). For the case of the optimal action value function, this *Bellman optimality equation* is

$$Q^*(s,a) = \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma \max_{a'} Q^*(s',a') \right]. \tag{2.3}$$

[1] Note that this explicit dependence on the future actions is omitted in Equations 2.1 and 2.2 and is indicated solely by the subscript π of the expectation.

For any action value function Q, the difference of the left-hand side and the right-hand side of this equation, i.e., $\sum_{s,a} p(s,a)\left\{Q(s,a) - \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma \max_{a'} Q(s',a')\right]\right\}$ with $p(s,a)$ being the probability of a state action pair, is denoted as Q's *Bellman error*.

The optimal policy π^* can be derived from Q^* directly via

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} Q^*(s,a') \\ 0 & \text{else} \end{cases}. \tag{2.4}$$

If $P^a_{ss'}$ and $R^a_{ss'}$ are known to the agent, the agent can compute Q^* (and thus π^*) using Dynamic Programming (Bellman, 1957). Determining π^* this way requires computing Q^*, which consists of $|\mathcal{S}| \cdot |\mathcal{A}|$ values. If $\mathcal{S}$ or $\mathcal{A}$ are vector spaces, the size of $\mathcal{S}$ or $\mathcal{A}$ grows exponentially with the number of dimensions; thus, learning the optimal action value function becomes exponentially harder. This is denoted as the "curse of dimensionality". Furthermore, usually not all components of the MDP are known to the agent; in such a situation where $P^a_{ss'}$ or $R^a_{ss'}$ are unknown, an agent can resort to one of the reinforcement learning algorithms that are discussed in the next section.

2.3 Reinforcement Learning

Reinforcement Learning (RL) is based on ideas from neurobiology and behavioral science and early works in RL were motivated by animal behavior and its neural basis (Minsky, 1954; Klopf, 1972; Sutton and Barto, 1981). Moreover, RL is closely connected to and inspired by the behavioristic concept of operant conditioning (Thorndike, 1911; Skinner, 1938). Thorndike (1911) states the "Law of Effect" for operant conditioning in animals as follows:

> "Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond."

This section focuses on the technical aspects of computational RL; for some interesting links to neurobiology please refer to Niv et al. (2005), who show that temporal difference errors in predictions of future reward (see below) may be represented by phasic activities of dopaminergic neurons in primates' midbrains.

Computational RL focuses on methods for learning an optimal policy π^*. More specifically, RL is about approximating optimal solutions to stochastic control problems (typically modeled as MDP), usually without complete knowledge of the system being

controlled. Furthermore, RL focuses on on-line, incremental learning algorithms rather than batch-style algorithms. Because of this, RL fits well to the kind of learning seen in animals and is well-suited for engineering control problems (Lewis and Vrabie, 2009).

Computational RL has been applied successfully to a multitude of challenging problems. This section gives some examples, see `http://umichrl.pbworks.com/w/page/7597597/Successes%20of%20Reinforcement%20Learning` for more. One prominent area for applying RL is for learning optimal strategies for games. For instance, Tesauro (1995) combined temporal difference learning with an artificial neural network for learning the board game "backgammon". The resulting system, denoted as TD-Gammon, achieved a level of play which was very close to that of top human backgammon players. Moreover, TD-Gammon required only minimal human knowledge and was thus not affected by human prejudice. This allowed TD-Gammon to explore successful strategies that had been ruled out by human players erroneously. By this, TD-Gammon allowed advancing the theory of correct backgammon play. RL has also been applied to other board games with varying success, e.g., othello (van Eck and van Wezel, 2008), english draughts (Faußer and Schwenker, 2010), and chess (Thrun, 1995).

One further prominent area for RL applications is robotic control (Kober et al., 2012). RL has been used to learn locomotion behaviors for legged robots: Kirchner (1998) used a hierarchical RL approach for learning a gait on the six-legged robot "Sir Arthur", while Kohl and Stone (2004) used policy gradient RL on the four-legged AIBO robot and outperformed hand-tuned gaits in terms of forward walking speed. Ng et al. (2004) applied the RL method PEGASUS for learning a controller for sustained inverted flight on an autonomous helicopter. Riedmiller et al. (2009) used RL for learning behaviors in the context of robot soccer. Mülling et al. (2013) used RL for learning to select and generalize striking movements in robot table tennis.

This section gives a brief overview over a subset of methods and concepts that have been developed in (computational) RL. For more details, please refer to Sutton and Barto (1998), Kaelbling et al. (1996), Bertsekas and Tsitsiklis (1996), and Heidrich-Meisner et al. (2007).

2.3.1 Temporal Difference Learning

Temporal Difference (TD) learning (Sutton, 1988) methods are probably the most popular *model-free* RL approaches. Model-free refers to the fact that TD learning neither requires the specification of a model of the environment (in contrast to Dynamic Programming methods which are based on $P^a_{ss'}$ and $R^a_{ss'}$) nor does it learn an explicit model of the environment. Instead, TD learns a policy indirectly by first approximating Q^* and then deriving π^* from Q^* based on Equation 2.4. Since TD is model-free, the only way for the agent to obtain information about the respective MDP is to interact with its environment and to observe the successor state s' and reward r when applying an action a in state s. TD learning provides means for stochastically approximating Q^* based on a set of quadruples $(s_t, a_t, r_{t+1}, s_{t+1})$. Different TD learning methods differ in the specific learning rules; the two most popular methods are Q-learning (Watkins,

Algorithm 2.1 Q-Learning (Watkins, 1989).

1: **Input**: Initial $Q(s,a)$, learning rate $\alpha \in (0,1]$, discount factor $\gamma \in [0,1]$
2: **while** True **do**
3: $s \sim S_o(s)$ # Sample start state for the episode
4: **repeat**
5: $a \sim \pi(a|s)$ # Sample action from policy, e.g., a policy ε-greedy in Q
6: $s' \sim P(s'|s,a)$ # Stochastic state transition according to $P^a_{ss'}$
7: $r \sim R(r|s,a,s')$ # Sample reward for state transition according to $R^a_{ss'}$
8: $Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ # Q-learning update rule
9: $s = s'$ # Continue with successor state
10: **until** $S_t(s)$ # Stop episode when state is terminal
11: $Q(s,a) = 0 \; \forall a$ # Terminal states have value 0 for all actions
12: **end while**

1989) with the learning rule

$$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha_t \left(r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t) \right) \tag{2.5}$$

and SARSA (Rummery and Niranjan, 1994) with the learning rule

$$Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t) + \alpha_t \left(r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t) \right). \tag{2.6}$$

Both algorithms are iterative and on-line in that they update Q_t in every time step based on the current observation, the update requires constant time independent of the number of observations seen, and memory consumption is bounded.The parameter α_t is a learning rate that typically decreases over time and controls how strongly the current observation affects the action value function. Both algorithms *bootstrap*, i.e., they compute their new estimate Q_{t+1} of action values based on their current estimate Q_t. For finite MDPs, the Q-learning update rule can be seen as a stochastic gradient descent on the approximate Bellman error, with $-(r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t))$ being the Bellman error's approximate stochastic gradient (Heidrich-Meisner et al., 2007). Pseudo-code for Q-learning is given in Algorithm 2.1; SARSA is obtained by replacing line 8 with the respective learning rule.

Model-free RL algorithms such as Q-Learning and SARSA are faced with the *exploration-exploitation dilemma*: on the one hand, for convergence to the optimal policy, they are required to try every state-action pair indefinitely often, i.e., they have to *explore* their environment. On the other hand, the ultimate reason for learning is to be able to act in a way such that the obtained reward is maximized, i.e., to choose actions with maximal $Q^*(s,a)$. At any point in time, the agent's best guess for this is to choose the action with maximal $Q_t(s,a)$, which is called *exploitation*. Thus, the agent is faced with a trade-off between two different objectives, namely exploration and exploitation.

One common way to deal with this is ε-greedy action selection:

$$\pi_t(s,a) = \begin{cases} 1-\varepsilon_t+\varepsilon_t/|A| & \text{if } a = \arg\max_{a'} Q_t(s,a') \\ \varepsilon_t/|A| & \text{else} \end{cases}. \tag{2.7}$$

This stochastic policy is implemented easily by executing the greedy action $a = \arg\max_{a'} Q_t(s,a')$ with probability $1-\varepsilon_t$ (exploitation) and choosing an action uniform randomly with probability ε_t (exploration). The main difference between Q-learning and SARSA is that the former is *off-policy* and the latter is *on-policy*. Learning being off-policy means that the agent can follow any policy, even one which chooses actions uniform randomly, but Q_t always approximates Q^*, the action value function of the optimal policy. In contrast, for on-policy learning methods like SARSA, Q_t will converge to Q^π where π is the behavior policy which is used for action selection during learning. Thus, SARSA will not learn the optimal action value function if the behavior policy does not converge to the optimal but unknown policy as $t \to \infty$.

For arbitrary initialization of Q_0, Q-Learning converges asymptotically to the optimal action value function Q^* for any finite MDP if all state-action pairs from $\mathcal{S} \times \mathcal{A}$ are executed infinitely often and the learning rate α_t converges to 0 with $\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty$ and $\sum_{i=1}^{\infty} \alpha^2_{n^i(s,a)} < \infty$ for all s, a where $n^i(s,a)$ is the index of the i-th time action a is executed in state s (Watkins and Dayan, 1992). Similar convergence proofs for SARSA exist (Singh et al., 2000), which require additionally that the behavior policy becomes greedy in the limit with infinite exploration; for instance, for ε-greedy action selection with $\varepsilon_t = 1/t$, convergence to Q^* is guaranteed.

There exist also non-gradient based, second-order TD learning methods like least-squares temporal difference (LSTD) learning (Bradtke et al., 1996; Boyan, 2002). LSTD does not require specifying a learning rate α and is stable for a broad range of conditions where function approximation (see below) is required. Unfortunately, LSTD can only learn state value function for fixed policies. Lagoudakis and Parr (2003) have proposed an extension of LSTD called least-squares policy iteration (LSPI), which allows learning action value functions for control problems, i.e., problems where the policy is improved during learning. LSPI is an off-policy algorithm like Q-learning; however, it reuses samples and has typically a lower *sample-complexity*, i.e., requires less observations of the environment to learn an optimal policy. On the other hand, the cost for each update is quadratic in the number of features as is memory consumption. Accordingly, LSPI is often used in an off-line fashion, i.e., the action value function and corresponding policy are not updated after every time step but less frequently.

2.3.2 Function Approximation

In domains with continuous state space, typically $\mathcal{S} \subset \mathbb{R}^{n_s}$ with n_s being the dimensionality of the state space, the action value function can no longer be represented

exactly since there are infinitely many states. Thus, some kind of approximate representation is required. Often, $Q(s,a)$ is represented by a function Q_θ that is parametrized by $\theta \in \mathbb{R}^d$. The goal of learning is then to find a value $\theta^* \in \mathbb{R}^d$ such that Q_{θ^*} becomes maximal, which implies that the corresponding greedy policy maximizes the long-term reward, while at the same time approximately obeying the Bellman equation. Note that employing function approximation typically implies that the optimal action value function Q^* and the optimal policy π^* are no longer representable. Thus, only a *close-to-optimal policy* $\tilde{\pi}^*$ can be learned which is optimal in the class of representable policies $\Pi = \{\pi_\theta \mid \theta \in \mathbb{R}^d\}$, i.e., $\tilde{\pi}^* = \arg\max_{\pi \in \Pi} \mathbb{E}_\pi[R_t]$. On the other hand, function approximators allow generalizing over the state space such that experience collected in one state allows the agent additionally to learn something about similar states.

A simple class of function approximators are *linear* function approximators. Linear function approximators are based on a set of d features $\phi_0(s,a), \ldots, \phi_{d-1}(s,a)$ and define the action value function to be $Q_\theta(s,a) = \sum_{i=0}^{d-1} \theta_i \phi_i(s,a)$. One way for defining the features are *tilings*: a tiling is an exhaustive partition of the state space $\mathcal{S}$. Each element of the partition is called a tile and is the receptive field for one binary feature $\phi_i(s,a)$. Usually grid-like tilings are used which make computing the indices of the active features straightforward. For grid-like tilings, the computation of the action value function becomes particularly easy: $Q_\theta(s)$ is simply the weight θ_i of the feature ϕ_i that is activated by s,a.

A drawback of tilings is that the value function has to be constant within one cell of the grid and varies discontinuously at the border of such a cell. This drawback could in principle be alleviated by increasing the resolution of the grid, i.e., by increasing the number of features; however this would decrease generalization, increase the number of parameters d to be learned, and thus increase the time until a suitable policy is learned. "Cerebellar Model Articulation Controller" (CMAC) (Albus, 1975) present a partial solution to this problem: instead of using one tiling, CMACs are composed of a set of superimposed tilings. For the case of grid-like tilings, these tilings are often offset randomly. This increases the resolution of the function approximator without decreasing its generalization. A disadvantage of CMACs is that the number of superimposed tilings has to grow in principal exponentially with the dimensionality if one wants to keep a similar resolution level. Thus, CMACs are usually very good function approximators for low dimensional problems but scale not easily to high dimensions.

One way of learning the optimal parameters θ^* for the function approximator is to use (stochastic) *gradient descent*. For this, the gradient $\nabla_{\theta_t} Q_t(s_t,a_t)$ and a desired target output v_t, which can be for instance the one-step SARSA estimate of the return $v_t = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1})$, at time t are required. Based on this, the update rule for θ becomes $\theta_{t+1} = \theta_t + \alpha[v_t - Q_t(s_t,a_t)]\nabla_{\theta_t} Q_t(s_t,a_t)$. For linear function approximation, the gradient is simply $\nabla_{\theta_t} Q_t(s_t,a_t) = \phi(s_t,a_t)$. Thus, in this case the update rule can be interpreted as computing the error $[v_t - Q_t(s_t,a_t)]$ and changing weight θ_i proportional to this error and the activation of the respective feature $\phi_i(s_t,a_t)$.

A general drawback of gradient descent-based learning is that it converges to local optima which are not necessarily global optima if the objective function is not convex.

More severely, gradient descent-based learning combined with off-policy learning methods such as Q-learning may diverge when combined with function approximation under certain circumstances (Baird, 1995). On-policy learning methods like SARSA, on the other hand, converge provably under the similar conditions as required in discrete domains. For more details regarding function approximation, please refer to Sutton and Barto (1998, Chapter 8).

2.3.3 Direct Policy Search

While function approximation allows applying temporal difference-based learning in domains with (low-dimensional) continuous state space, high-dimensional state spaces and continuous action spaces remain challenging. The former is due to the "curse of dimensionality" and the latter, inter alia, due to the requirement of finding an action a that maximizes $Q(s,a)$, which is non-trivial for continuous action spaces. Because of this, so-called *direct policy search* (DPS) methods have become popular recently in domains like robotics (Kober and Peters, 2010; Deisenroth et al., 2013).

DPS does not learn value functions but searches directly in the space of policies. For this, a parametric policy representation is employed, i.e., a policy π_θ parametrized by a vector $\theta \in \mathbb{R}^d$. DPS typically does not bootstrap but searches directly for a policy vector θ such that the expected return of π_θ becomes maximal. For this, two problems need to be addressed: how can the expected return of a policy π_θ be determined and how is the policy space searched efficiently? Typically, the expected return of a policy is approximated by sampling returns for the specific policy by executing it for one or several episodes. The expected return is then estimated based on these samples.

Different approaches for searching the space of policies have been proposed, ranging from evolutionary computation (Whiteson, 2012) over reward-weighted regression (Peters and Schaal, 2007; Kober and Peters, 2010) to information-theoretic policy search (Peters et al., 2010). An empirical investigation revealing some of the advantages and disadvantages of direct policy search and temporal difference learning is given by Kalyanakrishnan and Stone (2011).

2.3.4 Hierarchical Reinforcement Learning

Even though convergence to optimality has been proven for several RL methods under various conditions in the limit, the number of samples, i.e., interactions with the environment, that are required for learning a close-to-optimal policy is often prohibitively large because of the "curse of dimensionality". A common approach to remedy this large sample-complexity is to introduce different kinds of abstraction and hierarchical structure into RL algorithms. The resulting methods form the field of *Hierarchical Reinforcement Learning* (HRL), see Barto and Mahadevan (2003) for a good overview. HRL is also supported by findings from neurobiology and has been proposed as basis for psychological models as well as a framework for investigating the computational and neural underpinnings of hierarchically structured behavior (Botvinick et al., 2009).

By defining abstractions one generally introduces bias into the learning process, which can reduce sample-complexity drastically. However, this comes at the price of potentially not being able to represent and thus learn the optimal policy anymore because of the employed abstractions. The choice of the right kind of abstraction is thus crucial in order to reduce sample-complexity and at the same time allow representing and learning close-to-optimal policies. For a general discussion of advantages and disadvantages of employing an HRL approach, please refer to Chapter 3.

The most popular class of abstraction in HRL are *temporal abstractions*. When using temporal abstractions, the agent need not choose an action at every time step but might invoke temporally extended activities which take over control for several time steps. Such an activity might be for instance "open door" or "dock to charger" in a robotic application. Different terms have been proposed for temporal extended activities; in this work, the term *option* (Sutton et al., 1999b) is used when discussing conceptual details of activities and the term *skill* (Thrun and Schwartz, 1995) when speaking more abstractly about them. A policy which builds upon temporal abstractions is also called a *hierarchical policy*. By providing skills to the agent, the learning problem might become easier for the agent since it only has to learn the right sequence of skills instead of both how to execute the individual skills and how to sequence them.

A second kind of abstractions are *spatial abstractions*. Spatial abstractions are often combined with temporal abstractions and make learning a skill more simple by allowing, e.g., ignoring certain dimensions of the state space which are not relevant for the skill. When using spatial abstractions, the so-called *hierarchical credit assignment* problem needs to be addressed (Dietterich, 2000b): this problem refers to a situation where an agent obtains a reward and must decide if a certain skill which is currently executed is eligible for this reward. Since lower-level skills employ spatial abstractions, they cannot always decide if they are eligible or if the responsibility lies on higher levels of the architecture. On the other hand, higher levels of the architecture, which have the required information, should not be bothered with all details of lower levels. This would become impossible if they would have to decide for every reward who is eligible for it.

HRL requires also specifying which hierarchical policy is considered to be the optimal one. The intuitive notion would be to consider a hierarchical policy to be optimal if it accrues the same reward as the optimal flat policy π^*. However, because of the abstractions employed in an HRL architecture, such a policy might not be representable in a hierarchical fashion. Thus, this notion of optimality, which is denoted as *hierarchically optimal*, is not useful in situations where abstractions are employed. Moreover, hierarchically optimal policies often complicate transfer and reuse of learned skills since the best skill policy on a lower level depends typically on requirements of the higher level, task-dependent policy. A different concept of optimality is *recursive optimality*, which requires solely that all policies on all layers of the architecture are optimal in isolation. Note that this does not imply that the resulting hierarchical policy is hierarchically optimal; recursively optimal hierarchical policies are not even unique and there are some recursive optimal policies which accrue more reward than others.

Nevertheless, recursive optimality is a useful concept in practice. Please refer to Dietterich (2000b) for more details on the different optimality concepts.

Related to this is whether the temporal abstractions are used to actually *abstract* the action space, i.e., the agent can only choose among the skills, or to *augment* the action space, i.e., the agent can decide whether to invoke a skill or to learn on the lower layer of primitive actions directly. When augmenting the action space, the agent can always learn hierarchically optimal policies since it could eventually resort to learning on the finest time scale; however, abstracting the action space is typically more useful for reducing sample-complexity and for learning skills that are reusable in different tasks. See Jong et al. (2008) for a discussion.

A further issue in HRL is the choice of the architecture: one can distinguish between 3-layer[1] hierarchies, where skills (the middle layer) can only invoke primitive actions, and more complex action hierarchies where skills can also invoke other skills. This work focuses on 3-layer hierarchies. Learning in 3-layer hierarchies can be subdivided into three parts: *compositional learning*, which deals with learning the agent's overall policy and involves skill utilization, *skill learning*, which aims at learning policies for achieving the individual skills, and *skill discovery*, which deals with identifying reoccurring subproblems of problem classes, whose solutions—the skills—are reusable and simplify learning of solutions for future problems. Note that skill discovery is qualitatively different in that it alters the temporal abstraction hierarchy itself by adding new skills, while skill learning and compositional learning work on a fixed temporal abstraction hierarchy. Not all works in HRL have addressed all three subfields jointly; for instance, skill discovery (and sometimes skill learning) are omitted and subproblems (and potentially solution policies for these problems) have been manually defined beforehand. However, for a fully autonomous agent it is clearly desirable that it can identify and learn new skills autonomously.

The remainder of this section presents popular frameworks proposed for HRL, introduces a formalization of HRL based on Semi-Markov Decision Processes, and gives an overview over corresponding learning methods. An overview over the works in the field of skill discovery is given in Section 3.3.

2.3.4.1 Frameworks

The probably most popular approach to HRL is the *options framework* (Sutton et al., 1999b), which formalizes skills as *options*. An option $o =< I_o, \pi_o, \beta_o >$ consists of the following three components: the option's initiation set $I_o \subset \mathcal{S}$ determining the states in which the option may be invoked, the option's termination condition $\beta_o : \mathcal{S} \rightarrow [0,1]$ specifying the probability of option execution terminating in a given state, and the option's policy π_o which defines the probability of executing an action in a state under option o. In the options framework, the agent's policy π may in any state s decide not solely to execute a primitive action but also to call any of the options $o \in O$ for

[1] The lowest layer consists of the primitive actions, the middle layer of the skills, and the highest layer of the agent's overall policy.

which $s \in I_o$. If an option is invoked, the option's policy π_o is followed until the option terminates according to β_o. Option policies might invoke other options (resulting in hierarchies with more than 3-layers); it must be guaranteed though that in the end, the option call sequence terminates by selecting a primitive action. If the option's policy π_o chooses actions based solely on the current state, it is called a *Markov option*. Options whose policies depend not solely on the current state but on the entire history of states, actions, and rewards since the option was initiated are called semi-Markov. Note that any policy that may invoke other options, such as the top-level policy π in a 3-layer setting, is semi-Markov.

The option's policy π_o is defined relative to an option-specific "pseudo" reward function R_o that may differ from the global external reward function (Dietterich, 2000b). Skill learning consists of learning π_o given a so-called skill prototype $\Psi_o = (I_o, \beta_o, R_o)$, while compositional learning refers to learning π given primitive actions and options. Skill discovery, on the other hand, consists of choosing an appropriate skill prototype Ψ_o for a new option o.

Besides the options framework, further notable approaches to HRL are Max-Q value function decomposition (Dietterich, 2000b), Hierarchies of Abstract Machines (Parr and Russell, 1997), Feudal RL (Dayan and Hinton, 1993), Compositional Q-Learning (Singh, 1992b), and HQL (Kirchner, 1996). These frameworks are not discussed in detail in this work, for an overview please refer to Barto and Mahadevan (2003).

2.3.4.2 Semi-Markov Decision Processes

The formal MDP theory is not directly applicable to HRL since it assumes that all actions take one time step; unfortunately, extended activities like options employed in HRL may take several time steps. In contrast, discrete time Semi-Markov Decision Processes (SMDPs) (Howard, 1971; Puterman, 1994), which are an extension of MDPs, can model temporally extended activities. SMDPs are like MDPs with the exception that state transitions happen not immediately but after a random waiting time τ, where τ is a positive integer for the discrete-time case. The state transition probability P is a joint probability over the successor state s' and the waiting time τ given the current state and actions a, i.e., $P(s', \tau | s, a)$. The expected reward R for executing action a in state s gives the amount of discounted reward expected to accumulate over the waiting time τ. The notion of optimal value functions generalizes to SMDPs as well, e.g.,

$$Q^*(s,a) = \sum_{s',\tau} P(s', \tau | s, a) \left(R(s,a,s',\tau) + \gamma^\tau \max_{a'} Q^*(s', a') \right), \tag{2.8}$$

as do most Dynamic Programming algorithms. When using SMDPs to model HRL problems, the environment itself is still an MDP and skill learning based on primitive actions can still be performed by standard RL algorithms for MDPs. Compositional learning, however, requires SMDP techniques since the options may take several time steps. The resulting SMDP can be thus seen as induced by the underlying MDP and the temporal abstraction hierarchy.

Sutton et al. (1999b) introduced multi-time models of an option o, which generalize $P^a_{ss'}$ and $R^a_{ss'}$ from the underlying MDP to the induced SMDP. This allows treating compositional learning similarly to standard RL. The multi-time models are defined as

$$R^o_{ss'} = \mathbb{E}_O \left[r_{t+1} + \gamma r_{t+2} + \ldots \gamma^{\tau-1} r_{t+\tau} | E(o,s,t) \right], \tag{2.9}$$

where $E(o,s,t)$ is the event that option o is invoked in state s at time t and $t+\tau$ is the random time at which option o terminates, and

$$P^o_{ss'} = \sum_{\tau=1}^{\infty} \gamma^\tau p(s', t+\tau | E(o,s,t)), \tag{2.10}$$

where $p(s', t+\tau | E(o,s,t))$ is the probability that option o terminates in state s' at time $t+\tau$ when invoked in state s at time t. Based on this multi-time models, the Bellman equation for the optimal option value function Q^*_O can be written as:

$$Q^*_O(s,o) = \sum_{s'} P^o_{ss'} \left[R^o_{ss'} + \max_{o'} Q^*_O(s', o') \right]. \tag{2.11}$$

For a given Q^*_O, the optimal policy can be defined as the policy which picks the option $o = \arg\max_{o'} Q^*_O(s,o')$. Note that primitive actions can mixed in this formulation with options by considering them as one-step options.

2.3.4.3 SMDP Learning

Based on the Bellman equation for the optimal option value function Q^*_O, Q-learning can be extended to SMDPs (and thus to compositional learning) using the update rule

$$Q_{t+\tau}(s_t, o_t) \leftarrow Q_t(s_t, o_t) + \alpha_t \left(r + \gamma^\tau \max_{o'} Q_t(s_{t+\tau}, o') - Q_t(s_t, o_t) \right). \tag{2.12}$$

This update is performed when option o_t is invoked in state s_t and terminates after τ time steps in state $s_{t+\tau}$ and obtained reward $r = \sum_{k=0}^{\tau-1} \gamma^k r_{t+k+1}$ during execution. This form of Q-learning in SMDPs is known as SMDP Q-learning (Bradtke and Duff, 1994) and Macro Q (McGovern et al., 1997). Macro Q reduces to conventional Q-learning if all options terminate immediately. On-policy learning using SARSA can be extended to SMDPs accordingly.

A drawback of SMDP Macro Q-learning and SARSA are that they treat a temporally extended activity like an option as an opaque, indivisible unit (Precup, 2000) and update their action value function only when the option terminates, i.e., they learn only on the SMDP level and not on the underlying MDP level. Furthermore, SMDP Q-learning updates only the currently active option even though Q-learning itself is off-policy and could thus in principle be used to update policies of several options at once based on a single experience.

These drawbacks are addressed by intra-option learning methods like, e.g., one-step intra-option Q-learning (Precup, 2000). If the agent experiences a transition $(s_t, a_t, r_{t+1}, s_{t+1})$ on the MDP level while following policy $\overline{\pi}$, then the following update is applied for every Markov option $o =< I_o, \pi_o, \beta_o >$ with $\pi_0(s_t, a_t) = \overline{\pi}(s_t, a_t)$:

$$Q_{t+1}(s_t, o) \leftarrow Q_t(s_t, o) + \alpha_t \left(r_{t+1} + \gamma U_t(s_t, o) - Q_t(s_t, o)\right) \tag{2.13}$$

where $U_t(s,o) = (1-\beta_o(s))Q_t(s,o) + \beta_o(s)\max_{o'} Q_t(s,o')$. The quantity U_t is an estimate of the value of the state-option pair (s,o) which takes into account that the optimal option o' in s can only be invoked if o terminates. Precup (2000) has shown that for deterministic Markov options, intra-option Q-learning converges to the optimal option policies under similar preconditions as those required for Q-learning.

Off-policy learning of several option policies based on a single transition is clearly very advantageous; unfortunately, it can only be used in settings where off-policy learning algorithms exist which converge. Thus, the utility of intra-option Q-learning is limited to domains which do not require function approximation since it may (and actually does) diverge in this setting (see Section 2.3.2). There has been recent progress in developing off-policy learning algorithms that are stable when combined with function approximation, like, e.g., Greedy-GQ (Maei et al., 2010). The main requirement of Greedy-GQ is that the behavior policy needs to be stationary. Potential future methods which remove this requirement would be of great value for learning in HRL.

2.4 GRAPH THEORY

This section gives a brief introduction into graph theory and approaches for clustering graphs into partitions of their vertices. This section follows mainly the notation used by von Luxburg (2007), who also gives more details about the topics summarized in this section.

2.4.1 Basics

We define a *graph* as $G = (V,E)$ with vertex set $V = \{v_1, \ldots v_n\}$ and a set of edges $E \subset V \times V$. If for all $v_i, v_j \in V$, $(v_i, v_j) \in E \Leftrightarrow (v_j, v_i) \in E$, we call the graph undirected, otherwise we call it directed. We denote an edge connecting v_i and v_j as e_{ij}. The cardinality of a graph is denoted by $|G|$ and defined as $|G| = |V|$. Given a subset of vertices $A \subset V$, we denote its complement $V \setminus A$ by $\overline{A}$. A shorthand notation commonly used is $i \in A$, which denotes the set of indices $\{i \mid v_i \in A\}$.

We define a *path* connecting two vertices $v_a, v_b \in V$ in G as a sequence of edges $p_G(v_a, v_b) = (e_{ai_1}, e_{i_1 i_2}, \ldots, e_{i_{n-1} b})$ with $e_{ai_1}, e_{i_1 i_2}, \ldots e_{i_{n-1} b} \in E$. We say that this path has length $\text{len}(p_G) = n$. Among all paths connecting v_a and v_b, the (not necessarily unique) path $p^*_G(v_a, v_b)$ with minimal length is called the *shortest path*. The *geodesic distance* of v_a, v_b in G is defined as $d_G(v_a, v_b) = \text{len}(p^*_G(v_a, v_b))$.

We define a *subgraph* $G' \subset G$ as a graph $G' = (V', E')$ whose vertex set is a subset of G's vertex set, i.e., $V' \subset V$, and whose edge set is $E' = E \cap (V' \times V')$. We call $G' \subset G$ connected if any two vertices $v_i, v_j \in V'$ are connected by a path in G'. We call G' a *connected component* of G if it is connected and has no edges to other nodes in V, i.e., $(V' \times (V \setminus V')) \cap E = \emptyset$. We call G a *tree*, if it consists of a single connected component and removing any of its edges would split it into two connected components. We call G a *forest*, if all of its connected components are trees. A *multigraph* is a graph which may have more than one edge connecting two vertices $v_i, v_j \in V$, i.e., $E \not\subset V \times V$.

Moreover, we define a weighted graph as $G = (V, E, w)$, where $w : E \to \mathbb{R}$ is a function assigning to each edge a real-valued weight. The weight assigned to the edge e_{ij} is denoted by w_{ij}. For an undirected graph, we have $w_{ij} = w_{ji}$ and for two unconnected nodes $v_i, v_j \in V$, we set $w_{ij} = 0$. The weighted adjacency matrix of G is the matrix $W = \left(w_{ij}\right)_{i,j=1,\ldots,n}$. The degree of a vertex v_i is defined as $d_i = \sum_{j=1}^{n} w_{ij}$. Accordingly, we define the degree matrix as a diagonal matrix with the degrees on the diagonal, i.e., $D = \mathrm{diag}(d_1, \ldots, d_n)$. For two subgraphs $A, B \subset G$, which need not necessarily be disjoint, we define $W(A,B) = \sum_{i \in A, j \in B} w_{ij}$. Furthermore, we define the volume of a (sub-)graph $G' = (V', E')$ as $\mathrm{vol}(G') = \sum_{i \in V'} d_i$.

The sets $A_1, \ldots, A_k$ are called a *partition* of V if $A_i \neq \emptyset \; \forall i$, $A_i \cap A_j = \emptyset \; \forall i \neq j$, and $\bigcup_{i=1}^{k} A_i = V$. We call such a partition $\mathcal{P} = \{A_1, \ldots, A_k\}$ also a clustering of G and the individual A_i a *cluster*. Moreover, $[v]_{\mathcal{P}} \in \mathcal{P}$ denotes the cluster to which node v has been assigned, i.e., $v \in [v]_{\mathcal{P}}$. Given two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ of V, we say that $\mathcal{P}_1$ refines $\mathcal{P}_2$ if and only if for all pairs of graph nodes $(v_i, v_j) \in V \times V$, $[v_i]_{\mathcal{P}_1} = [v_j]_{\mathcal{P}_1}$ implies $[v_i]_{\mathcal{P}_2} = [v_j]_{\mathcal{P}_2}$, i.e., all graph nodes being in the same cluster in $\mathcal{P}_1$ are necessarily also in the same cluster in $\mathcal{P}_2$. We denote this with $\mathcal{P}_1 \leq \mathcal{P}_2$, where the relation $\leq$ defines a partial ordering on the set of all partitions of V, compare Jonsson and Barto (2006).

2.4.2 Graph Cuts

Graph cuts are based on defining an objective function on graph partitions, i.e., they assign to every partition $\mathcal{P} = \{A_1, \ldots, A_k\}$ of graph nodes V a scalar value. The partition which minimizes this quantity is the optimal graph cut according to this criterion. These objective functions are often based on the notion of a cut, which is defined as $\mathrm{cut}(\mathcal{P}) = \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A_i})$. The most direct way to define an objective function for a partition is the MinCut objective: $\mathcal{P}^* = \min_{\mathcal{P}} \mathrm{cut}(\mathcal{P}) = \min_{\mathcal{P}} \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A_i})$. The MinCut objective favors partitions in which each cluster is well separated from the rest of the graph, i.e., where the weights of the edges which connect a cluster A_i with the rest of the graph $\overline{A_i}$ are small. Unfortunately, the MinCut objective often results in partitions in which most of the A_i consist of one or only a few nodes. This is often not desirable; typically, one is interested in finding a partition in which each cluster is "reasonably

large". Two commonly used extensions of the MinCut objective that take the clusters' size into account are RatioCut (Hagen and Kahng, 1992) and the normalized cut NCut (Shi and Malik, 2000) :

$$\text{RatioCut}(\mathcal{P}) = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A_i})}{|A_i|} \qquad \text{NCut}(\mathcal{P}) = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \overline{A_i})}{\text{vol}(A_i)}, \tag{2.14}$$

where $\text{cut}(A_i, \overline{A_i})$ is shorthand notation for $\text{cut}(\{A_i, \overline{A_i}\})$. The two objective criteria differ in how they measure the size of a cluster: while RatioCut measures the size of a cluster by the number of nodes in this cluster, NCut uses the cluster's volume as size. Both criteria aim at choosing clusters of similar size—with the definition of size being the main difference—and at the same time separating each cluster as well as possible from the rest. Unfortunately, balancing both objectives makes minimizing RatioCut and NCut $\mathcal{NP}$-hard, see Wagner and Wagner (1993) for a discussion. Spectral Clustering (see Section 2.4.3.1) provides means to find approximate solutions for these problems in polynomial time.

There exists an interesting relationship between the NCut criterion and random walks on graphs as pointed out by Meila and Shi (2001). For a random walk on a graph, the transition probability of jumping from vertex v_i to v_j in one step is defined as $p_{ij} = w_{ij}/d_i$. The corresponding transition matrix of the random walk is thus $P = D^{-1}W$. The following proposition according to von Luxburg (2007) holds:

Proposition 1 NCut via transition probabilities: *Let G be connected and non bipartite.*[1] *Assume that one runs the random walk $(X_t)_{t\in\mathbb{N}}$ starting with X_0 in the stationary distribution $\Pi = d_i/vol(V)$. For disjoint subsets $A, B \subset V$, denote by $P(B|A) := P(X_{t+1} \in B | X_t \in A)$. Then:*

$$NCut(A, \overline{A}) = P(\overline{A}|A) + P(A|\overline{A})$$

See von Luxburg (2007) for a proof. Thus, finding a partition of a graph with minimal NCut implies that a random walk on this graph has minimal probability of jumping between vertices that belong to different clusters. This relationship between random walks and NCuts is important for this work since finding a close-to-optimal NCut of the sample transition graph (see Section 4.2.1) effectively partitions the MDP into subproblems such that a randomly exploring agent would only very unlikely transition from one of these subproblems into another one (see Section 3.1.1).

2.4.3 Graph Clustering

Since finding a graph partition with minimal NCut is $\mathcal{NP}$-hard, several heuristics have been proposed that determine partitions with close-to-minimal NCut. This subsection

[1] A graph $G = (V, E)$ is called bipartite if its vertices V can be split into two disjoint subsets A, B such that no edge in E connects two vertices that are both in A or both in B.

Algorithm 2.2 Normalized Spectral Clustering (Shi and Malik, 2000).

Input: Graph $G = (V, E, w)$ with $|V| = n$, number of clusters k
$\mathcal{L}_{rw} = I_n - D^{-1}W$ # Random-walk graph Laplacian for G
Matrix $U \in \mathcal{R}^{n \times k}$ containing first k eigenvectors $u_1, \ldots, u_k$ of $\mathcal{L}_{rw}$ as columns
$U = \text{EIGENDECOMPOSITION}(\mathcal{L}_{rw}, k)$
$y_1, \ldots y_n = \text{ROWS}(U)$ # Denote the n rows of U by $y_1, \ldots y_n \in \mathcal{R}^k$
$C_1, \ldots, C_k = \text{KMEANS}((y_i)_{i=1,\ldots,n}, k)$ # Cluster the y_i into k clusters $C_1, \ldots, C_k$
return Partition $\mathcal{P} = \{A_1, \ldots, A_k\}$ with $A_i = \{v_j | y_j \in C_i\}$

presents three such heuristics: spectral clustering, PCCA$^+$, and agglomerative hierarchical clustering. An empirical comparison of these heuristics is given in Section 4.3.1.

2.4.3.1 Spectral Clustering

Spectral clustering (see von Luxburg (2007) for an overview) denotes a set of approaches for creating a partition of a set of datapoints based on a similarity graph using methods from spectral graph theory (Chung, 1996). The similarity graph contains the datapoints as vertices and uses a pairwise similarity measure on these datapoints for defining the edge weights. Spectral clustering aims at identifying a partition of the similarity graph such that points in different clusters are dissimilar from each other while points in the same cluster are similar to each other. This is typically formalized using RatioCut or the normalized cut NCut; spectral clustering can thus be seen as a computational efficient heuristic for finding close-to-optimal solutions for these $\mathcal{NP}$-hard problems. In the context of this work, spectral clustering is used in a slightly different context: instead of clustering a similarity graph, spectral clustering is used to determine a partition of a sample transition graph, which encodes the structure of an MDP.

Spectral clustering is based on graph Laplacian matrices. Different variants of these matrices exist:

1. The unnormalized graph Laplacian matrix $\mathcal{L} = D - W$.
2. The symmetrized graph Laplacian matrix $\mathcal{L}_{sym} = D^{-\frac{1}{2}}\mathcal{L}D^{-\frac{1}{2}} = I_n - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$.
3. The random-walk graph Laplacian matrix $\mathcal{L}_{rw} = D^{-1}\mathcal{L} = I_n - D^{-1}W = I_n - P$.

In the context of this work, $\mathcal{L}_{sym}$ is used as basis for identifying proto-value functions in Section 5.5. $\mathcal{L}_{rw}$ can be used as basis for a spectral clustering algorithm to approximate the partition with minimal NCut (see Algorithm 2.2). This algorithm can be motivated from perturbation theory: in the ideal case, the between-cluster similarities, i.e., the similarity of datapoints that belong to different clusters, are zero. In this case, the eigenvectors u_k of $\mathcal{L}_{rw}$ are the indicator vectors of these clusters (see proof of Proposition 2 in (von Luxburg, 2007)) and the $y_i \in \mathcal{R}^k$ will be 0 in all but one dimension, in which they will have value 1. This dimension indicates the connected component to which v_i belongs. Thus, y_i will be identical for all i where v_i belong to the same connected component and k-means clustering will assign them to the same cluster which yields the "correct" partition for this ideal case. In a more realistic case, one might still have distinct clusters in G but the between-cluster similarities will not

Algorithm 2.3 PCCA$^+$: Robust Perron Cluster Cluster Analysis (Deuflhard and Weber, 2005).

1: **Input:** Graph $G = (V, E, w)$ with $|V| = n$, number of clusters k
2: $T = D^{-1}W$ # Stochastic random-walk matrix for G
3: # Matrix $U \in \mathcal{R}^{n \times k}$ containing the k eigenvectors $u_1, \ldots, u_k$ of T as columns
4: $U = \text{EIGENDECOMPOSITION}(T, k)$
5: $y_1, \ldots y_n = \text{ROWS}(U)$ # Denote the n rows of U by $y_1, \ldots y_n \in \mathcal{R}^k$
6: $\pi = \{\arg\max_i ||y_i||_2\}$ # Select k rows, starting with row with maximal 2-norm
7: **while** $|\pi| < k$ **do**
8: # Add row with maximal distance to hyperplane spanned by selected rows π
9: $\pi = \pi \cup \{\arg\max_{i \notin \pi} \left|\left|y_i - P_{\text{span}\{y_\pi\}}(y_i)\right|\right|_2\}$ # $P_{\text{span}\{y_\pi\}}(y_i)$: projection of y_i on hyperplane
10: **end while**
11: $\chi = U \times U_\pi^{-1}$ # Multiply U with inverse of its $k \times k$ submatrix with rows π selected
12: **return** Partition $\mathcal{P} = \{A_1, \ldots, A_k\}$ where $A_i = \{v_j \mid \arg\max_l \chi[j,l] = i\}$

be exactly zero but only close to zero. Accordingly, the Laplacian matrices will be "perturbed" versions of the ones in the ideal case whose eigenvectors will be close to the ideal indicator vectors. The y_i will also be slightly perturbed versions of the ideal ones, i.e., most entries will be close to 0 and one entry indicating the cluster should be close to 1. If the perturbations are not too strong, k-means clustering should be able to recover the optimal partition also from the perturbed y_i. See von Luxburg (2007) for a summary of a more formal treatment of perturbation theory.

A loose upper bound on the computational worst-case complexity of normalized spectral clustering is $\mathcal{O}(n^3 + kn)$, assuming that the eigendecomposition in line 4 requires $\mathcal{O}(n^3)$ and k-means clustering in line 6 requires $\mathcal{O}(kn)$, which is the case when the number of iterations is constant in k and n.

2.4.3.2 Robust Perron Cluster Cluster Analysis

A further spectral clustering algorithm derived from perturbation theory is "Robust Perron Cluster Cluster Analysis" (PCCA$^+$) by Deuflhard and Weber (2005). PCCA$^+$ is similar to Algorithm 2.2, with the main differences being that (a) the spectral clustering is not based on the random-walk graph Laplacian $\mathcal{L}_{rw}$ but based on a stochastic random walk matrix $T = D^{-1}W$ and (b) that the rows are not clustered using k-means but by projecting the rows onto a k-dimensional hyperplane and assigning the nodes to the closest indicator vector (the closest simplex corner). PCCA+ is shown in Algorithm 2.3. A loose upper bound on the computational worst-case complexity of PCCA$^+$ is $\mathcal{O}(n^3 + kn)$, assuming that the eigendecomposition in line 4 requires $\mathcal{O}(n^3)$ and each iteration of the loop in line 7-9 is in $\mathcal{O}(n)$.

2.4.3.3 Agglomerative Hierarchical Clustering

An alternative to spectral clustering for finding a partition with close to minimal NCut is agglomerative hierarchical clustering (Hastie et al., 2008, Section 14.3.12), see Algorithm 2.4. This algorithm starts by assigning all vertices into a separate cluster

Algorithm 2.4 Agglomerative Hierarchical Clustering

1: **Input:** Graph $G = (V, E, w)$, number of clusters k
2: $\mathcal{P} = \{\{v\} \mid v \in V\}$ # Initially: one cluster per vertex
3: $V_\mathcal{D} = \{\{v\} \mid v \in V\}$, $E_\mathcal{D} = \emptyset$ # Optional: Initial nodes and edges of dendrogram
4: **while** $|\mathcal{P}| > k$ **do**
5: $M_c = \{(p_1, p_2) \mid (p_1 \times p_2) \cap E \neq \emptyset\}$ # Merge-candidates: clusters connected in G
6: $p_1^*, p_2^* = \underset{p_1, p_2 \in M_c}{\arg\max}\, \mathrm{NCut}(p_1, p_2)$ # Identify merge candidate with maximal NCut
7: $\mathcal{P} = (\mathcal{P} \setminus \{p_1^*, p_2^*\}) \cup \{p_1^* \cup p_2^*\}$ # Merge p_1^* and p_2^*
8: # Optional:
9: $V_\mathcal{D} = V_\mathcal{D} \cup \{p_1^* \cup p_2^*\}$ # Add new cluster as node to dendrogram
10: $E_\mathcal{D} = E_\mathcal{D} \cup \{(\{p_1^* \cup p_2^*\}, p_1^*), (\{p_1^* \cup p_2^*\}, p_2^*)\}$ # Connect new node to its two source nodes
11: **end while**
12: **return** Partition $\mathcal{P}$, Dendrogram $\mathcal{D} = \{V_\mathcal{D}, E_\mathcal{D}\}$

and then iteratively merges greedily the pair of connected clusters with maximal NCut,[1] i.e., the two clusters which have strong between-cluster similarity relative to the within-cluster similarities. The algorithm stops once the partition consists of k clusters; the resulting clustering should have small NCut since clusters with large NCut have been merged early. However, due to the greedy nature of the algorithm, there is no guarantee for finding the partition with minimal NCut.

This algorithm exploits situations in which the graph is only sparsely connected: the size of the set of merge candidates identified in row 5 depends on how many of the clusters are connected by an edge. If the graph is only sparsely connected, typically only few clusters are connected in G, and thus the set of merge candidates is small. This reduces the runtime of the algorithm since the arg max in row 6 goes over a smaller set. Furthermore, agglomerative clustering can optionally also create a so-called dendrogram $\mathcal{D}$, from which the entire history of cluster merges can be recovered. The dendrogram is a tree for $k = 1$ and a forest for $k > 1$.

In the general case, the computational worst-case complexity of this algorithm is in $\mathcal{O}(n^3)$. By using a more efficient version than is shown in Algorithm 2.4 where, among other things, the list of merge-candidates M_c is implemented as a priority queue with the NCut of the merge used as priority, this complexity can be reduced to $\mathcal{O}(n^2 \log n)$. For specific linkage criteria like the single-linkage, the worst-case complexity can even be reduced to be in $\mathcal{O}(n^2)$. These linkages, however, will not be suited for the problems discussed in this thesis. Please refer to Murphy (2012, Chapter 25.5.1) for more details.

[1] Note that this notion generalizes NCut from entire partitions to sub-partitions consisting of two clusters, i.e., $\mathrm{NCut}(A, B) = \frac{\mathrm{cut}(A,B)}{\mathrm{vol}(A)} + \frac{\mathrm{cut}(B,A)}{\mathrm{vol}(B)}$.

3
Skill Discovery

"If I have seen further it is by standing on the shoulders of giants"

Sir Isaac Newton, letter to Robert Hooke, 1676

3.1 INTRODUCTION

THIS chapter introduces the problem of *skill discovery*. As discussed in Section 1.1, the term *skill* denotes a piece of procedural knowledge which allows achieving a specific result in a close-to-optimal way regarding criteria like reliability or speed. In the context of computational HRL, skills are typically considered to be temporal abstractions. Temporal abstractions have proven to allow speeding-up learning and planning and building prior knowledge into artificial intelligence system, see, e.g., (Fikes et al., 1972; Sacerdoti, 1974; Korf, 1985; Iba, 1989) and have also been explored in the context of MDPs (Singh, 1992a; Parr and Russell, 1997; Dietterich, 2000b; Precup, 2000), see also Section 2.3.4.

This work discusses temporal abstractions in the options framework (Sutton et al., 1999b). As discussed in Section 2.3.4 (see there for more details), within the options framework, a skill is formalized as an option $o = (I_o, \beta_o, \pi_o)$ consisting of the option's initiation set $I_o \subset \mathcal{S}$ determining the states in which the option may be invoked, the option's termination condition $\beta_o : \mathcal{S} \to [0,1]$ specifying the probability of option execution terminating in a given state, and the option's policy π_o which defines the probability of executing an action in a state under option o. Moreover, the option's policy π_o is defined relative to an option-specific reward function R_o. Skill learning denotes the process of learning π_o given the skill prototype $\Psi_o = (I_o, \beta_o, R_o)$, while skill discovery consists of choosing an appropriate skill prototype Ψ_o for a new option o.

Skill discovery approaches can be categorized according to the class of heuristic which is used internally for discovering skills. Among these classes are: (a) solution-based heuristics, which determine skills for a domain by introspecting successful (potentially optimal) solutions and identifying common structures or specific properties, (b) factoring-based heuristics, which try to exploit independence properties of state space dimensions, (c) frequency-based heuristics, which compute local statistics of

states for identifying subgoal states such as bottlenecks, and (d) graph-based heuristics, which aim at identifying and exploiting the structure of a domain by representing the structure explicitly as a graph. Heuristics from different classes can nevertheless yield similar skills. For instance frequency-based and graph-based heuristics may lead to similar skills since visit counts of states often coincide with graph-theoretic measures like betweenness and both aim typically at identifying bottlenecks and use these as option subgoals.

Section 3.2 discusses potential benefits and challenges of using a skill-based HRL approach and which properties are desirable for autonomous skill discovery. Section 3.3 presents a review of related works in skill discovery, grouped by the underlying type of heuristic, and discusses to which extent these works exhibit the desirable properties identified before. Thereupon, Section 3.4 discusses how the performance of skill discovery can be evaluated empirically. However, first an illustration of how exploiting domain structure can be helpful for skill discovery in a classical domain is given.

3.1.1 Illustration: Towers of Hanoi

This subsection illustrates the idea of exploiting problem structure for decomposing a problem into more simple subproblems based on the "Towers of Hanoi" (ToH) game. This game consists of three stacks on which N_p pieces are placed. These pieces have different sizes and the movement of pieces is restricted such that only one piece may be moved at a time and no piece may be placed on top of smaller pieces, i.e., pieces on the same stack must always be ordered according to their size. In general, the task of an agent playing the game is to find a sequence of moves of minimal length that transforms a start state into a specific goal state.

Figure 3.1 shows an example of four states of ToH with $N_p = 5$. The figure shows that even though there is a large number of possible states of the game, the shortest sequence of moves from start state s_0 to goal state s_G must traverse through the two middle states since these are the only configurations in which the black piece can be moved from Stack 0 to Stack 2 (this is in fact true for any pair of start and goal states where the black piece is on Stack 0 in the start state and on Stack 2 in the goal state). Thus, if this property of the domain is known to the agent, the general task of finding the shortest path from start state s_0 to goal state s_G can be broken down into two subtasks: (1) finding the shortest path from start state s_0 to s_a and (2) finding the shortest path from s_b to the goal state s_G. However, typically one would like the agent to discover this property of the domain on its own rather than giving this knowledge to it explicitly.

How can an agent identify this domain property autonomously? Figure 3.2 shows the *state transition graph* of the ToH game for $N_p = 5$. In this state transition graph, every graph node corresponds to one state of the game and every edge to a possible move between two states of the game; see Section 4.2.1 for a more formal definition. Note that all edges are bidirectional since any move can be undone. A state transition graph captures thus structure of the domain in the form of a graph.

If the transition graph is given, it can be split into densely connected subgraphs by any of the clustering algorithms discussed in Section 2.4.3. One can show that the

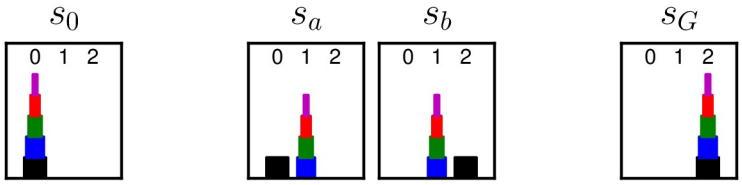


Figure 3.1 – *Four states of "Towers of Hanoi" with 5 Pieces.* Let s_0 be the start state and s_G be the goal state. The shortest sequence of moves from s_0 to s_G must contain the two states s_a and s_b since the black piece must be moved from Stack 0 to Stack 2 and this requires that all other pieces are on Stack 1, which corresponds to s_a and s_b.

optimal clustering into $k = 3$ clusters for the NCut-linkage (see Section 2.4.2) is the one shown in Figure 3.2. The edges connecting two nodes that belong to different clusters correspond to the graph's "bottlenecks".[1] The start state s_0 of the example in Figure 3.1 lies in the "blue" cluster and the goal state s_G in the "red" cluster. The edge connecting the states s_a and s_b corresponds to one of the bottlenecks, namely to the one between the blue and red cluster. Thus, the clustering of the state transition graph allows identifying the desired domain property and thus, decomposing the problem into two subproblems.

In summary, one approach of exploiting problem structure for decomposing a problem into more simple subproblems is to cluster the problem's state transition graph into densely connected subgraphs, consider the connections of these clusters as bottlenecks, and try first to reach an appropriate subgoal—typically corresponding to one of the identified bottlenecks—and only after this the actual goal state. The questions addressed in this thesis are: (a) how can bottlenecks be identified in domains with unknown state transition graphs and (b) how can this concept be extended from simple toy problems like the ToH game to more challenging domains that resemble typical real-world robotic control problems.

3.2 CONSIDERATIONS

This section motivates why a skill-based HRL approach can be useful for an agent, which challenges need to be addressed, and which properties are desirable for a skill discovery method.

3.2.1 Potential Benefits

This section addresses the question "why should an agent discover and learn a collection of skills in the first place instead of solely learning monolithic task-solutions directly?". This question is important since from a computational point of view, discovering and learning skills typically only increases computational demands and memory

[1] The term "bottleneck" will be defined more formally in Section 4.2.2; for now, a bottleneck in a graph can be considered to be an edge which lies on the shortest path between many pairs of graph nodes.

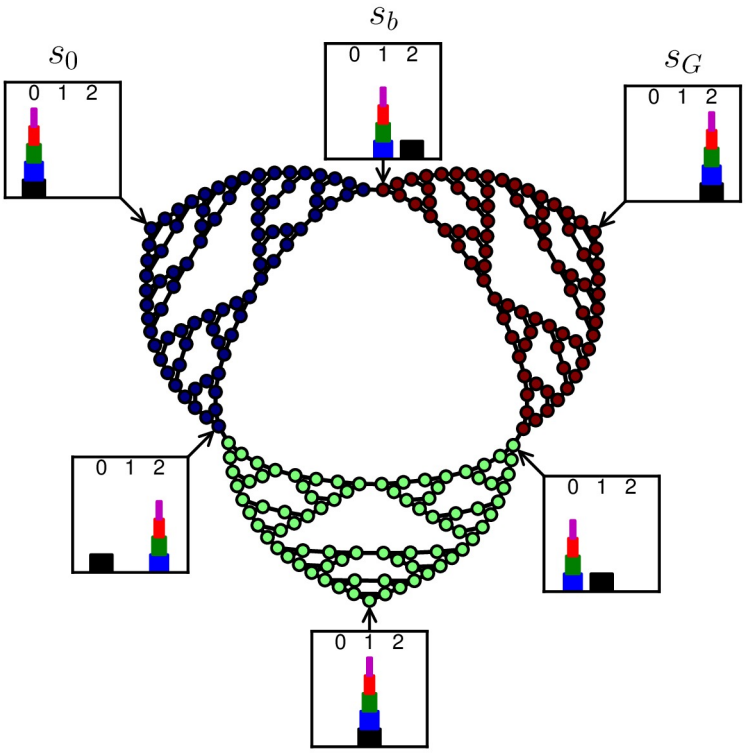


Figure 3.2 – *State transition graph of "Towers of Hanoi" with 5 pieces.* The graph is annotated with typical start, goal, and bottleneck states. The coloring of the graph nodes corresponds to the optimal clustering of the graph into $k = 3$ clusters.

consumption. Thus, this increased resource consumption of temporal abstractions must be compensated by a considerable benefit elsewhere.

The first motivation for skill learning is the following: intuitively, a skill-based approach is appealing since it allows the agent to learn in a way which resembles classical *divide-and-conquer* approaches that have been proven successful for a variety of problems: in the divide phase, a problem is split into a set of subproblems (skill discovery) and solutions for these subproblems are determined (skill learning). Thereupon, in the conquer phase, the solutions of the subproblems are combined to a solution of the overall problem (skill utilization in the compositional learning of the higher level policy). Solving the overall problem is simplified since it requires the agent solely to chunk together sub-solutions which shortens the effective length of the solution. For instance, Sutton et al. (1999a) have shown that the state-action space of a complex mission planning task can be reduced from more than 24 billion elements to less than a million elements by using temporal abstractions. Effectively, choosing among higher-level skills allows the agent to take larger and at the same time more meaningful steps through the search space of behavioral strategies than by simply choosing between primitive actions (Barto et al., 2013).

Related is the concept of *modularity*: many complex systems, be they man-made or natural, are organized hierarchically from a set of modules, which have some degree of mutual independence (Barto et al., 2013). Simon (1996) used the term "nearly decomposable" to denote problems which can be solved by such a modular solution close-to-optimally. Skills may be considered as the modular building blocks of goal-directed behavior.

A further advantage of skills is that they allow the agent to employ *spatial abstractions*, i.e., that learning solutions for subproblems can be simplified in situations where, e.g., some of the dimensions of the state space are irrelevant for certain subproblems and need not be taken into account or several dimensions can be combined into a lower dimensional representation (Dietterich, 2000c; Li et al., 2006; Barto et al., 2013). This makes the subproblems lower-dimensional, which can be important because of the "curse of dimensionality" (see Section 2.2.4). Note that a comparable monolithic policy may not be able to use the same kind of state abstraction since the overall problem may require intrinsically all state space dimensions and only the individual skills' subproblems may be lower-dimensional. However, it should be noted that if solving the subtasks is as difficult as solving the original task, the hierarchical decomposition can actually make learning the optimal policy harder because of the additional complexity added by the hierarchical credit assignment problem.

As discussed by McGovern et al. (1997), a further potential benefit of using skills in RL is that they can offer a beneficial *exploration bias* to the stochastic policy of the agent. For instance, the often used reach-subgoal options make the respective subgoal state more prominent during learning: just picking the corresponding option once moves the agent to the respective subgoal (given that the option is executed successfully). This effectively modifies the random-walk connectivity of the MDP: every state in the option's initiation set gets now connected to the option's subgoal, which is often a bottleneck of the domain. This may help to reduce the "flailing" pattern which is often observed for stochastically exploring agents. An example for this is shown in Figure 3.3: while random exploration with only primitive actions visits every state equally often (left plot), random exploration with primitive actions and options, which lead to the domain's bottlenecks, visit the bottlenecks and states on the path between these bottlenecks considerably more often (middle plot). Further evidence for a positive effect of a skill hierarchy onto exploration is given by Vigorito and Barto (2010) in a structured domain in which random exploration is very unlikely to visit certain states. Here, the skill hierarchy allows exploring the environment more intelligently by taking advantage of the environment's structure and to reach parts of the state space that are not easily accessible via exploration based on primitive actions.

However, there is no guarantee that the explorative bias provided by temporal abstractions is always helpful. Jong et al. (2008) present an empirical study on the utility of temporal abstractions and provide examples where the explorative bias can be even harmful. Figure 3.3 gives an example: if options can be invoked anywhere (middle plot), the probability of visiting the goal state (the right upper graph node) by chance is actually smaller than for the case of primitive actions only (left plot). Only when

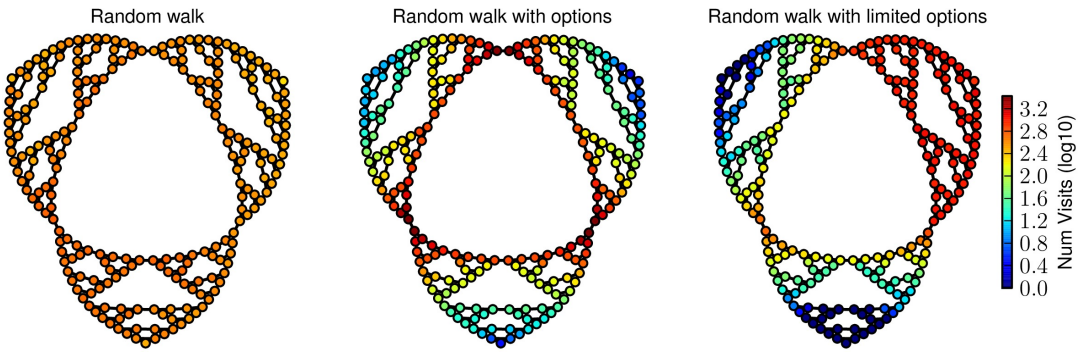


Figure 3.3 – *Visit count under random walks in "Towers of Hanoi"*: the plots show how often states are visited under a random walk of length 10^5. Left plot: random walk with primitive actions only. Middle plot: random walk with primitive actions and options for reaching bottlenecks. Right plot: random walk with primitive actions and options for reaching bottlenecks; options cannot be invoked in the vicinity of the goal state (right, upper graph node). Probability of selecting an option was set to 0.05.

option execution is restricted to states that are not in the vicinity of the goal state, i.e., are separated by a bottleneck from the goal, the probability of visiting the goal state is actually increased (right plot). Thus, using a hierarchical RL approach is not a panacea; it requires some careful architectural choices and thus some domain knowledge to be practical (see Section 3.2.2).

A further potential benefit of temporal abstractions is that they may *accelerate the propagation of action values* (McGovern et al., 1997): since temporal abstractions like options effectively reduce the temporal resolution of the high-level policy, rewards can be propagated back more efficiently. For instance, in standard Q-learning, values are propagated backwards one step per update while in Macro Q, they are propagated back several time steps, more specifically: over τ time steps if the respective option lasted for τ time steps. McGovern et al. (1997) show that this can considerably improve sample-efficiency in a setup where reward is sparse and the value function is initialized pessimistically. Note that the temporal resolution is increased again when learning the option policies themselves. Thus, this benefit applies mainly on the upper layers of an HRL architecture.

A further reason for creating options is that they allow *transfer* in multi-task settings, in which the agent is faced with several different but related tasks (Thrun and Schwartz, 1995; Bernstein, 1999; Perkins and Precup, 1999; Pickett and Barto, 2002; Konidaris and Barto, 2007; Mehta et al., 2008; Konidaris et al., 2012; Hawasly and Ramamoorthy, 2013). Transfer learning (Taylor and Stone, 2009) is considered to be significant for both natural and artificial agents since many real world problems exhibit the multi-task property: for instance, driving in a car each day to work is always slightly different because of changing weather conditions, road conditions, or traffic. This causes a slightly different but nevertheless related task each day. Transfer of knowledge can make an agent "competent" (White, 1959) in such an environment, which means that the agent can solve efficiently a multitude of tasks imposed on him in this environment.

In the framework of RL, multi-task learning often corresponds to a set of MDPs which share the same state-transition probabilities $P^a_{ss'}$ but differ in the rewards $R^a_{ss'}$ which encode the different tasks.[1] In this case, options learned in one task can potentially be reused in any of the other tasks; for instance, an option for reaching a bottleneck like a doorway might be useful for a multitude of tasks. Thus, options allow that basic capabilities are learned only once and reused later on, i.e., they make an agent more competent. This can not be achieved easily with a flat policy. However, this comes at the price that the freedom to select actions is reduced since task policies need to commit to identical actions in the same state when using the same option. Thus, the class of policies which are representable by a temporal abstraction hierarchy is typically smaller than the class of flat policies.

Related to this is the motivation for using options in non-stationary environments, where options may allow *adapting* policies more easily to (small) changes in the environment (Digney, 1998; Stolle and Precup, 2002). In a changing environment, the transition probability may change over time, i.e., $P^a_{ss'}$ may be non-stationary. A classical example is a maze-world with several doorways, where doors may open or close at certain points in time. While such a change may require that a flat policy is completely relearned, which involves forgetting the old solution and learning a new solution more or less from scratch, a hierarchical policy may get by with adapting the high-level policy to use different options without modifying the options' policies themselves.

Some authors (Konidaris and Barto, 2009; Barto et al., 2013) point out that in continuous domains (and in large discrete domains), there is an additional *representational benefit* from using options: since continuous domains have infinitely many states and/or actions, the agent must necessarily resort to an approximate solution of the problem, e.g., by representing the value function by a function approximator (see Section 2.3.2) or by using a parametrized policy representation (see Section 2.3.3). When using such a fixed, flat parametric representation, there is typically a trade-off between representability and learnability of policies: the more policies can be represented by the parametric representation, the more parameters need to be learned, and the more difficult it gets to find the optimal parameter values. By using options, the policy is represented hierarchically instead of a monolithically. This frees the primary, top-level policy from representing complex solution strategies for all parts of the state space jointly by allowing the outsourcing of partial solutions into the options' policies. At the same time, the option policies can focus on a particular subset of the state space and can thus employ function approximators with fewer basis functions. This is useful since it breaks down a many-parameter problem into several problems with less parameters and thus alleviates the curse of dimensionality. Konidaris and Barto (2009) call these skills which can employ simpler function approximation techniques "lightweight options". Moreover, the authors provide empirical evidence that the benefit of hierarchical policy representations can be considerable in continuous domains.

[1] There is also some work on learning options that are reusable in domains with differing $P^a_{ss'}$, most notably the work of Konidaris and Barto (2007), in which options are learned in a non-task-specific agent space but can be used for learning solutions in the task-specific problem space.

3.2.2 Challenges

While the former section has outlined several potential benefits of using a hierarchical RL approach, it also briefly sketched some of the major challenges. This section discusses these challenges in more depth. The first issue is the choice of a *learning architecture* which determines how exactly options are integrated in the learning process. Jong et al. (2008) argue that one has to carefully design at what point in time options are introduced into the learning process, whether the high-level policy can choose solely among options (abstraction) or among options and primitive actions directly (augmentation), where in the state space which option can be invoked, and which exploration mechanism is used. Thus, some experience with the options framework is required for designing a learning architecture which allows the agent to actually profit from a set of reusable skills. Related to the choice of the learning architecture is the hierarchical credit assignment problem (see Section 2.3.4), which needs to be addressed when the architecture employs spatial abstractions.

A second issue is the *trade-off between compactness and representability* of close-to-optimal policies as discussed by Thrun and Schwartz (1995). The authors consider the description length of policies in a multi-task scenario: policies for different tasks which are based on shared skills have a more compact joint-representation, i.e., have a smaller description length, since they share the same skills. Since learning requires the agent to search the space of policies for an optimal one, a smaller policy space (one with smaller description length) typically means that learning is more sample-efficient, i.e., less interaction with the environment is required for finding the optimal policy in the policy space. On the other hand, a hierarchical policy in a space with small description length typically also suffers a performance loss, i.e., the performance of the optimal hierarchical policy in a task can be worse than the performance of the optimal flat policy. This resembles the situation in supervised learning, where model classes with small capacity are good for situations with little training data but higher capacity is desirable once more training data becomes available (von Luxburg and Schölkopf, 2011). However, it is well known that also models with small capacity, i.e., small description length, can be optimal if the model class matches the respective problem well. Similarly, a good choice of skill prototypes may allow both sample-efficiency and (close-to) optimality. This emphasizes the need for good skill discovery approaches, which generate temporal abstractions that reduce the description length considerably without suffering a big performance loss.

A further issue when the agent is required to discover and learn skills autonomously is *learning speed*, i.e., sample efficiency. Several authors have noted that the number of samples required for identifying and learning useful skills can be considerably larger than for learning a close-to-optimal policy (Singh, 1992b; Thrun and Schwartz, 1995). The authors suspect that this is because discovering skills using their approach is much harder than learning control. If this is the case, then why should skills be learned in the first place? One possible use case is a scenario where an agent is faced with a sequence of different but related tasks: while a skill learning agent would be less efficient in the first tasks it could benefit in later tasks by reusing skills acquired earlier. However,

in general it would be desirable to develop methods for skill discovery which allow identifying and learning useful skill before a close-to-optimal task policy has been learned.

A further requirement on the learned skills is that they should be reusable in different but related tasks and thus be *task-independent*. However, skill discovery and skill learning often take place concurrently to learning a task solution. Thus, two challenges arise: how can one learn skills which are reusable, i.e., not specific for a given task, when they are discovered and learned in a setting with externally imposed task? And: how can one trade-off the learning of task-independent skills and task-specific solutions? One approach to the first challenge is to ignore the reward in skill discovery since the reward typically encodes the external task, i.e., to use an option-specific pseudo reward R_o which is not based on the external reward. Related to this is the learning of skills in a *developmental* setting (see Chapter 6), in which no external task is imposed onto the agent but the agent can explore its environment freely. In such a setting, only skill discovery approaches which do not identify skills based on properties of task solutions and value functions are applicable. On the other hand, skill discovery offers great potential in this setting since it allows continual learning by acquiring new skills based on bootstrapping existing structural and procedural knowledge (Barto et al., 2013).

A further issue is the *granularity* of the skills, i.e., how many skills should be learned. Clearly, learning only a single skill will not be very useful in most domains; on the other hand, learning too many skills may cause a kind of overfitting where skills become to specific and can hardly be reused. Furthermore, too many skills may overly increase computational resource usage and may make learning the high-level policies more difficult (Sutton et al., 1999b); this is known as "utility problem" or "skill proliferation". Pickett and Barto (2002) present an empirical comparison of different choices for the number of skills and show that the resulting difference in performance can be considerable. Thus, the specific choice of the number of skills is crucial but hard to choose beforehand. It is thus desirable that skill discovery chooses the granularity automatically during learning based on the domain properties.

A further critical factor is the *timing* of skill discovery, i.e., the point in time in which skills are discovered and added to the hierarchical RL architecture. The timing is critical since performing skill discovery too early may result in suboptimal skill prototypes due to insufficient experience of the agent in the environment. Performing skill discovery too late may be potentially harmful if the agent has already learned a close-to-optimal task policy based on primitive actions and the new skills only lead to a digression from this policy. Moreover, even if the options with optimal policies for reaching actual bottlenecks are provided to the agent, providing the agent too early with these options might interfere with a pessimistic initialization of the action value function (Jong et al., 2008).

Lastly, *continuous domains* pose additional challenges for skill discovery as discussed by Konidaris and Barto (2009). These challenges arise mostly from the fact that initiation sets I_o and termination conditions β_o of the skill prototypes cannot be defined by sets of discrete states but must use some kind of generalization over the continuous

state space. For instance, letting an option terminate successfully solely in a single goal state is not viable since an agent may never visit the same state twice. Thus, the goal must be an entire region of the state space, which raises the question how large the region should be chosen: a too small region may be impossible to reach while a too large region may make the option trivial. Similarly, initiation sets must be regions of the state space with infinitely many states, which must be determined by skill discovery. Moreover, since continuous domains require approximate representations of policies and value functions, there may be a complex interdependency between the granularity of skills and the capacity of policy representations.

3.2.3 Desirable Properties

Based on the challenges identified for skill-based HRL, this section identifies five desirable properties for skill discovery:

PROPERTY P1 Skill discovery should be incremental. This means that skill discovery must not be restricted to a single specific point in time during learning but should rather be performed regularly. This addresses the challenge of timing since choosing an appropriate frequency for skill discovery is less critical than choosing a single point in time. Incremental skill discovery methods also allow discovering skills of increasing sophistication gradually.

PROPERTY P2 Skill discovery should be applicable in a developmental setting and be able to identify skills which are reusable in several different tasks. This requires that skills are identified based on exploring an environment without external reward feedback (or by ignoring the external, task-specific reward feedback). This addresses the task-independence challenge.

PROPERTY P3 Skill discovery should decide automatically how many skills are identified. This means that users need not have a good intuition about how many skills are appropriate for a specific domain but can determine the skill granularity based on more intuitive parameters like specifics of a bottleneck. This addresses the challenge of skill granularity.

PROPERTY P4 Skill discovery should not require that the number of states is finite and small but be able to handle infinitely large state spaces by taking some kind of state similarity into account. This addresses the challenge of domains with continuous state spaces.

PROPERTY P5 Skill discovery should be sample-efficient. While sample-efficiency is not a binary criterion but rather one of degree, it is clearly desirable that skills are discovered as early as possible; typically, before task solutions are learned. This addresses the challenge of learning speed.

Note that subgoal S1 stated in Section 1.2 corresponds to the properties P1 and P3, subgoal S2 corresponds to property P4, and subgoal S3 to property P2. Furthermore, the methods proposed in this work have been developed explicitly with the aim of being sample-efficient, i.e., exhibiting property P5. The challenges of choosing the

learning architecture and the policy representation are not addressed here since they are typically handled outside of skill discovery. The same holds true for the hierarchical credit assignment problem.

3.3 RELATED WORKS

Most prior work on autonomous skill discovery is based on the concept of *bottleneck areas* in the state space. Informally, bottleneck areas have been described as the border states of densely connected areas in the state space (Menache et al., 2002) or as states that allow transitions to a different part of the environment, the so-called *access states* (Şimşek and Barto, 2004). A more formal definition is given by Şimşek and Barto (2009) which define bottleneck areas as those states which are local maxima of betweenness—a measure of centrality on graphs —on the state transition graph. Figure 3.4 gives an illustration of betweenness in the Towers of Hanoi game (see Section 3.1.1): nodes of the transition graph which correspond to states in which the largest piece can be moved from one stack to an other have large betweenness (red color in graphic) while nodes which correspond to states in which the second-largest piece can be moved from one stack to an other have medium betweenness (yellow color in graphic). Other nodes have considerably smaller betweenness. Thus, nodes with high betweenness correspond to states that one would consider as bottlenecks intuitively. Other graph-based centrality measures have been proposed as alternatives to betweenness, such as connection graph stability and connection bridge centrality (Moradi et al., 2010).

Once bottleneck areas have been identified, typically one (or several) skills are defined that try to reach this bottleneck from a local neighborhood of the bottleneck. Since betweenness requires complete knowledge of the transition graph and is computationally relatively expensive,[1] several heuristics have been proposed to identify bottlenecks. The next section gives an overview over related works in skill discovery grouped by the class of heuristics being used and discusses which works exhibit which of the properties identified in Section 3.2.3.

3.3.1 Solution-Based Heuristics

The class of solution-based heuristics is based on analyzing the structure of learned policies or value functions; thus, there is no explicit definition of what constitutes a bottleneck or subgoal state but subgoal states are identified based on properties of task solutions. Typically, the learned policies need to be close-to-optimal. Furthermore, some methods are tailored to multi-task settings and analyze the learned policies for commonalities. This approach was pioneered by Amarel (1968) and Anzai and Simon (1979), who created subgoal states by examining solutions of previous problems.

An early method from this class suited for MDPs is SKILLS (Thrun and Schwartz, 1995). SKILLS explicitly models the trade-off between policy description length and

[1] The computational complexity of the computation of the betweenness on a weighted graph with n nodes and m edges lies in $\mathcal{O}(nm + n^2 \log n)$.

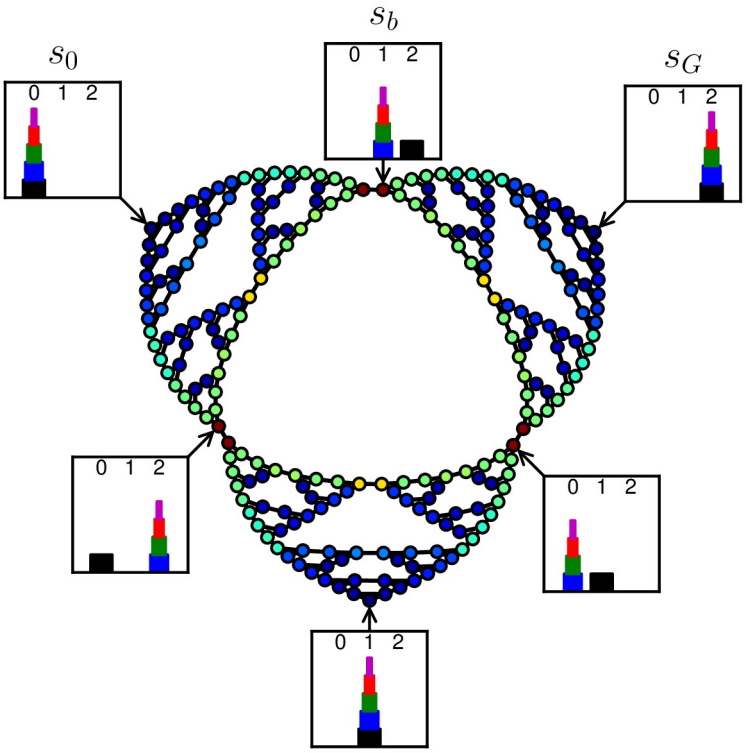


Figure 3.4 – *Betweenness for "Towers of Hanoi" with 5 pieces.* Red nodes of the state transition graph correspond to states with large betweenness, yellow nodes to medium betweenness, and blue nodes to small betweenness.

performance loss (see Section 3.2.2) in a multi-task setting. SKILLS initializes skills randomly and optimizes them using stochastic gradient descent such that description length and performance loss are minimized. The description length is computed for all policies in the multi-task problem jointly; this way, sharing temporal abstractions between tasks can reduce description length. Performance loss is measured based on a comparison to the optimal value functions (hence the requirement for first learning optimal value functions). Empirical evaluations in maze worlds show that useful skills can be learned which reduce the description length considerably without incurring a large performance loss.

A similar approach is PolicyBlocks (Pickett and Barto, 2002), with the main difference being that PolicyBlocks takes redundancy explicitly into account while SKILLS addresses this only implicitly. Moreover, PolicyBlocks uses a greedy bottom-up approach while SKILLS uses stochastic gradient descent. The authors show that PolicyBlocks can outperform SKILLS and even manually designed options in multi-tasks scenarios where the tasks differ considerably.

Kirchner (1995) proposed to identify subgoal states based on the so-called "Subgoal-Utility" (SGU). The SGU is computed as follows: for any state s_t, k random actions are performed and the Q-values of the k visited states are accumulated. This is compared

with the accumulated Q-values of the states that are visited under k steps of the current policy starting from s_t. If this difference (the SGU) is large, this indicates that s_t is a state where the specific choice of actions is quite important and thus, the state can be considered as a kind of access state to a new region of the state space. It has been shown empirically in maze worlds that maxima of SGU are typically states in front of doorways since in these states it is important that the agent takes the appropriate actions to go through the doorways (Bieberstein, 2006). The SGU of any visited state is given as an additional reward to the agent. This way, the agent is rewarded for visiting subgoal states. In contrast to most other discussed approaches, SGU does not create explicit skills but uses the discovered subgoal states for generating a kind of pseudo-reward, which is used to boost learning. Thus, SGU can be considered as method for reward shaping (Gullapalli and Barto, 1992).

A related approach is Q-Surfing (Kirchner and Richter, 2000): Q-surfing computes so-called significance values based on the action value function Q. These significance values are large in a state s if the difference between the value $Q(s,a^*)$ of an optimal action a^* and the mean Q-value in this state is large. The authors show that, as the value function converges, the significance values become large in bottleneck and corner states of the domain. The authors use the significance values for speeding up the propagation of Q-values in the planning phase of a model-based RL architecture; however, as the significance values are indicators of bottlenecks, they could also be used for skill discovery in principle (albeit not before the action value function is about to converge).

A more recently proposed skill discovery method is "skill chaining" (Konidaris and Barto, 2009). Skill chaining has been designed explicitly for domains with continuous state spaces. It produces chains (or more general: trees) of skills such that each skill allows reaching a specific region of the state space, such as a terminal region or a region where an other skill can be invoked. Skill chaining requires specifying an area of interest (typically the terminal region of the state space) which is used as target for the skill at the root of the tree. In the first step, a skill for reaching this goal is learned. Since the domain is continuous and the capacity of the skill policy is bounded, the skill's policy can only learn a local solution.[1] Skill chaining determines the set of states from which the goal can be reached using this local solution, and uses this set as target area for a second skill. This way, skill chaining learns incrementally, backwards from the goal, a chain of skills from start to goal state. Thus, skill chaining can be considered as a solution-based heuristic which identifies skills not only based on properties of the environment but also based on the representational capacity of skill policies. In discrete domains, where optimal policies can be represented exactly, skill chaining would typically only learn a single, global skill. Two disadvantages of skill chaining are that (a) it must be able to reach the goal state before any skill can be discovered and (b) for multi-task domains with several goal regions or in developmental settings without any goal state, it is unclear how the root of the skill tree should be chosen.

[1] Note that this is related to the description length versus performance loss trade-off: by using skill policy representations with larger capacity, the policies could represent more complex and thus less local behavior.

Skill chaining can also be combined with learning-from-demonstrations: in this case, the agent need not discover skills completely autonomous but is provided with a set of example trajectories obtained by human demonstrations, which it splits into simpler segments. These segments correspond to behavioral building blocks and are mapped onto skills (Konidaris et al., 2010).

More recently, Hawasly and Ramamoorthy (2013) proposed "Incremental Learning of Policy Space Structure" (ILPSS). ILPSS is devised for multi-task settings and tries to exploit common structure in learned (not necessarily optimal) policies. It clusters the regions of the state space which are preferred by successful task trajectories and extracts the corresponding policies fragments for the regions. These policy fragments form the basis for reusable options. ILPSS can be used in domains with continuous state space but requires specifying the number of policy fragments. Not requiring optimal policies but only ones which are able to reach the goal makes ILPSS slightly more sample-efficient than similar approaches.

3.3.2 Factoring-Based Heuristics

A second class of heuristics are factoring-based approaches. These approaches generally assume that the state space possesses a specific structure in the sense that there exists conditional independence relations between some state variables (dimensions of the state space) or that some state variables vary more frequently than others since they correspond to information on lower levels.

Hengst (2002) proposed HexQ, a method which attempts to decompose an MDP by dividing and abstracting the state space such that a hierarchy of simpler MDPs is generated. For this, it assumes that some of the state variables (dimensions of the state space) vary more frequently than others. An ordering of the state variables according to this criterion is determined based on statistics obtained during a random exploration. Thereupon, so-called exit states (corresponding to subgoal states) are determined in which an action can cause a change in one of the less frequently changing state variables. Based on this, the MDP is decomposed into several subproblems. One drawback of HexQ is that a simple ordering of state variables may not always reflect the causal relation of state variables adequately.

Variable Influence Structure Analysis (VISA) by Jonsson and Barto (2006) assumes that there exist conditional independences between state variables for given actions and that these are given to the agent in the form of a dynamic Bayesian network (DBN). Based on this DBN, VISA creates a causal graph describing relationships of the state variables and partitions this graph into strongly connected components. Note that this partitioning is a partitioning of state variables and not of states themselves which would be the case in the graph-based heuristics. As in HexQ, exit states for these partitions are determined and options and state abstractions are generated based on this. While VISA is quite sample-efficient, this is mainly because it requires that domain knowledge in the form of a DBN is given by the user; thus, VISA not an autonomous skill discovery approach.

3.3.3 Frequency-Based Heuristics

A further class of heuristics are *frequency-based approaches* that compute local statistics of states, for instance how frequently a state is visited. This heuristic was explored by Korf (1985) and Iba (1989) for discovering macro-operators in deterministic search problems. In the context of discovering skills in MDPs, Digney (1996) proposed to consider states that are visited frequently as subgoal states. Similarly, Asadi and Huber (2005) consider states as subgoals, which lie—under a given policy—on a considerably larger number of paths than their potential successors.

McGovern and Barto (2001) proposed "Diverse Density", which identifies bottlenecks as those states that are visited frequently on successful but infrequently on unsuccessful trajectories, where a trajectory is considered as successful when a goal state is reached. Stolle and Precup (2002) propose a similar approach, with the main difference being that a multi-task setting is assumed. The agent learns good policies for each task (thus, this could also be seen as a solution-based heuristic) and counts how often states are visited under these policies. States visited frequently are identified as subgoal states. In contrast to diverse density, no differentiation between successful and unsuccessful trajectories is made.

"Relative Novelty" (Şimşek and Barto, 2004) considers bottlenecks as those states that allow the agent to transition to an area in the state space that is otherwise difficult to reach from an other region. Relative novelty of a state under a state sequence is defined as the ratio of the novelty of its successor states in this sequence to its predecessors, where the novelty of a state set is inversely proportional to the square root of visits of this state set before. In order to compensate for sampling bias, only states with large relative novelty in a large proportion of state sequences are considered as subgoal states (hence this approach is a frequency-based heuristic).

3.3.4 Graph-Based Heuristics

An other popular class of heuristics are *graph-based approaches*. Graph-based approaches are based on estimates of the state transition graph and aim at partitioning this graph into subgraphs that are densely connected internally but only weakly connected with each other. The extent to which a partitioning of a graph possesses this property is determined by a so-called linkage criterion, often one of the graph cut objectives discussed in Section 2.4.2.

Menache et al. (2002) have proposed Q-Cut, a top-down approach for partitioning the global transition graph based on the max-flow/min-cut heuristic. They use the RatioCut linkage (Hagen and Kahng, 1992) for determining bottlenecks. One disadvantage of their approach is that it requires that two states are chosen as fixed source and sink nodes for the graph cut calculation and that Q-cut can only split a graph into two components. The latter issue can be resolved by Segmented Q-Cut, which uses Q-Cut in a divide-and-conquer manner.

Şimşek et al. (2005) follow a similar approach but partition local estimates of the global transition graph using a spectral clustering algorithm and use repeated sampling

for identifying globally consistent bottlenecks. Due to the repeated sampling the approach shares some properties with the frequency-based heuristics and is not very sample-efficient. The graph partitioning is based on the Normalized Cut linkage (Shi and Malik, 2000), see Section 2.4.2. In a follow-up, Şimşek and Barto (2009) employed a similar approach but used betweenness (see Figure 3.4) instead of graph cuts for bottleneck identification. Local maxima of betweenness of the overall graph are likely to be local maxima of its subgraphs; thus, by identifying these local maxima in a collection of subgraphs, the maxima of the overall graph can be recovered with high probability. The authors use an interaction graph rather than a transition graph; this interaction graph uses the expected rewards as edge weights and is thus not applicable in a developmental setting.

Mannor et al. (2004) proposed a bottom-up approach that partitions the global transition graph using agglomerative hierarchical clustering (see Figure 3.5 for an illustration). Clustering can either be made based on topology using a non-standard linkage, which tries to identify clusters of similar size which are only weakly interconnected, or based on the homogeneity of the value function. The approach is non-incremental, i.e., the clustering can be executed only once and thus does not allow identifying skills at different times during learning. The authors have evaluated their approach in the continuous mountain car domain by uniformly discretizing the state space. However, such a uniform discretization is suboptimal since it does not scale well to higher dimensional state spaces due to the "curse of dimensionality".

Kazemitabar and Beigy (2009) have proposed to detect strongly connected components in a directed version of state transition graph and to consider the uni-directional edges connecting two such components as bottlenecks. The authors show empirically that the approach performs well in a small maze world. However, it is unclear how well the approach would perform in domains with stochastic state transitions since edge weights (corresponding to transition probabilities) are not taken into account. A potential advantage of the algorithm is that it is a linear time algorithm in the number of graph edges and nodes.

More recently, Mathew et al. (2012) proposed to identify metastable regions of the state space, which correspond to clusters of the transition graph, by performing spectral graph clustering using robust Perron cluster cluster cluster analysis (PCCA+) (Deuflhard and Weber, 2005), see Section 2.4.3.2. Based on the metastable regions and their connectivity, skill prototypes and a skill tree having the goal state as its root can be defined. Again, the approach is non-incremental.

Ghafoorian et al. (2013) have proposed to identify bottleneck edges using Ant Colony Optimization (ACO) (Dorigo et al., 2006). For this, the environment is explored freely for some time and a state transition graph is formed. Bottleneck edges are identified using ACO applied to this graph for fixed start and goal states. The approach bears some similarity to the work of Menache et al. (2002), with the main difference being that ACO is used instead of graph cuts. Because of the required initial exploration, the method is not very sample-efficient.

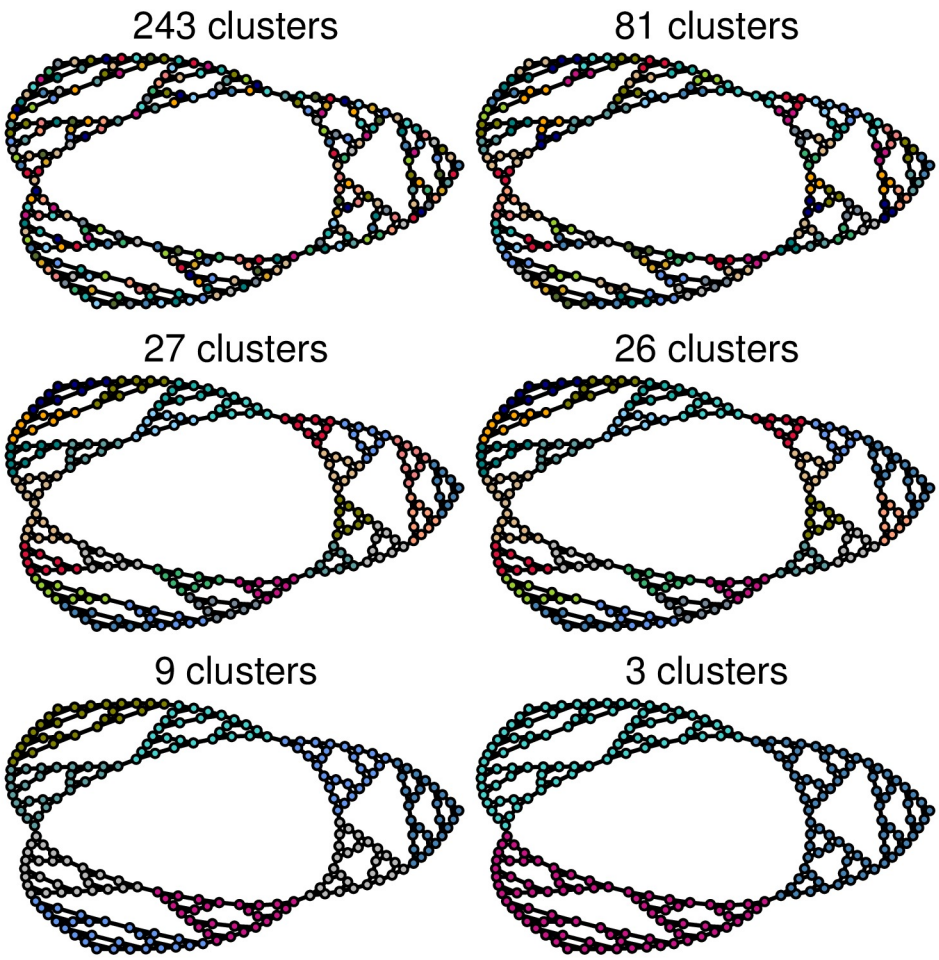


Figure 3.5 – *Illustration of agglomerative hierarchical clustering in "Towers of Hanoi".* Initially, all $N_p = 243$ graph nodes are in separate clusters. Afterwards, two adjacent clusters with minimal linkage are merged in each step. Shown are the clusterings of cardinality 81, 27, 26, 9, and 3. The clustering with cardinality 26 differs from the one with 27 clusters in that the right-most cluster has been merged with one of its neighbor.

Bacon and Precup (2013) have proposed a skill discovery approach based on label propagation. In this work, a transition graph is created based on the transitions observed in 100 episodes of Q-Learning. Groups of densely connected graph nodes, the so-called communities, are identified using weighted label propagation (Raghavan et al., 2007; Pang et al., 2009). The nodes of edges connecting two such communities are used as subgoal states for options with the corresponding communities as initiation set. While the approach is incremental in principle, it must not be invoked too early (the authors propose to wait for 100 episodes) since this may cause to identify false positive bottlenecks. The authors also present a concept for extending the algorithm to domains with continuous state spaces by using the ε-net heuristic (Mahadevan and Maggioni, 2007) for graph construction (see also Section 5.2.1). However, no actual results are shown for the continuous case.

3.3.5 Meta Heuristics

A kind of meta heuristic is presented by Niekum and Barto (2011): the method called "Latent Skill Discovery" builds on top of any of the other heuristics for discovering subgoal states and performs an infinite Gaussian mixture model clustering of these subgoal states for determining how many options are created. This is important when there are many subgoal states that actually correspond to the same option. This can easily happen with frequency-based heuristics which compute statistics for individual states. For graph-based heuristics, however, this issue is less problematic as partitioning the graph into clusters will typically also yield an easy way of associating subgoal states that correspond to the same subproblem.

3.3.6 Summary and Discussion

Which class of heuristics is best suited for achieving the goal stated in Section 1.2? Table 3.1 gives a summary of the discussed works and to which extent they possess the desirable properties identified in Section 3.2.3. One can see that none of the related works exhibit all properties and is sample-efficient at the same time. Please note that sample-efficiency is not an objective criterion; some reasoning for the assessment of the methods is given in the next paragraphs. Please note also that the table does not say anything about the quality of the discovered skills, i.e., whether they are reusable and make learning on higher levels actually easier.

A disadvantage of the solution-based heuristics like SKILLS and PolicyBlocks is that skills can only be defined once a close-to-optimal policy or value function (or an entire set of those) have been learned or require at least that a task can been solved successfully repeatedly like in skill chaining. Thus, identifying and learning useful skills is often slower than learning optimal policies and thus not sample-efficient at all. Accordingly, the learned skills can only be useful later on in subsequent tasks or if the environment changes and policies need to be adapted. While still useful, this limits the utility of skill discovery overly since skill discovery should be able to identify skills before optimal value functions are learned.

	incremental	developmental	automatic granularity	continuous domains	sample-efficient
SOLUTION-BASED					
(Thrun and Schwartz, 1995)	$\sim$	$-$	$-$	$-$	$-$
(Kirchner, 1995)	✓	$-$	✓	$-$	$\circ$
(Kirchner and Richter, 2000)	✓	$-$	✓	$-$	$\circ$
(Pickett and Barto, 2002)	$-$	$-$	✓	$-$	$-$
(Konidaris and Barto, 2009)	✓	$-$	✓	✓	$\circ$
(Hawasly and Ramamoorthy, 2013)	✓	$-$	$-$	✓	$\circ$
FACTORING-BASED					
(Hengst, 2002)	✓	✓	✓	$-$	$\circ$
(Jonsson and Barto, 2006)	$-$	$\sim$	✓	$-$	$++$
FREQUENCY-BASED					
(Digney, 1996)	✓	✓	✓	$-$	$\circ$
(McGovern and Barto, 2001)	✓	$-$	✓	$-$	$\circ$
(Stolle and Precup, 2002)	$-$	$-$	$-$	$-$	$\circ$
(Şimşek and Barto, 2004)	✓	✓	✓	$-$	$\circ$
GRAPH-BASED					
(Menache et al., 2002)	✓	✓	✓	$-$	$++$
(Mannor et al., 2004)	$-$	✓	✓	$\sim$	$+$
(Şimşek et al., 2005)	✓	✓	✓	$-$	$\circ$
(Şimşek and Barto, 2009)	✓	$-$	✓	$-$	$\circ$
(Kazemitabar and Beigy, 2009)	$-$	✓	✓	$-$	$+$
(Mathew et al., 2012)	$-$	✓	✓	$-$	$+$
(Ghafoorian et al., 2013)	$-$	$-$	✓	$-$	$\circ$
(Bacon and Precup, 2013)	$\sim$	✓	✓	$\sim$	$+$
`OGAHC` (Chapter 4)	✓	✓	✓	$-$	$++$
`FIGE` (Chapter 5)	$-$	✓	✓	✓	$+$
`OGAHC` + `IFIGE` (Chapter 6)	✓	✓	✓	✓	$++$

Table 3.1 – *Overview over skill discovery approaches*. Shown is which approach possesses which of the desirable properties identified in Section 3.2.3. ✓ denotes that a method exhibits a property, $-$ that it does not exhibit a property, and $\sim$ that it exhibits a property but with some caveats. Sample-efficiency is assessed ranging from bad (-) over average (o) and good (+) to excellent (++). `OGAHC`, `FIGE`, and `IFIGE` are methods proposed in this work.

Factoring-based heuristics are not applicable in all kinds of MDPs since they assume that there exists conditional independence relations between some state variables or that some state variables vary more frequently than others. Moreover, they require either an initial phase of random exploration (making them less sample-efficient) to determine which state variables change how often or require that the dependencies of state variables are specified by the designer, which limits the autonomy of the agent.

While frequency-based heuristics could in principle identify skills before the emergence of optimal value functions, they require typically repeated-sampling to obtain accurate estimates of the statistics which makes them not very sample-efficient. Furthermore, frequency-based approaches are not easily adapted to continuous domains since this would require that statistics are not computed for single states (there are infinitely many) but for entire regions of the state space. It is unclear how these regions should be defined.

The author considers graph-based approaches to be most promising for achieving the stated goal for the following reasons: (a) Transition graphs are independent of task-specific information like the reward function or goal states since they typically encode solely the possible state transitions. This makes graph-based approaches applicable in developmental settings and allows discovering skills which are reusable in different tasks. (b) Graph-based approaches are typically more sample-efficient than other approaches since generating transition graphs requires solely keeping track of possible state transitions, which is typically easier than learning control policies or accumulate sufficient state-specific statistics. (c) Even if only a small part of the domain has been explored thoroughly, the structure of this part can be represented as a partial transition graph, in which bottlenecks may be detectable long before goal states have been reached or the optimal policy has been learned. (d) Graphs explicitly represent the domain's structure while other heuristics only implicitly capture this structure in policies or state-specific statistics. As the central idea of this thesis is to identify and exploit domain structure, an explicit representation is desirable for analyzing and visualizing the progress of identifying structure.

The main disadvantages of many graph-based approaches are that (a) they can impose computationally hard problems, which can be solved only approximately, (b) they are often not incremental, i.e., generation of the transition graph and skill discovery based on this graph can be performed only once at a specific point in time, and (c) they do not scale well to large or continuous domains since they represent every state explicitly as a graph node. Chapter 4 proposes the novel graph-based skill discovery approach `OGAHC`, which is fully incremental and computational tractable. Chapter 5 presents `FIGE`, a novel approach for transition graph generation which allows extending graph-based skill discovery to continuous domains. Chapter 6 combines `OGAHC` with `IFIGE`, which is an incremental extension of `FIGE`. The resulting approach exhibits property P1-P4 and—as the empirical experiments suggest—is also comparatively sample-efficient.

3.4 PERFORMANCE EVALUATION

While skill learning is a well defined task in the sense that the optimal π_o is determined entirely (though not necessarily uniquely) by the skill prototype $\Psi_o = (I_o, \beta_o, R_o)$, optimality in the context of skill discovery is much harder to define. This is because skill discovery is not done for its own sake but rather to support learning solutions to specific tasks later on. Thus, it can be considered as a kind of second-order learning. Accordingly, theoretical results like asymptotic convergence to optimality are usually not possible in the context of skill discovery. Instead of that, evaluation is typically empirical; for instance, whether the discovered skills have certain desirable properties or how an agent equipped with the discovered skill performs in external tasks. Thus, empirical evaluation is particularly important in the context of skill discovery. Empirical Evaluation of skill discovery methods requires defining reasonable performance metrics and baselines. Different metrics and baselines are discussed in the following subsections.

3.4.1 Performance Metrics

The quality of the discovered skills can be evaluated on different levels, i.e., using different *performance metrics*. Ultimately, acquired skills should help an agent to perform well in an externally imposed task; thus, performance metrics can be defined based on the reward obtained in this external task. The following three performance metrics are based on the external reward:

- $R_{\text{epi}}(t)$ is the accumulated reward obtained by the agent in the t-th episode. $R_{\text{epi}}(t)$ allows evaluating the performance of an agent at a specific point in time, irrespective of what happened before or afterwards. Note that $R_{\text{epi}}(t)$ is often "noisy", i.e., has high variance, since both the agent and the environment can be stochastic. Thus, it is typically more appropriate to use an estimate of the expected value $\mathbb{E}[R_{\text{epi}}(t)]$ rather than $R_{\text{epi}}(t)$ directly. One way of estimating $\mathbb{E}[R_{\text{epi}}(t)]$ is to perform several independent runs and use the mean of the resulting values for $R_{\text{epi}}(t)$ as estimate. An alternative is to use the moving window average $R^k_{\text{mwa}}(t) = \frac{1}{2k+1} \sum_{i=-k}^{k} R_{\text{epi}}(t+i)$ with window length $2k+1$ as an estimate of $\mathbb{E}[R_{\text{epi}}(t)]$. This quantity has less variance for large k. However, $R^k_{\text{mwa}}(t)$ may not always be a good estimate for $\mathbb{E}[R_{\text{epi}}(t)]$ (it may be biased); in particular, if the behavior of the agent changes strongly within the time window. Thus, in general it is preferable to estimate $\mathbb{E}[R_{\text{epi}}(t)]$ by averaging over a large number of independent runs. However, if these runs are computationally expensive and only a small number of runs can be conducted, this approach can be combined with the moving window average approach for small k.

- $R^\infty = \lim_{t\to\infty} \mathbb{E}[R_{\text{epi}}(t)]$ is the expected accumulated reward per episode in the limit. This metric is insensitive to the speed of learning and only captures the asymptotic performance of a learning system. One typically approximates $R^\infty \approx R_{\text{epi}}(t^*)$ for

some sufficiently large t^* since the limit can not be derived from empirical results. Normally, this metric favors methods which explore more strongly initially and have policy representations with higher capacity (more free parameters).

- $R^t_{\text{acc}} = \sum_{i=1}^{t} R_{\text{epi}}(i)$ is the reward accumulated by the agent during the first t episodes. Related is the average return $R^t_{\text{avg}} = R^t_{\text{acc}}/t$, which makes it easier to relate the accumulated reward to the outcome of a single episode. Typically (for not too large t), these metrics favor methods which can balance exploration and exploitation intelligently, i.e., explore enough to find high quality policies fast but also exploit these policies such that sufficient reward is accumulated. Moreover, methods which use policy representations with small capacity (less free parameters) that can nevertheless represent close-to-optimal behavior perform typically good under these metrics. In the context of multi-task and transfer learning, one is particularly interested in learning speed since the main objective is to accelerate learning on new tasks as a result of previous experience (Perkins and Precup, 1999). Learning speed is captured well by these metrics for not too large t.

Note that these performance metrics based on the external reward are in no sense specific to evaluating skill discovery methods; they can be used to assess the performance of any RL method. Besides these generic performance metrics based on the task performance, further metrics can be defined which are more specific for evaluating skill discovery methods. For graph-based skill discovery, one can propose specific metrics for assessing the quality of a state transition graph and the quality of a graph clustering.

For evaluating the quality of a learned transition graph, the *graph likelihood* metric considers a transition graph as a generative model for transitions in the domain. For a given graph G, the probability $p(T|G)$ of a set of transitions T can be computed using the graph as generative model. If the transitions are sampled from the domain, this probability should be comparatively large since one can assume that T is comparatively likely in the actual domain. One can consider the likelihood $L_T(G) = p(T|G)$ of graph G for given transitions T as metric for the quality of G: the larger the likelihood of a graph, the better explains this graph the observed transitions and thus the domain's dynamics. For details please refer to Section 4.2.1, where the graph likelihood for discrete MDPs is defined, and to Section 5.3.1 for an extension to MDPs with continuous state spaces.

One way of assessing the quality of a partition $\mathcal{P}$ of a graph G is the normalized cut $\text{NCut}(\mathcal{P})$, where smaller $\text{NCut}(\mathcal{P})$ corresponds to better partitions $\mathcal{P}$ (see Section 2.4.2). An alternative metric can be defined if a ground-truth partition $\mathcal{P}_{GT}$ is available,[1] which is (by construction or by domain knowledge) known to be optimal. For this, the agreement of two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ of the graph nodes V is defined using the accordance ratio

$$\text{acc}(\mathcal{P}_1, \mathcal{P}_2) = \frac{1}{|V|^2} \sum_{v,\bar{v} \in V} \delta(\delta([v]_{\mathcal{P}_1}, [\bar{v}]_{\mathcal{P}_1}), \delta([v]_{\mathcal{P}_2}, [\bar{v}]_{\mathcal{P}_2})), \tag{3.1}$$

[1] Note that the ground-truth partition is not necessarily the one with the minimal NCut on G, i.e., $\mathcal{P}_{GT} \neq \arg\min_{\mathcal{P}} \text{NCut}(\mathcal{P})$, in particular if G does not reflect the domain's dynamics well.

i.e., the ratio of element-pairs on which $\mathcal{P}_1$ and $\mathcal{P}_2$ agree on assigning them to the same or to different clusters (recall $\delta(x,y) = 1$ if $x = y$ else 0 and that $[v]_{\mathcal{P}} = p_i$ implies that $v \in p_i$ and $p_i \in \mathcal{P}$, cf. Section 2.4.1). Using this, one can define the quality of a partition $\mathcal{P}$ via its *ground-truth accordance*: $\text{acc}_{GT}(\mathcal{P}) = \text{acc}(\mathcal{P}, \mathcal{P}_{GT})$. Comparing these two metrics, $\text{NCut}(\mathcal{P})$ evaluates the partition for a given graph in isolation, while $\text{acc}_{GT}(\mathcal{P})$ assesses both the quality of the graph partition and how well the graph captures the actual domain. $\text{NCut}(\mathcal{P})$ is thus an appropriate metric for comparing methods for graph clustering while acc_{GT} can assess the whole skill discovery process including graph construction and graph clustering.

3.4.2 Baselines

For defining useful *baselines*, one can create modifications and simplifications of the devised learning architecture and evaluate the respective performance in externally defined tasks. For instance, in the case of a 3-layer, hierarchical architecture, the architecture consists of a high-level policy which can invoke skills that are discovered and learned autonomously. The first baseline is obtained by removing the necessity that skills are discovered autonomously: in the "*predefined skills*" baseline, the skill prototypes Ψ_o are manually identified and available to the agent from the very beginning. Thus, the agent only has to learn the skill policies for these predefined skills and the high-level policy. This should simplify the learning problem and result in an upper baseline on the performance of any skill discovery approach. Note that this baseline is only applicable in domains where one can manually decide on useful skill prototypes. A further simplification is obtained by the "*prelearned skills*" baseline: in this baseline, the agent is provided not only with skill prototypes from the very beginning but also with the corresponding optimal skill policies. Thus, the agent only has to learn the high-level policy for given skills.

A further baseline is the performance of the "*monolithic*" approach that learns a flat, non-hierarchical policy without skills but uses otherwise the same learning mechanisms. This baseline allows studying in which scenarios skill discovery and hierarchical policies pay off, i.e., perform better than a more simple, flat policy. Further baselines can be devised for specific skill discovery approach, for instance, an incremental skill discovery approach can be compared with its non-incremental counterpart (see, e.g., Figure 4.14) or an approach which chooses the skill granularity automatically can be compared to an approach with fixed granularity.

As pointed out by Hengst (2002), learning (skill) policies in a sample-efficient way is largely orthogonal and complementary to decomposing a problem, i.e., skill discovery. Thus, the empirical studies of this work use mainly simple one-step TD-based learning, potentially combined with intra-option learning. Combining skill discovery with, e.g., model-based RL methods or eligibility traces, would increase sample efficiency (at the cost of increased computational demands); however, all skill discovery methods would typically benefit from this effect similarly and thus, the qualitative comparison of skill discovery methods would remain unchanged. Thus, one can adhere to plain one-step TD learning for reasons of simplicity.

4
Incremental Graph-Based Skill Discovery

> "Progress, of the best kind, is comparatively slow. Great results cannot be achieved at once; and we must be satisfied to advance in life as we walk, step by step."
>
> Samuel Smiles

4.1 Introduction

This chapter presents the novel graph-based skill discovery approach `OGAHC`, which is incremental, insensitive to the explorative behavior of the agent and to different degrees of stochasticity of the environment, and can identify skills that are independent of the current task. The aim of `OGAHC` is thus to achieve subgoal S1 stated in Section 1.2.

The general structure of a graph-based skill discovery approach is shown in Figure 4.1: first, a transition graph G is generated for a set of transitions T that have been sampled by the agent in an environment (see Section 4.2.1). Thereupon, a partition $\mathcal{P}$ of the graph is generated using graph clustering (see Section 4.2.3) based on an externally defined linkage criterion l (see Section 4.2.2). Based on this partition, skill prototypes Ψ_o are generated (see Section 4.2.4), which form the basis for skill learning and for learning a hierarchical task policy using compositional learning. Note that incremental skill discovery requires executing all of these components repeatedly. This requires in turn that some components have an internal memory for storing, e.g., the current transition graph and the corresponding partition.

The main contributions of this chapter are (a) a means for transition graph generation and a linkage criterion, which are "off-policy" and robust against domain stochasticity, and (b) an incremental graph clustering approach. This chapter is restricted to discrete MDPs; an extension of the concepts to continuous domains is presented in Chapter 5 and 6.

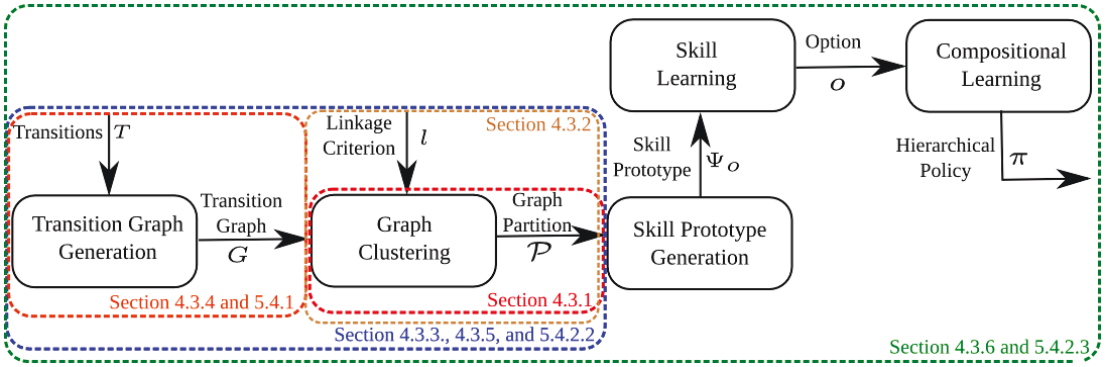


Figure 4.1 – *Data flow diagram of graph-based skill discovery*. The boxes in the lower row (transition graph generation, graph clustering, and skill prototype generation) are essential parts of graph-based skill discovery, while the boxes in the upper row (skill learning and compositional learning) constitute the rest of a hierarchical RL architecture. Note that all stages of the diagram are executed continuously and interwoven. Dashed boxes show in which sections subparts of the overall system are evaluated separately.

4.2 METHODS

We adopt the concept of identifying bottlenecks of a domain as basis for skill discovery and investigate approaches that identify such bottlenecks based on the sample transition graph using agglomerative clustering. We base skill discovery solely on the MDP's state transition probabilities $P^a_{ss'}$ and do not take the expected rewards $R^a_{ss'}$ into account since we consider $R^a_{ss'}$ to encode a task and aim at identifying task-independent skills that can be discovered in developmental settings and can be reused in different tasks (cf. Property P2 in Section 3.2.3).

4.2.1 Transition Graph Generation

We construct the *sample transition graph* $G = (V, E, w)$, which is a directed, weighted graph, that is used as proxy for the actual unknown transition graph $\mathbb{G}$, as follows: for a given set of n transitions $T = \{(s_i, a_i, s'_i)\}_{i=1}^n$ sampled from an MDP, we set $V = \{s \mid \exists (s_i, a_i, s'_i) \in T : s_i = s \vee s'_i = s\}$, i.e., to the set of all states that have been visited, and $E = \{(s,s) \mid \exists (s_i, a_i, s'_i) \in T : s_i = s \wedge s'_i = s'\}$, i.e., to the set of all one-step transitions that have been observed. For each action $a \in A$, we add the edge attribute $n^a_{vv'} = \sum_{i=1}^n \delta((v,a,v'),(s_i,a_i,s'_i))$ to any edge $(v,v') \in E$ and the node attribute $n^a_v = \sum_{i=1}^n \delta((v,a),(s_i,a_i))$ to any node $v \in V$ where $\delta(x,y) = 1$ if $x = y$ else 0. In other words, n^a_v counts how often action a has been invoked in state v and $n^a_{vv'}$ how often action a has been invoked in state v and the successor state has been v'.

Different choices for the weights $w(e)$ of edge $e = (v,v')$ have been proposed. While Mannor et al. (2004) used essentially uniform edge weights $w_{\text{uni}}(e) = 1 \, \forall e \in E$, Şimşek et al. (2005) proposed the edge weights $w_{\text{on}}((v,v')) = \sum_a n^a_{vv'}$, i.e., how often the transition $v \to v'$ has been observed in T. While w_{uni} ignores both the stochasticity of

the environment and the action preferences of the agent, w_{on} take both into account (i.e., it is "on-policy" with regard to the sampling policy). We argue that in order to identify domain properties like bottlenecks, one should take the stochasticity into account but be independent of the sampling policy, since stochasticity is a domain property while the agent's sampling policy is not. Thus, we would like to estimate "off-policy" weights w_{off} from T, i.e., weights which are independent of the policy that was used to sample T.

For this, we consider a graph as a generative model for transitions and choose graph weights w such that the likelihood $L_T(G) = p(T|G)$ of the resulting graph G for a set of observed transitions $T = \{(s_i, a_i, s_i')\}_{i=1}^n$ becomes maximal. Since there is a one-to-one correspondence between states s_i and graph nodes v_i, one can also consider T as a set of transitions on the graph, i.e., $T = \{(v_i, a_i, v_i')\}_{i=1}^n$. We consider transitions to have been sampled from the graph using the following generative process: (1) Sample a graph node $v \in V$ uniform randomly, i.e., $p(v) = 1/|V|$. (2) Sample an action a in node v uniformly from the action space A, i.e., $p(a|v) = p(a) = 1/|A|$. (3) Sample the "successor node" v' according to the graph's edge weights, i.e., $p(v'|v,a) = w_{vv'}$, where we use the shorthand notation $w_{vv'} = w((v,v'))$. Note that this is based on the simplifying assumption $v' \perp a \mid v$, i.e., that the successor node v' is independent of the chosen action a given the current node v'. While this assumption is over-simplifying, it enforces that edge weights cover the effects of all possible actions jointly. The likelihood is thus $L_T(G) = p(T|G) = \prod_{i=1}^n p((v_i, a_i, v_i')|G) = \prod_{i=1}^n p(v_i'|v_i, a_i) p(a_i|v_i) p(v_i) = \frac{1}{|V|^n |A|^n} \prod_{i=1}^n w_{v_i v_i'}$. If the number of graph nodes is fixed, the likelihood depends thus solely on the transition graph's weights and can be denoted by $L_T(w)$

Appendix A.1 shows that the maximum likelihood estimate of the edge weights for this generative model is $w_{\text{off}}((v,v')) = \frac{1}{|A|} \sum_a \frac{n_{vv'}^a}{n_v^a}$ when correcting for the bias of the sampling policy via importance sampling. Since these weights take the policy's sampling bias into account, they can be considered to be "off-policy". When ignoring the sampling bias of π, one obtains weights proportional to the "on-policy" weights w_{on} accordingly. Note that the derivation of $w_{\text{off}}((v,v'))$ assumes that all actions $a \in A$ can be invoked in all states, i.e., $p(a|v) = p(a) = 1/|A|$. If only a subset $A_v \subsetneq A$ of the actions can be executed in state v, the normalizing constant would become $1/|A_v|$. See Section 4.3.4 and Section 4.3.5.2 for an empirical comparison of the different choices for the edge weights.

Note that other choices for edge weights are possible; in particular, it can be desirable to incorporate rewards into the edge weights in situations where $R_{ss'}^a$ contains structure that shall be exploited for skill discovery. For instance, Şimşek and Barto (2009) used the expected negative reward as edge weight and Bacon and Precup (2013) suggested for future work to choose edge weights based on values of neighboring states, which may allow grouping states of similar current valuation together. We do not discuss these approaches in more detail since we consider $R_{ss'}^a$ to encode a task and aim at identifying task-independent skills that can be discovered in developmental settings and can be reused in different tasks (Property P2).

4.2.2 Linkage Criteria and Bottlenecks

This section formalizes the notion of a domain *bottleneck*. For this, we first define the *boundary* of two connected areas of the state space. Let $G = (V, E, w)$ be a transition graph, regardless of whether it is the domain's true transition graph $\mathbb{G}$ or a learned sample transition graph. The boundary of two disjoint node sets $A, B \subset V$ is defined as the edges which connect A and B in either direction: $b_E(A,B) = E \cap (A \times B \cup B \times A)$. The boundary states are defined accordingly via $b_V(A,B) = \{v \in V \mid \exists v' \in V : (v,v') \in b_E(A,B)\}$. Note that this definition is similar to the one given by Mathew et al. (2012).

We now introduce the concept of a *linkage criterion*[1] $l : 2^V \times 2^V \to \mathbb{R}$, where 2^V denotes the power set of V. A linkage criterion gives a quantitative assessment if the boundary of two connected, disjoint subgraphs $A, B \subset V$ forms a bottleneck in G. The larger the linkage $l(A,B)$, the more evidence the criterion offers for a bottleneck between A and B. By choosing a threshold ψ, one can create a binary criterion for "bottleneckness" by identifying a bottleneck between A and B if $l(A,B) > \psi$. Thus, we define the domain's *bottleneck edges* as follows:

$$b^* = \{(v,v') \mid \exists\, A,B \subset V : A \cap B = \emptyset \wedge b_E(A,B) \neq \emptyset \wedge (v,v') \in b_E(A,B) \wedge l(A,B) > \psi\}$$

Thus, b^* are the edges connecting two disjoint subgraphs of G which have a linkage larger than threshold ψ.

As base for linkages, we define the (directed) *connectivity mass* c_m of subgraphs A and B as $c_m(A,B) = \sum_{e \in E \cap (A \times B)} w(e)$. The connectivity mass gets large for two large, densely connected subgraphs with large edge weights. Mannor et al. (2004) proposed a linkage criterion for bottleneck identification in transition graphs that is defined as follows (using our notation and $|A|$ being the number of vertices in subgraph A):

$$M(A,B) = \frac{\min(|A|,|B|)\log(\max(|A|,|B|))}{c_m(A,B) + c_m(B,A)}. \tag{4.1}$$

This linkage is based on uniform edge weights such that the denominator is equal to the number of edges between A and B in G. The linkage thus assigns large values to subgraphs of similar sizes that are only weakly interconnected. The linkage M has no intuitive interpretation; the authors have chosen it based on an empirical comparison of several candidate linkages.

A better motivated criterion was given by Şimşek et al. (2005), who proposed the normalized cut NCut (see Section 2.4.2) as basis for bottleneck identification. This is intuitively reasonable as the NCut corresponds to the probability that a randomly behaving agent transitions in one time step from a state in subgraph A to a state in subgraph B or vice versa. If this probability is small, the connection of A and B can be seen as a bottleneck. Şimşek et al. (2005) proposed the $\hat{N}_{cut}$ criterion which is a

[1] Note that despite the term "linkage", a large linkage generally corresponds to two very dissimilar, distant, or separated clusters.

symmetrized version of standard NCut and defined as follows (using our notation):

$$\hat{N}_{cut}(A,B) = \frac{1}{2}\left(\frac{c_m(A,B)+c_m(B,A)}{c_m(A,V)+c_m(B,A)} + \frac{c_m(B,A)+c_m(A,B)}{c_m(B,V)+c_m(A,B)}\right), \tag{4.2}$$

where we added the factor $\frac{1}{2}$ to ensure that $\hat{N}_{cut}(A,B)$ corresponds to a proper probability. The authors used a top-down spectral clustering algorithm to find graph cuts that minimize $\hat{N}_{cut}$; in order to use it as a linkage criterion where large values indicate a bottleneck, we shall use $l(A,B) = -\hat{N}_{cut}(A,B)$. The authors used the criterion with on-policy edge weights w_{on}; we claim that it should rather be used with off-policy weights w_{off} (see Section 4.3.5).

While the $\hat{N}_{cut}$ linkage is well grounded, the choice of the threshold ψ remains open. However, this choice is inherent in all bottleneck-based skill discovery methods since "bottleneckness" is not a binary property but rather a matter of degree. Since skill discovery typically requires a binary decision on whether two state space areas are separated by a bottleneck, the choice of a threshold is inevitable. For the $\hat{N}_{cut}$ linkage, the choice of the threshold ψ is simplified by the observation that $-\psi$ has an intuitive interpretation:[1] it specifies that the connection of two subgraphs is a bottleneck if the probability that the agent transitions in one time step from a state in subgraph A to a state in subgraph B or vice versa is below $-\psi \in [0,1]$. This connection makes the choice of ψ easier for NCut-based linkages than for, e.g., the M-linkage. Section 4.3.2 gives some empirical results on the influence of the parameter ψ onto the resulting partition.

4.2.3 Incremental Graph Clustering

In order to identify all bottlenecks (defined via a linkage criterion l and a corresponding threshold ψ) of a transition graph G, we determine a partition $\mathcal{P}^*$ of minimal cardinality of the graph nodes V into disjoint clusters p_i such that neither of the subgraphs induced by these clusters contains a bottleneck. Formally:

$$\mathcal{P}^* = \operatorname*{arg\,min}_{\mathcal{P}\in\mathbb{P}(V)} |\mathcal{P}| \qquad \text{s.t.} \max_{p\in\mathcal{P},\, q\subsetneq p} l(p\setminus q, q) \leq \psi, \tag{4.3}$$

where $\mathbb{P}(V)$ is the set of all partitions of V. The constraint $\max_{p\in\mathcal{P},\, q\subset p} l(p\setminus q,q) \leq \psi$ guarantees that no cluster $p \in \mathcal{P}^*$ can be split further into two parts q and $p\setminus q$ such that the boundary $b_e(p\setminus q, q)$ forms a bottleneck (which would correspond to $l(p\setminus q,q) > \psi$). On the other hand, the minimal cardinality objective "$\operatorname*{arg\,min}_{\mathcal{P}\in\mathbb{P}(V)} |\mathcal{P}|$" ensures that the boundary $b_e(p_i,p_j)$ between two clusters $p_i, p_j \in \mathcal{P}$ is actually a bottleneck (otherwise p_i and p_j would have been merged). Note that $\mathcal{P}^*$ need not necessarily be uniquely defined; in this case ties are broken randomly.

[1] This thesis considers the threshold $-\psi$ since $-\hat{N}_{cut}$ is used as a linkage.

Alternatively to specifying ψ, one could also fix the number of clusters k and identify a partition $\mathcal{P}^*$ with $|\mathcal{P}^*| = k$ such that the maximal intra-cluster linkage l is minimized, i.e., the k clusters are internally maximally strong connected:

$$\mathcal{P}^* = \arg\min_{|\mathcal{P}|=k} \max_{p \in \mathcal{P},\, q \subset p} l(p \setminus q, q). \tag{4.4}$$

This formulation generally requires specifying the number of clusters k in advance. For many problems, this is non-trivial as knowing how many clusters (bottlenecks) exist may be as hard as knowing the bottlenecks themselves. Thus, the formulation in Equation 4.3 is more appealing since it only requires specifying an upper bound on the linkage. In the case of the NCut linkage, this effectively means that one has to specify how improbable a transition between two state sets must be under a random walk to consider the connection of these two node sets as a bottleneck.

Since finding the optimal solutions $\mathcal{P}^*$ for both problem formulations is $\mathcal{NP}$-hard for the most interesting linkage criteria such as the NCut (Wagner and Wagner, 1993), one has to resort to an approximate graph clustering approach similar to the ones discussed in Section 2.4.3. These are given for the problem formulation in Equation 4.4, i.e., for given number of clusters k. However, an extension to the problem formulation in Equation 4.3 using ψ is straight-forward for agglomerative hierarchical clustering (see Section 4.2.3.1).

All approaches presented in Section 2.4.3 are non-incremental, i.e., they can be executed only once for a fixed transition graph. When the true transition graph $\mathbb{G}$ is not known but learned from experience, this raises the question at what specific point in time clustering should be performed. This presents a trade-off: on the one hand, one would like to perform the clustering as early as possible in order to maximize the impact of the discovered skills during learning; on the other hand, performing the clustering too early may result in false positive bottlenecks that are due to considerable deviations of the estimated transition graph from the true transition graph. For instance, Bacon and Precup (2013) explored the environment for 100 episodes before constructing and clustering the transition graph. Thus, no skills could be discovered during the first 100 episodes or afterwards.

To alleviate this problem, it would be highly desirable to use an incremental ("online") clustering algorithm instead, which can be invoked several times during learning and can find more and more bottlenecks over time, i.e., exhibit property P1 from Section 3.2.3. This would remove the requirement to choose one specific point in time for skill discovery. In order to make such an incremental clustering algorithm useful for skill discovery, it has to fulfill the following properties: a) Subsequent executions of the clustering should yield consistent partitions, i.e., bottlenecks identified in one invocation of the clustering should persist in subsequent ones. Otherwise, skills corresponding to these bottlenecks (see Section 4.2.4) would need to be deleted (causing an undesirable loss of learned knowledge) or to be modified to a new target, which would require re-learning and might have a detrimental effect onto higher-level policies which are based on these skills. b) The algorithm should not identify bottlenecks in parts of the domain

Algorithm 4.1 Constrained Agglomerative Clustering

1: **Input:** graph G, constraint set C, linkage criterion l, threshold ψ
2: $\mathcal{P} = \{\{v\} | v \in V\}$ # Initial partition: one cluster per vertex
3: **loop**
4: # Merge-candidates that fulfill all constraints. Note that the symbol $\bigwedge_{c \in C}$ in line 5 denotes a logical "and" over all constraints.
5: $M_c = \{(p_1, p_2) \mid (p_1, p_2) \in (\mathcal{P} \times \mathcal{P}) \wedge \bigwedge_{c \in C} c(p_1, p_2)\}$
6: $p_1^*, p_2^* = \underset{(p_1,p_2) \in M_c}{\arg\min}\ l(p_1, p_2)$ # Find merge-candidate with minimal linkage in G
7: **if** $l(p_1^*, p_2^*) > \psi$: **return** $\mathcal{P}$ # No more densely connected clusters $\rightarrow$ return partition
8: $\mathcal{P} = (\mathcal{P} \setminus \{p_1^*, p_2^*\}) \cup \{p_1^* \cup p_2^*\}$ # Merge p_1^* and p_2^*
9: **end loop**

which it has not explored sufficiently because apparent bottlenecks in its estimate of the transition graph might be just artifacts of its exploration strategy rather than true properties of the domain. Note that specifying the number of clusters k beforehand instead of ψ is not practical in incremental clustering since it does not allow the implicit increase of granularity of the partitions, i.e., the number of bottlenecks, over time.

We now extend agglomerative hierarchical clustering such that it can be used in an incremental fashion. For this, we first add constraints to the algorithm (see Section 4.2.3.1) and present then `OGAHC`, which realizes a graph clustering method that possesses the properties (a) and (b) stated above (see Section 4.2.3.2).

4.2.3.1 Constrained Agglomerative Clustering

This section extends the agglomerative hierarchical clustering presented in Algorithm 2.4 by allowing (optionally) specifying constraints about which clusters can be merged. Moreover, the algorithm does not require specifying the resulting number of clusters k but rather the desired "strength" ψ of a bottleneck. Constraints are defined as boolean functions $c : 2^V \times 2^V \rightarrow \mathbb{B}$, which map two (typically disjoint) sets of nodes onto a boolean. The semantic of such a constraint is that two clusters $p_1, p_2 \subset V$ can only be merged by the agglomerative clustering if $c(p_1, p_2)$ is true for each constraint c. The constraints should be symmetric, i.e., $c(p_1, p_2) = c(p_2, p_1)$.

Pseudo-code is given in Algorithm 4.1: the algorithm starts by assigning each graph node into a separate cluster (line 2). Afterwards it greedily identifies the pair of clusters p_1^*, p_2^* that has minimal linkage and fulfills all constraints $c \in C$ (line 5-6). If the linkage of the pair p_1^* and p_2^* is above the threshold ψ, there is no further pair of clusters in $\mathcal{P}$ that is not separated by a bottleneck; accordingly, $\mathcal{P}$ is returned as the best greedy approximation of $\mathcal{P}^*$ that can be obtained under the given constraints (line 7). If the linkage is not greeter than ψ, p_1^* and p_2^* are merged (line 8) and the algorithm continues in line 5.

Note that the pseudo-code given in Algorithm 4.1 should not be implemented one-to-one; a more efficient implementation would maintain a priority-queue of merge candidates M_c where the linkage acts as priority. Furthermore, we have skipped the

Algorithm 4.2 On-line Graph-Based Agglomerative Hierarchical Clustering (OGAHC)

1: **Input:** linkage criterion l, parameters ρ, ψ, m
2: $\mathcal{P}_0 = \emptyset$
3: $G = \emptyset$ # Transition graph $G = (V, E, w)$ is initially empty, constructed from experience later
4: $t = 0$ # Count how many iterations have been conducted
5: **loop**
6: $T = \{(s_i, a_i, s'_i)\}_{i=1}^{m} = \text{ACT(m)}$ # Act for m steps and observe transitions T
7: $G = \text{UPDATE}(G, T)$ # Update transition graph with transitions T
8: $G' = \text{SMOOTH}(G, \rho)$ # Add virtual transitions for under-explored nodes
9: # Build constraint set C which ensures that partitions are consistent
10: # Note that "lambda" is used as a shorthand keyword for defining a function
11: $C = \{\textbf{lambda } p_i, p_j : (p_i \times p_j) \cap E \neq \emptyset\}$ # Merge only clusters p_i, p_j connected in G
12: **for** $(p_A, p_B) \in \mathcal{P}_t \times \mathcal{P}_t$ with $p_A \neq p_B$ **do**
13: # Must not merge two clusters with elements that had not been merged in last iteration
14: $C = C \cup \{\textbf{lambda } p_i, p_j : \neg[(p_i \cup p_j) \cap p_A \neq \emptyset \wedge (p_i \cup p_j) \cap p_B \neq \emptyset]\}$
15: # Partition G' using constrained agglomerative clustering (CAC, Algorithm 4.1)
16: $\mathcal{P}_{t+1} = \text{CAC}(G', C, l, \psi)$ # Note: $\mathcal{P}_{t+1} \leq \mathcal{P}_t$
17: # Use $\mathcal{P}_{t+1}$, e.g., for defining skill prototypes...
18: $t = t + 1$
19: **end loop**

optional construction of an explicit dendrogram given in Algorithm 2.4. Adding this to Algorithm 4.1 would be straight-forward.

4.2.3.2 On-line Graph-Based Agglomerative Hierarchical Clustering

We propose now the method *On-line Graph-Based Agglomerative Hierarchical Clustering* (OGAHC, see Algorithm 4.2). OGAHC addresses the two main challenges for an incremental graph clustering approach identified above: the first challenge is that subsequent partitions obtained by graph clustering must be consistent, i.e., bottlenecks identified in one invocation of the clustering should persist in subsequent ones. We call a sequence of partitions $\mathcal{P}_0, \ldots, \mathcal{P}_t$ consistent if $\mathcal{P}_t \leq \mathcal{P}_{t-1} \leq \cdots \leq \mathcal{P}_0$ (see Section 2.4.1 or Jonsson and Barto (2006) for a definition of this partial ordering of partitions). In other words: two nodes that have been assigned to different clusters in partition $\mathcal{P}_{t_1}$ must not be assigned to the same cluster in any subsequent partition $\mathcal{P}_{t_2}$ with $t_2 > t_1$. Thus, a consistent sequence of partitions ensures that bottlenecks identified in one invocation of the clustering persist in subsequent ones.

OGAHC ensures that the sequence of generated partitions is consistent by maintaining an increasing set of constraints, which must be obeyed by the clustering. For each pair of clusters $p_A, p_B \in \mathcal{P}_t$ with $p_A \neq p_B$, one constraint for the computation of $\mathcal{P}_{t+1}$ is created. This constraint ensures that no two cluster p_i and p_j are merged during the creation of $\mathcal{P}_{t+1}$, whose union has a non-empty intersection with both p_A and p_B (see line 14).

The second challenge for an incremental graph clustering is that bottlenecks must not be identified prematurely. This problem arises since graph clustering typically acts on the sample transition graph G and not on the "true" transition graph $\mathbb{G}$. When

Algorithm 4.3 Graph Smoothing (SMOOTH)

Input: Graph $G = (V, E, w)$, parameter ρ
$G' = \text{COPY}(G)$ # Modify only copy of graph
$k = \max(5, \rho)$ # Number of nearest neighbors considered. At least 5.
for $v, a \in V \times A$ with $n_v^a < \rho$ **do** # For any under-explored pair of node v and action a
  # Determine k nearest neighbors of v in V (according to, e.g., geodesic distance)
  $V' = \text{NN}_V^k(v)$
  # Add virtual transitions between v and its nearest neighbors.
  for $v' \in V'$ **do**
    $n_{vv'}^a \mathrel{+}= (\rho - n_v^a)/k$
  $n_v^a = \rho$
return G'

using the off-policy weights and employing an appropriate exploration strategy, G converges to $\mathbb{G}$ in the limit of an infinite number of sample transitions (see Section 4.3.4). However, for finite number of transitions, there are typically differences between G and $\mathbb{G}$. Thus, it may happen for two clusters $p_i, p_j \in V$ that $l_G(p_i, p_j) > \psi$ even though $l_{\mathbb{G}}(p_i, p_j) \leq \psi$ (a "false positive" bottleneck detection).[1] On the other hand, it may also happen that $l_G(p_i, p_j) \leq \psi$ even though $l_{\mathbb{G}}(p_i, p_j) > \psi$ (a "false negative" bottleneck detection). For an incremental graph clustering approach, a false positive is more critical since the consistency constraints enforce that this false positive is retained in all subsequent iterations. In contrast, a false negative can be corrected in a later invocation of the algorithm. Since there is a trade-off between both types of errors, i.e., one type of error can be reduced at the cost of increasing the other type of error, it is desirable that an incremental graph clustering algorithm is *conservative* initially. Being conservative means in this context that a bottleneck is detected only when there is considerable evidence in favor of it. By this, the number of false positives is reduced and the number of false negatives is increased.

This initial conservativeness is realized in `OGAHC` by means of a heuristic that may be termed as "assume dense local connectivity in the face of uncertainty". Technically, instead of working on the maximum likelihood estimate G of the transition graph, the clustering is performed on a "smoothed" transition graph $G' = \text{SMOOTH}(G, \rho)$. In G', for each node $v \in V$ and action a with $n_v^a < \rho$, i.e., state-action pairs which have not been explored sufficiently, virtual transitions to nearby states are "imagined". Technically, $n_{vv'}^a$ is incremented by $(\rho - n_v^a)/k$ for any v' that is among the $k = \max(5, \rho)$ nearest neighbors[2] (see Algorithm 4.3). The free parameter ρ of the algorithm determines how conservative the algorithm is with larger values of ρ corresponding to increased conservativeness. The nearest neighbors can be computed based on any measure of state similarity; if no such measure exists, one could alternatively use the k graph nodes with minimal geodesic distance d_G to v in G.

[1] We denote the linkage on the true transition $\mathbb{G}$ by $l_{\mathbb{G}}$ and the linkage on the sample transition graph by l_G.

[2] We consider at least 5 nearest neighbors to avoid that virtual transitions are too local.

Working on the smoothed transition graph reduces the risk of false positive bottleneck detections since the linkage value of under-explored regions of the state space is typically decreased by adding virtual transitions. Thus, fewer clusters are formed in early invocations of the algorithm. Over time, n_v^a increases, fewer virtual transitions are added, the connectivity decreases, the typical linkage of subgraphs increases, and thus, more clusters are formed. An empirical evaluation of graph smoothing showing this effect is given in Section 4.3.3.

The main loop of `OGAHC` in Algorithm 4.2 can be summarized as follows: the agent acts for m steps in the environment and records the resulting state transitions (line 6). These transitions are added to the transition graph using the methods discussed in Section 4.2.1 (line 7). Thereupon, virtual-transitions are added to under-explored parts of the sample transition graph according to the graph-smoothing heuristic (line 8). Constraints are created that ensure that only clusters whose corresponding subgraphs are connected in G can be merged (line 11) and that subsequent partitions are consistent (line 12-14). Finally, constrained agglomerative clustering (CAC) is used to create a partition of the smoothed transition graph (line 16). This partition can be used, e.g., to update the set of skill prototypes. Thereupon, the agent acts for the next m steps and the procedure is repeated. Note that sparsity properties of G—which correspond to a reduced runtime of the constrained agglomerative clustering algorithm because of less merge candidates—are maintained in the smoothed graph G'.

4.2.3.3 Illustration

We give a brief illustration of incremental bottleneck identification using `OGAHC` in the "Towers of Hanoi" domain (see Section 3.1.1). Figure 4.2 shows the constructed transition graphs and the resulting partitions after different number of steps in an example run. The parameters of `OGAHC` have been set to $\psi = -0.003$ and $\rho = 3$, and the clustering was performed every $m = 2500$ steps. Target nodes of virtual transitions have been sampled according to the graph structure, i.e., virtual transition are more likely between nodes with small geodesic distance d_G in the graph than between more distant nodes. One can see that the algorithm is conservative in bottleneck identification: after 5000 steps, no bottleneck is detected even though the transition graph provides some evidence in favor of one (indicated by an annotation in the figure). However, since some parts of the neighborhood of this bottleneck have not been explored sufficiently, graph smoothing generates virtual transitions that bypass this bottleneck. Thus, the corresponding bottleneck is not detected at this point in time. After 7500 steps however, when more exploration was conducted, `OGAHC` adds fewer virtual transitions, the linkage becomes larger than ψ, and the partition is split accordingly. Similarly, after 12500 steps two further bottlenecks are identified. One of these bottlenecks (between the "red" and the "blue" cluster) is suboptimal since three nodes are labeled "blue" even though they would fit better into the "red" cluster. This is an example where agglomerative hierarchical clustering does not find the optimal partition due to its greedy nature.

Figure 4.3 shows the partitions generated by `OGAHC` after 15000 steps for different values of ρ ($\psi = -0.003$ and $m = 2500$). For $\rho = 0$ (no virtual transitions), there are

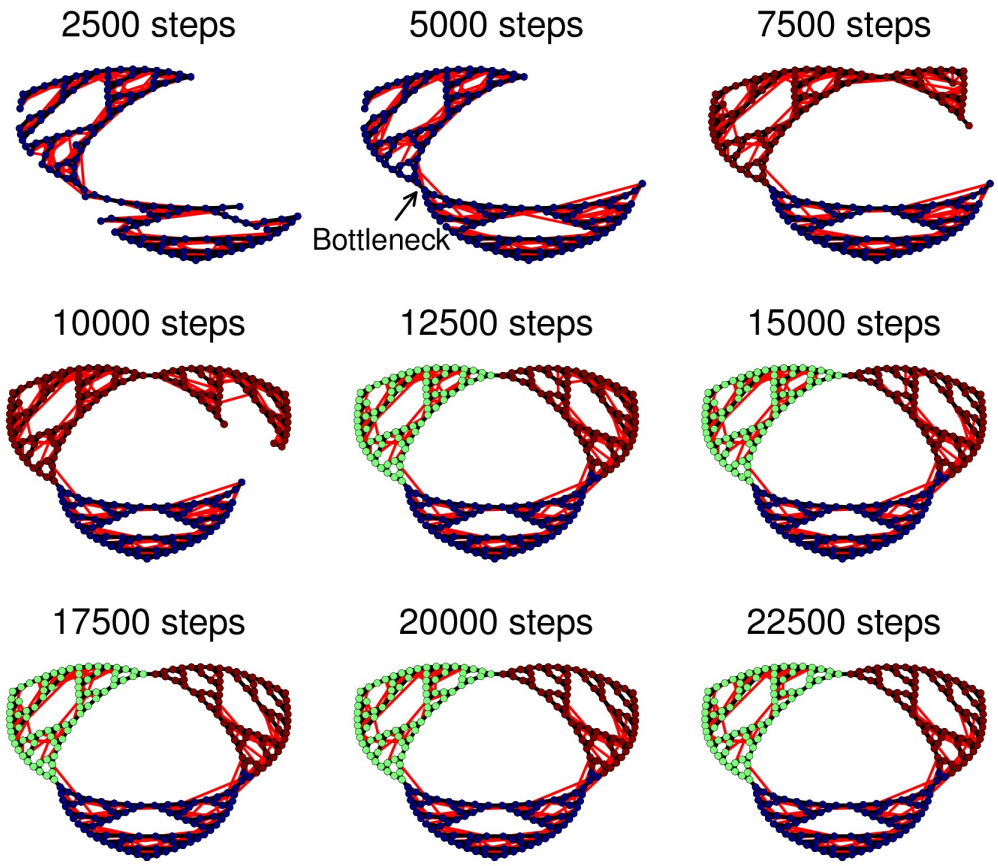


Figure 4.2 – *Illustration of* `OGAHC` *in "Towers of Hanoi" for a randomly exploring policy.* Red lines correspond to virtual edges added during graph smoothing according to the "assume dense local connectivity in the face of uncertainty" heuristic. `OGAHC` is invoked every 2500 steps and identifies bottlenecks after 7500 and 12500 steps. Note that the bottleneck identified between the "red" and the "blue" cluster is suboptimal.

many false positives which have been identified mostly already after 2500 or 5000 steps and cannot be corrected later on because of the consistency constraints. For $\rho = 3$, `OGAHC` is more conservative in identifying bottlenecks early on and does not identify bottlenecks before sufficient evidence is present. Thus, the resulting partition is close-to-optimal. For $\rho = 6$, the algorithm is overly conservative and adds many virtual transitions such that even after 15000 steps, the algorithm has not decided on any bottleneck position. Thus, the choice of ρ is important for the performance `OGAHC` and a good trade-off between too liberal and too conservative is required.

4.2.4 Skill Prototype Generation

Given a partition $\mathcal{P}$ obtained using `OGAHC` or any other graph clustering approach, one skill prototype is generated for reaching each identified bottleneck area. We set the bottleneck area of two clusters A and B to the boundary b_v of the corresponding

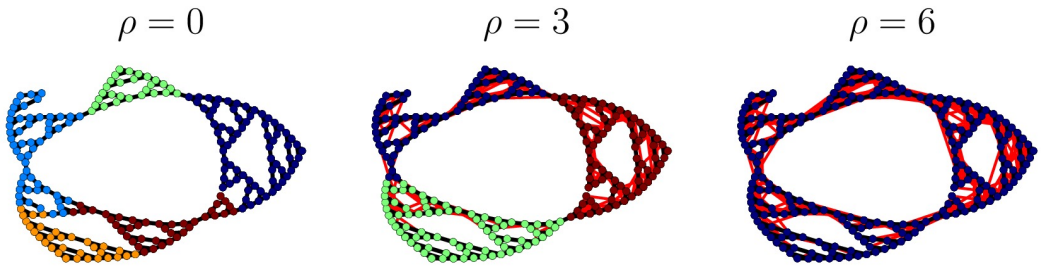


Figure 4.3 – *Illustration of graph smoothing in "Towers of Hanoi".* Shown are results after 15000 steps of random exploration. Red lines correspond to virtual edges added during graph smoothing according to the "assume dense local connectivity in the face of uncertainty" heuristic. Without any graph smoothing ($\rho = 0$), many spurious bottlenecks are identified. With strong graph smoothing ($\rho = 6$), no bottlenecks are identified at all. Medium graph smoothing ($\rho = 3$) results in a close-to-optimal clustering of the graph.

subgraphs of G. The skill prototype $\Psi_{AB} = (I_{AB}, \beta_{AB}, R_{AB})$ that is generated for the bottleneck between A and B is then defined as follows:

$$\begin{aligned} I_{AB} &= (A^* \cup B^*) \setminus b_V(A,B) \\ \beta_{AB}(s) &= 0 \text{ if } s \in I_{AB} \text{ else } 1 \\ R_{AB}((s,a,r,s')) &= -1 \text{ if } s' \in (A^* \cup B^*) \text{ else } r_p - 1, \end{aligned}$$

where $A^* = \{v \in V \mid \exists v' \in A : (v,v') \in E\}$ is the set of all nodes connected to a node in A. In other words: the skill prototype's initiation set I_{AB}, where the skill can be invoked, consists of all states $s \in S$ which are connected via an edge to a node $v \in A \cup B$ and are not bottleneck states $b_V(A,B)$ between A and B. The skill prototype terminates if it leaves its initiation set, and obtains a reward of -1 per time step. If the skill reaches a state which is not connected to any node $v \in A \cup B$, the additional "penalty" $r_p \ll 0$ is given for failing to fulfill a skill's objective. Thus, the optimal policy corresponds to reaching the bottleneck area as fast as possible while avoiding to leave the clusters A and B. Figure 4.4 illustrates a skill prototype derived from a partition of the transition graph of the "Towers of Hanoi" game.

Further skill prototypes Ψ_{As_t} are generated for reaching identified terminal states s_t from the adjacent clusters A:

$$\begin{aligned} I_{As_t} &= A^* \setminus \{s_t\} \\ \beta_{As_t}(s) &= 0 \text{ if } s \in I_{As_t} \text{ else } 1 \\ R_{As_t}((s,a,r,s')) &= -1 \text{ if } s' \in A^* \text{ else } r_p - 1, \end{aligned}$$

Note that the skill prototypes may change over time when A or B change because of re-clustering; however, a skill and its corresponding bottleneck area can never disappear in `OGAHC` because of the constraints that prevent merging clusters with nodes that have been in different clusters in previous partitions.

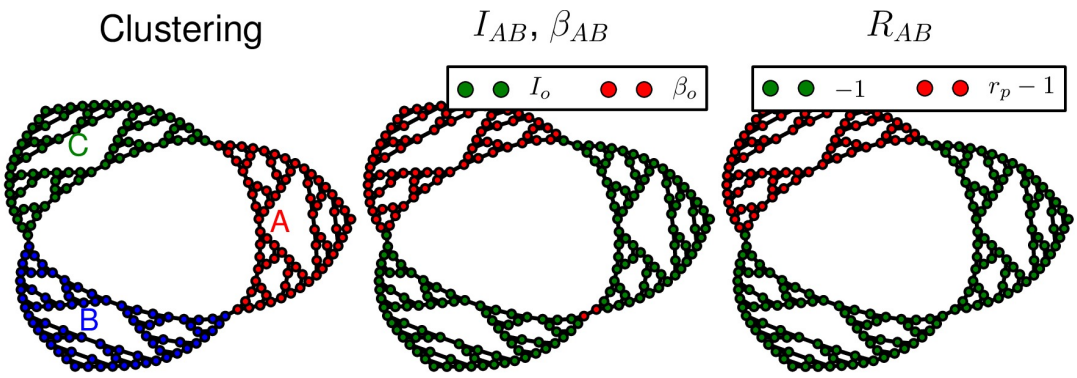


Figure 4.4 – *Illustration of a skill prototype for a given partition of the transition graph in "Towers of Hanoi".* The left graph shows a partition of the transition graph into three clusters, the middle plot the initiation set and termination condition of the resulting skill prototype Ψ_{AB} (with green nodes being the nodes in which the skill can be invoked and does not terminate and red nodes vice versa), and the right plot the corresponding pseudo reward function. Please note that the optimal policy for this skill would thus correspond to reaching the two bottleneck nodes between *A* and *B* as fast as possible.

4.3 RESULTS

In this section, we present an empirical evaluation of the proposed skill discovery architecture presented in Figure 4.1. First, we evaluate subcomponents of the architecture separately: in Section 4.3.1, we compare the quality of three different (off-line) graph clustering methods for given graphs with respect to their ability to identify bottlenecks. In Section 4.3.2, we investigate how critical the choice of the bottleneck parameter ψ is. Section 4.3.3 considers the effect of graph smoothing onto the quality of the resulting partition. Thereupon, in Section 4.3.4, we investigate the convergence properties of the on-policy and off-policy edge weights. In Section 4.3.5, we evaluate transition graph generation and graph clustering jointly with regard to the quality of the obtained partitions for different exploration strategies of the agent and different domain stochasticity. Finally, in Section 4.3.6, we evaluate the architecture as a whole in a multi-task RL problem and compare incremental and non-incremental graph clustering for skill discovery.

4.3.1 Graph Clustering

In this section, we compare three different graph clustering algorithms empirically with respect to their ability to identify bottlenecks of randomly generated graphs. We compare normalized spectral clustering (Shi and Malik, 2000), PCCA$^+$ (Deuflhard and Weber, 2005), and agglomerative hierarchical clustering. See Section 2.4 for a discussion of these algorithms.

The evaluation is based on randomly generated euclidean graphs where all graph nodes are embedded in $[0,1]^2$. The random euclidean graphs have been created as

follows: (1) n_{node} graph nodes are sampled uniform randomly from $[0,1]^2$, (2) all pairs of nodes are sorted according to their euclidean distance, and (3) the $n_{\text{node}} \cdot n_{\text{degree}}$ pairs with minimal distance are connected via an edge, such that the average degree of a node becomes n_{degree}. If the resulting graph consists of more than one connected component, the process is repeated until a connected graph is obtained. Figure 4.5 shows an example of a graph generated via this procedure for $n_{\text{node}} = 200$ and $n_{\text{degree}} = 4$.

The objective for graph clustering is to identify a partition of the graph into n_{cluster} clusters such that the NCut of the partition becomes minimal, i.e., the clusters are separated by bottlenecks of maximal strength. For a given partition $\mathcal{P}$, we define the mean pairwise normalized cut of all clusters as $\overline{\text{NCut}} = |\text{CC}\{\mathcal{P}\}|^{-1} \sum_{p_i,p_j \in \text{CC}\{\mathcal{P}\}} \text{NCut}(p_i, p_j)$, where $\text{CC}\{\mathcal{P}\}$ denotes the set of all connected pairs of clusters in $\mathcal{P}$. The three graph clustering algorithms have been applied to the random graphs and $\overline{\text{NCut}}$ has been computed for the resulting partitions. Note that PCCA$^+$ required sometimes additional post-processing of the partitions since some of the partitions contained clusters which consisted of more than one connected component. In this case, all but the largest connected component of a cluster have been reassigned to one of the other clusters, namely to the cluster which was connected with the maximal number of edges to the component.

Figure 4.6 shows the median $\overline{\text{NCut}}$ and the runtime[1] of the clustering algorithms for different values of n_{cluster}, n_{node}, and n_{degree}. Varying the number of clusters n_{cluster} shows that agglomerative clustering results in the best partitions for small n_{cluster}, while agglomerative and spectral clustering are on a par for larger n_{cluster}. PCCA$^+$ is slightly worse for all values of n_{cluster}. The runtime of PCCA$^+$ and spectral clustering increases with n_{cluster} while it decreases for agglomerative clustering. This is because for larger n_{cluster}, the dendrogram (see Section 2.4.3.3) need not be fully constructed since the process can be stopped once a forest of n_{cluster} trees was constructed. Thus, the dendrogram nodes close to the root, which are computationally expensive, can be skipped.

Varying the number of graph nodes n_{node} shows a similar pattern: PCCA$^+$ tends to result in the worst partitions and agglomerative clustering in the best ones, in particular for large n_{node}. Furthermore, agglomerative clustering also scales better to large graphs in terms of runtime. Thus, the better asymptotic complexity of $\mathcal{O}(n^2 \log n)$ compared to $\mathcal{O}(n^3)$for the eigendecomposition-based approaches is already relevant for moderately large graphs (cf. upper bounds on computational complexity in Section 2.4.3). Combining the results for varying n_{cluster} and n_{node} suggests that agglomerative clustering performs the better the larger the size of the average cluster $n_{\text{node}}/n_{\text{cluster}}$ is.

When varying the average node degree n_{degree}, agglomerative clustering performs worse for large n_{degree} both in terms of $\overline{\text{NCut}}$ and runtime. The latter can be explained

[1]Note that the absolute runtime is not directly comparable since spectral clustering and PCCA$^+$ are based on highly optimized code for eigenvalue decomposition in LAPACK (Anderson et al., 1999), while agglomerative hierarchical clustering is implemented using pure Python code. Thus, the runtime of agglomerative hierarchical clustering could probably be reduced by about an order of magnitude. Nevertheless, the trends of the runtime are instructive.

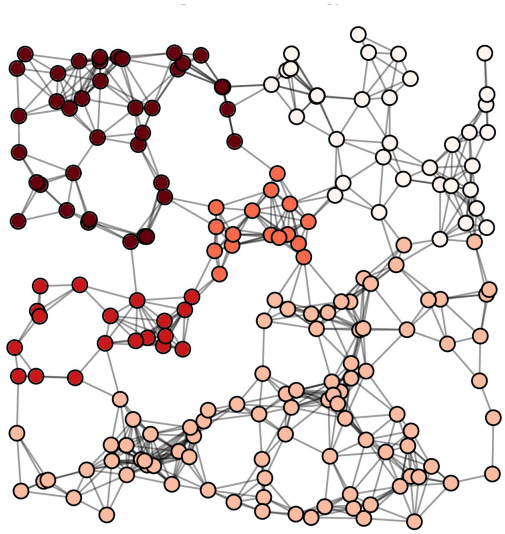


Figure 4.5 – *Example of a randomly generated euclidean graph.* The randomly generated graph consists of $n_{\text{node}} = 200$ nodes and has degree $n_{\text{degree}} = 4$, resulting in 800 edges. The node coloring corresponds to a close-to-optimal partition of the graph nodes into $n_{\text{cluster}} = 5$ clusters.

by the fact that increased connectivity of the graph increases the number of merge candidates, which directly influences the runtime of the algorithm.

In summary, agglomerative hierarchical clustering appears to achieve the best results for sparse graphs (graphs with low connectivity) and large clusters. For situations where the graph is densely connected, agglomerative clustering is less recommendable since its runtime increases considerably and the quality of the partition is also slightly worse than for the other clustering algorithms. Nevertheless, since sparsely connected graphs are typical for MDPs, agglomerative clustering should be well suited in general for identifying bottlenecks in MDPs.

4.3.2 Bottleneck Criterion

In this section, we investigate how critical the choice of the bottleneck parameter ψ is, i.e., how it affects the number of clusters of the partition. For this, a graph consisting of $3^6 = 729$ nodes has been generated (see left diagram of Figure 4.7). This graph consists of 3×3 subgraphs that all have cardinality $3^4 = 81$. Each of these subgraphs is connected via one edge to its 4 neighbor subgraphs. Moreover, each subgraph is internally structured according to the same principle, i.e., consisting of 3×3 subgraphs which are mutually connected using the 4-neighborhood. Because of the hierarchical structure of the graph, partitions consisting of 3^k clusters with $k \in [0, 2, 4, 6]$ should be preferred.

The right diagram in Figure 4.7 shows a plot of ψ versus the number of clusters obtained via non-incremental agglomerative clustering. The minimal value $\psi = -1$ corresponds to a situation where every node forms a separate cluster since a bottleneck

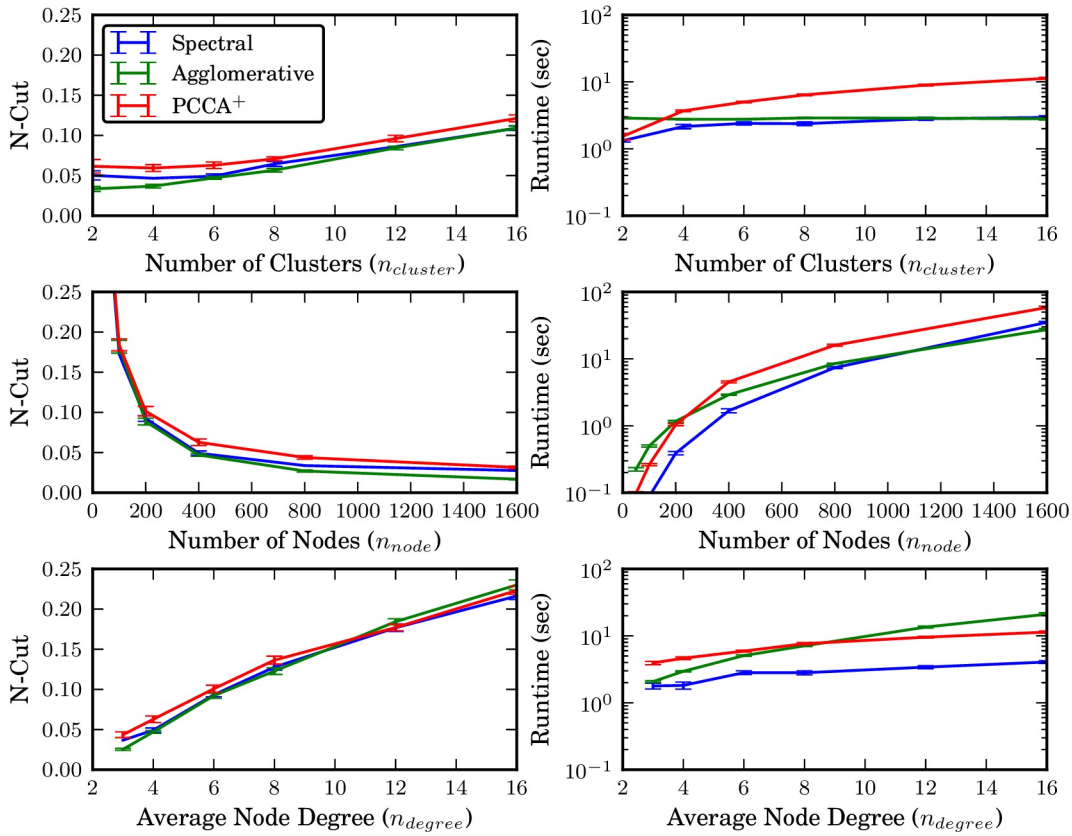


Figure 4.6 – *Comparison of graph clustering algorithms.* The graphs show results on random euclidean graphs with respect to the mean normalized cut of the obtained partitions and the runtime. Default values are $n_{\text{cluster}} = 6$, $n_{\text{node}} = 400$, and $n_{\text{degree}} = 4$. Smaller $\overline{NCut}$ corresponds to better partitions. Shown are median and standard error for 20 repetitions.

is detected for any pair of subgraphs since the linkage is lower bounded by -1. On the other hand, values of ψ close to 0 result in a partition consisting of one cluster that contains all graph nodes. This is because there exists no pair of subgraphs such that the probability of transitioning from one subgraph to the other becomes so small that it would actually be considered as a bottleneck for values of ψ very close to 0. For intermediate values of ψ, partitions with a cardinality of approximately 3^2 or 3^4 are chosen typically. This shows that the specific choice of ψ is not very critical since the resulting partitions remain very similar for a large range of values for ψ. Note that the differences between the individual runs are due to different random permutations of graph node indices (the clustering breaks ties between merge candidates with the same linkage based on node indices).

4.3.3 Graph Smoothing

This section presents an evaluation of the effect of graph smoothing onto the number of false positives (FPs) and false negatives (FNs) in bottleneck detection using *non-*

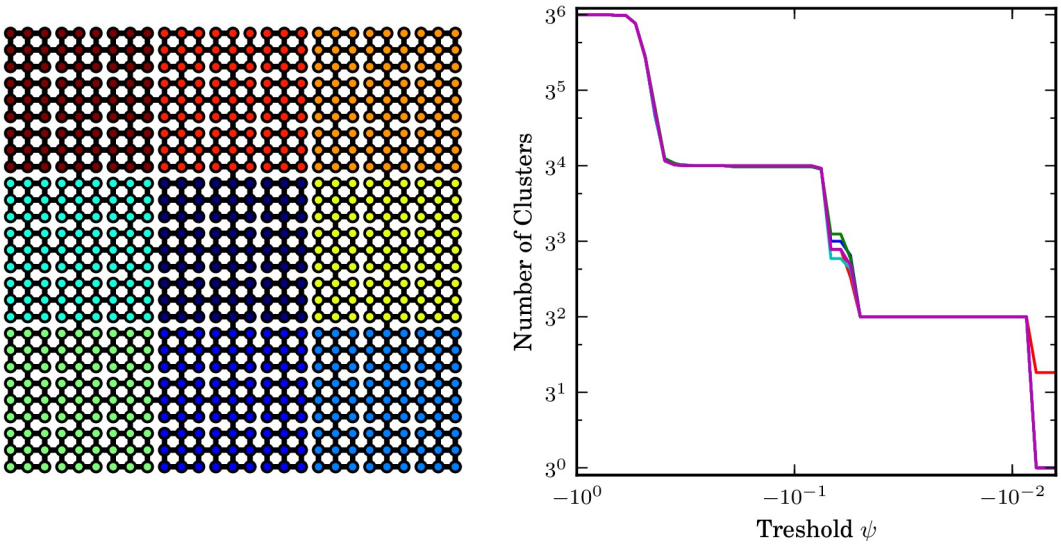


Figure 4.7 – *Analysis of granularity of partitions*. The graph depicts the effect of ψ onto the granularity of the partition: for a graph with a simple hierarchical structure (left diagram), the right diagram shows how the partition's granularity, i.e., the number of clusters of the partition, changes with ψ. The partitions have been generated using agglomerative hierarchical clustering (see Algorithm 2.4). Shown are results for 5 random permutations of graph node indices.

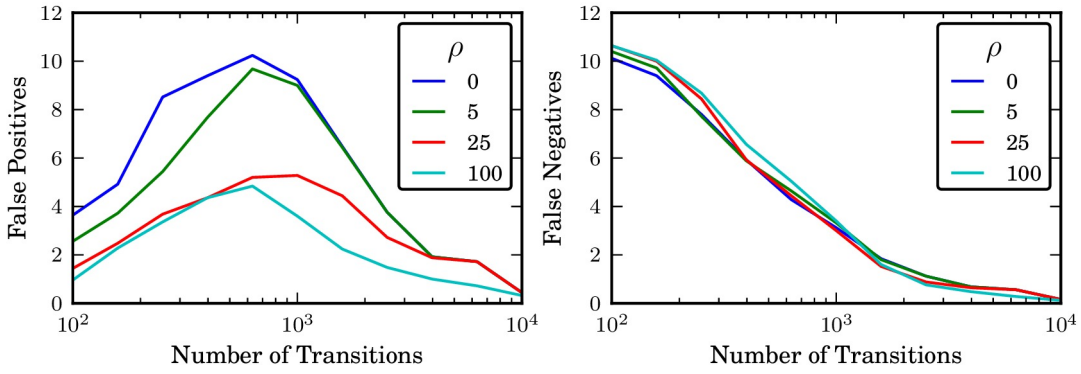


Figure 4.8 – *Effect of graph smoothing onto bottleneck detection quality*. The graph depicts the false positives and false negatives in bottleneck detection (compared to ground-truth) for non-incremental graph clustering for different values of ρ and different number of transitions used for generating the transition graph. Shown is mean over 25 repetitions.

incremental agglomerative clustering. The evaluation was performed on a $9 \times 9 = 81$ nodes subgraph of the graph used in Section 4.3.2. However, in contrast to Section 4.3.2, the true graph $\mathbb{G}$ is not known to the agent but the agent can only approximate it in the form of a sample transition graph G by performing a random walk on the graph and using the methods from Section 4.2.1. Because of the differences between $\mathbb{G}$ and G, smoothing the sample transition graph becomes important since it allows handling the trade-off between FPs and FNs in bottleneck detection.

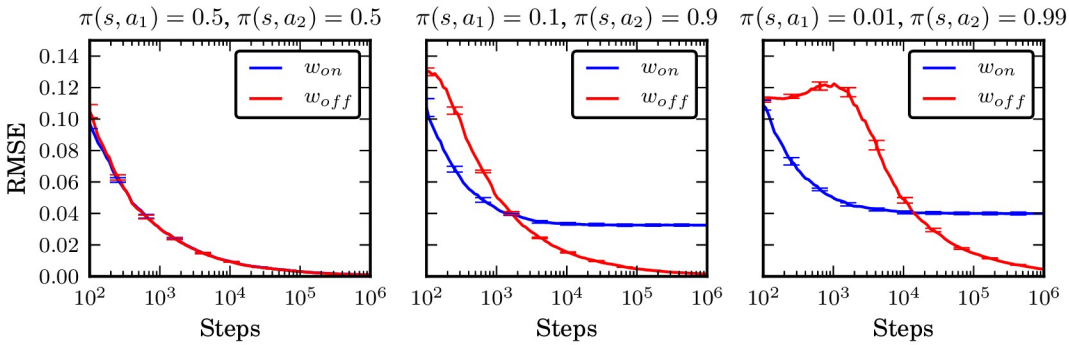


Figure 4.9 – *Empirical comparison of on- and off-policy edge weights*. The graphs show the root mean squared error (RMSE) of the estimated edge weights after different number of sampling steps on a random graph under different sampling policies. Shown are mean and standard error of the mean for 10 independent runs.

Figure 4.8 shows the number of FPs and FNs in detection of bottlenecks b^*, compared to the bottleneck edges detected when clustering $\mathbb{G}$. Different values for the degree of smoothing $\rho \in \{0, 5, 25, 100\}$ and different number of transitions m used for constructing the sample transition graph are shown. Smoothing was based on the geodesic distance of graph nodes. In this setup, there is a total of 12 bottlenecks for $\psi = -0.25$. For $m = 10^4$, the clustering identified these (and only these) bottlenecks nearly perfectly: FPs and FNs are very close to 0 on average. If clustering is performed earlier, e.g., after $m \leq 10^3$ steps, some actual bottlenecks are missed ($FN \gg 0$) and some parts are wrongly classified as bottlenecks ($FP \gg 0$). Being more conservative (larger ρ) reduces the FPs while surprisingly, it does have only a small negative effect on the FNs. Thus, in this domain, graph smoothing with $\rho \geq 25$ is strongly recommended. Note that for incremental clustering approaches, FPs are far more critical than FNs since FNs can be compensated in subsequent clusterings while FPs remain because of the consistency constraints.

4.3.4 Edge Weights

This section compares empirically the on-policy edge weights $w_{on}((v,v')) = \frac{1}{n_v}\sum_a n^a_{vv'}$ to the off-policy edge weights $w_{off}((v,v')) = 1/|A|\sum_a (n^a_{vv'}/n^a_v)$, cf. Section 4.2.1. For this comparison, we have created a stochastic MDP with 10 states and 2 actions with $P^a_{ss'}$ chosen randomly and densely, i.e., $P^a_{ss'} > 0\ \forall s, s', a$. Figure 4.9 shows the root mean squared error (RMSE) of the estimated edge weights under different sampling policies, which is computed as $\mathrm{RMSE}(w) = \sqrt{1/|V|^2 \sum_{v,v'}(w_{vv'} - P_{vv'})^2}$, where we use the shorthand notation $P_{vv'} = 1/|A|\sum_a P^a_{vv'}$.

The figure shows that for uniform sampling policies with $\pi(s,a_1) = \pi(s,a_2) = 0.5$, both w_{on} and w_{off} perform identical and converge to zero RMSE. However, for non-uniform sampling policies, only w_{off} converges to zero RMSE while w_{on} converges to other weights, namely to $w_{on}((v,v')) = \sum_a \pi(v,a) P^a_{vv'}$. Thus, the on-policy weights

depend actually on the sampling policy π. Note that RL agents using, e.g., ε-greedy exploration behave typically like the non-uniform sampling policies. Thus, using the off-policy edge weights is crucial.

4.3.5 Cluster Accordance Analysis

This section presents an empirical evaluation of the quality of the partitions generated by different clustering approaches and linkage criteria in randomly generated MDPs. We investigate the effect of domain stochasticity and different exploration behavior of the agent onto the resulting partitions. In contrast to the experiment presented in Section 4.3.3, we consider in this section also the incremental graph clustering method `OGAHC`.

4.3.5.1 Experimental Setup

We have created 50 random MDPs as follows: the state space of all MDPs has been set to a two dimensional grid of 400 states ($\mathcal{S} = \{0,1,\ldots,19\}^2$) and the action space to contain four discrete actions ($\mathcal{A} = \{(-1,0),(1,0),(0,-1),(0,1)\}$). For any state transition, a reward of -1 is given, and an episode starts always in $s_0 = (0,0)$ and terminates in $s_t = (19,19)$.

The MDP's state transition probability $P^a_{ss'}$ depends on an implicit connectivity graph $\mathbb{G}_c$ (see Figure 4.10 for three random examples). This graph is created based on a partitioning (the later ground-truth partition) of the states that is generated by drawing 7 states (the "centers") uniform randomly from $\mathcal{S}$ and assigning each state to the cluster of its closest center (breaking ties randomly). A graph edge is added between any pair of states that are neighbors in the 4-neighborhood and in the same cluster. One additional edge per cluster-pair (p_i, p_j) is added between a randomly drawn pair of neighbors where one is in cluster p_i and the other in p_j. For any state-action pair s,a, let the deterministic successor state be $d(s,a) = s+a$ if $(s,s+a)$ is an edge in $\mathbb{G}_c$ and else $d(s,a) = s$. We set $P^a_{ss'} = 1-(8/9)\chi$ for $s' = d(s,a)$ and $P^a_{ss'} = (1/9)\chi$ for any other state s' from the 9-neighborhood of s. The parameter χ determines the MDP's stochasticity. Note that for $\chi = 0$, the connectivity of $\mathbb{G}_c$ controls the connectivity of the sample transition graph, while for $\chi > 0$, it influences only its edge weights.

For each MDP, an optimal policy has been computed off-line and a trajectory consisting of n transitions has been sampled by following the optimal policy ε-greedily. We have set ε initially to 1 and then decayed it after each step by the factor $1-\varepsilon_{decay}$. Based on the sampled transitions, a partition of the states is determined and compared to the ground truth partition using the accordance ratio acc_{GT} (see Section 3.4.1). Statistical hypothesis testing has been conducted using Student's independent two-samples t-test.

4.3.5.2 Linkage Criteria

In a first experiment, we compare different linkage criteria in a non-incremental clustering setting for $\chi = 0$. To analyze the effect of non-uniform exploration of the agent, we have varied ε_{decay}. Large values of ε_{decay} correspond to an agent which starts to

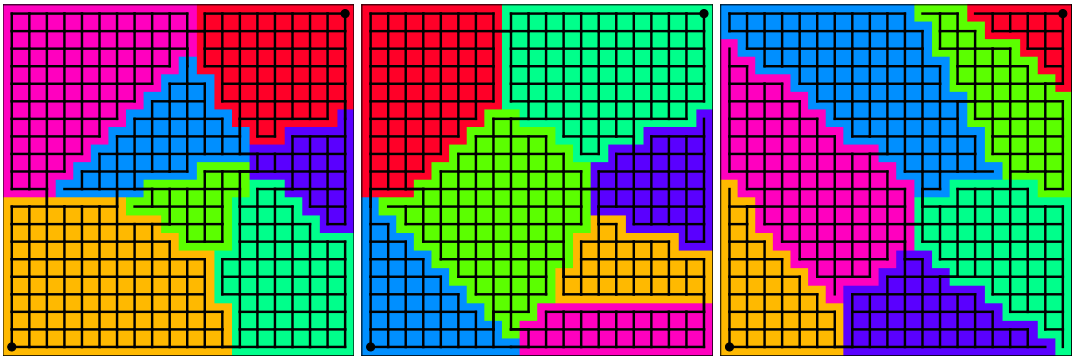
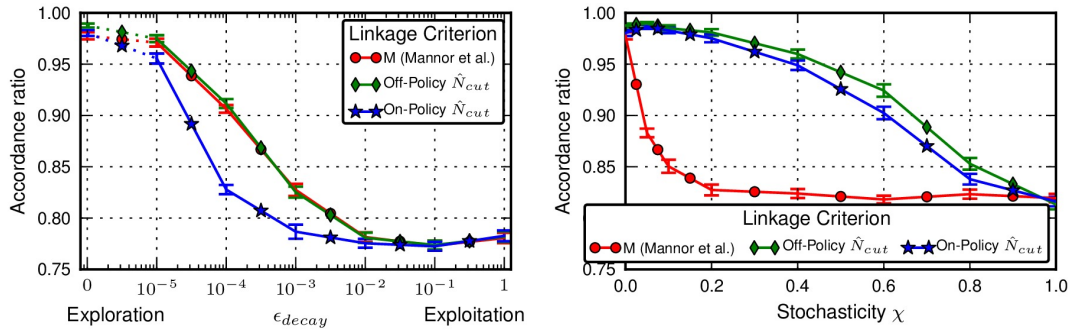


Figure 4.10 – *Examples of random graphs governing the MDP's state connectivity.* Shown are three of the fifty connectivity graphs $\mathbb{G}_c$ of the randomly generated MDPs.



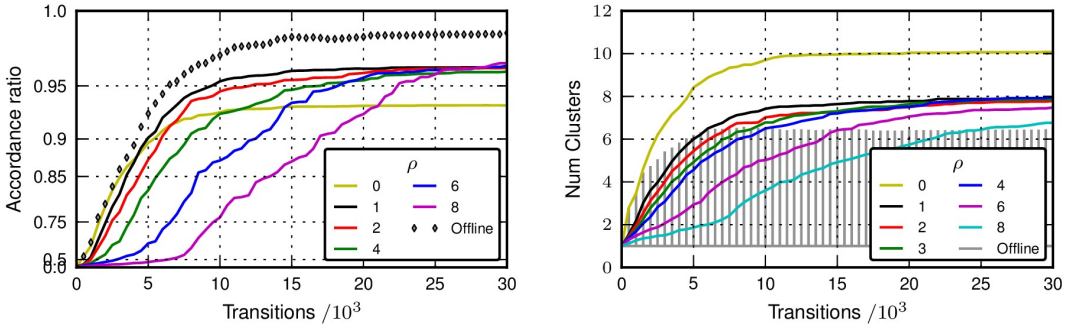
(a) Different degrees of exploration

(b) Different degrees of domain stochasticity

Figure 4.11 – *Effect of domain stochasticity and exploration on graph clustering.* The graph depicts a comparison of different linkage criteria for different degrees of exploration and different stochasticity of the environment. Shown are mean and standard error of the mean of the accordance ratio acc_{GT} over the 50 random MDPs.

exploit more early and as a consequence, obtains a more biased estimate of the transition graph. Figure 4.11a shows a comparison of the partitions obtained after $n = 25000$ transitions for the M-linkage (see Section 4.2.2) and for the $\hat{N}_{cut}$-linkage with both on-policy and off-policy edge weights as discussed in Section 4.2.1. The main result shown in the figure is that the $\hat{N}_{cut}$ linkage with on-policy weights obtains significantly worse partitions than the other two linkages for intermediate values of ε_{decay} ($p < 0.008$ for $\varepsilon_{decay} \in \{10^{-5}, 10^{-4}, 10^{-3}\}$). This can be attributed to the fact that the on-policy $\hat{N}_{cut}$ linkage bases its partitions not solely on the environment's transition probabilities but also on the agent's action selection which is undesirable if the agent selects actions non-uniformly.

In a second experiment ($n = 25000$, $\varepsilon_{decay} = 0$), we have varied the stochasticity χ of the environment. Figure 4.11b shows that the M-linkage obtains significantly worse results than the off-policy $\hat{N}_{cut}$ linkage if the environment gets slightly non-deterministic ($p < 0.0001$ for $0.05 \leq \chi \leq 0.8$). This is due to the use of uniform edge weights in the M-linkage; these weights do not allow differentiating between



(a) Accordance ratio acc_{GT} **(b)** Resulting number of clusters n_{cluster}

Figure 4.12 – *Comparison of graph clustering methods.* Shown is a comparison of the incremental graph clustering method `OGAHC` (for different values of ρ) with off-line, non-incremental clustering. The plots visualize (a) the mean of the accordance ratio acc_{GT} and (b) the mean of n_{cluster} over the 50 random MDPs.

probable and less probable transitions, which is apparently important for skill discovery in non-deterministic environments. In summary, the results show that the off-policy $\hat{N}_{cut}$-linkage is the most robust linkage criterion; the following experiments have been conducted with this linkage accordingly.

4.3.5.3 Incremental Graph-Clustering

In this subsection, we compare the incremental graph clustering method `OGAHC` with its off-line counterpart (essentially the approach proposed by Mannor et al. (2004), see Section 3.3.4). We have used the same 50 random MDPs as before and set $n = 30000$, $\varepsilon_{decay} = 0$, $\chi = 0$, and $\psi = -0.0375$. For `OGAHC`, the clustering has been updated every 500 steps and different choices of ρ have been evaluated. The off-line clustering has been performed after $m \in \{500, 1000, \ldots, 30000\}$ transitions, taking all m transitions that have been acquired so far into account at once.

Figure 4.12a shows how the accordance ratio of the partitions identified by incremental graph clustering changes over time. It can be seen that smaller values of ρ achieve higher accordance ratios in the early phase since they tend to identify clusters more early (see Figure 4.12b). However, these clusters are potentially suboptimal since small ρ has the risk of detecting several false positive bottlenecks (see Section 4.3.3). Accordingly, for $\rho = 0$ the accordance ratio plateaus on a lower level than for larger values of ρ since these false positives cannot be corrected later on because of the consistency constraints. This can be seen also in Figure 4.12b: for $\rho = 0$, the number of clusters is on average 10 in the long run while there are actually only 7 clusters. Thus, there are 3 false positives on average. Larger values of ρ perform worse initially since they are conservative and thus miss many potential bottlenecks (many false negatives). In the long run ($m = 30000$), however, the quality of the partition becomes better than for $\rho = 0$ since there are less false positives.

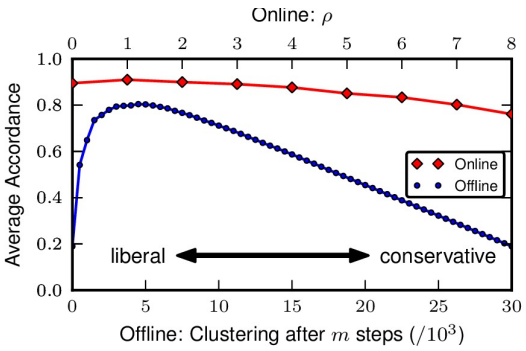


Figure 4.13 – *Average accordance of Incremental and Non-Incremental Clustering.* Shown are the accordances during the 30000 exploration steps of Figure 4.12a. OGAHC is denoted by "online". Note that the relative scale of the two horizontal axes is arbitrary.

Comparing OGAHC and off-line, non-incremental clustering, one can see that for any number of transitions, OGAHC tends to be slightly worse than the off-line clustering obtained at that point in time. This is due to the enforced consistency with prior clusterings and to the influence of ρ. However, the clusterings of OGAHC can be refined over time while the clusterings of the off-line approach are fixed. Thus, the performance of the two method should not only be compared at the specific point in time where the off-line clustering is performed. Figure 4.13 shows the accordance ratio averaged over the 30000 steps[1] for different values of ρ and m. For both small and large values of m, the average accordance of the off-line clustering is low. For small m, the average accordance is low because the clustering is based on only few transitions and cannot be improved later on; for large values of m, the average accordance is low because there is no clustering at all for a long initial period. Intermediate values ($m \approx 4500$) obtain a higher average accordance of approximately 0.81. In contrast, the on-line clustering with OGAHC depends less on the specific choice of ρ. The optimal value for ρ is 1, which results in an average accordance of approximately 0.91; however, for any value $0 \leq \rho \leq 6$, OGAHC achieves a significantly higher average accordance than the off-line clustering for $m = 4500$ ($p < 0.019$). Thus, even without fine-tuning ρ, the on-line clustering can outperform the off-line clustering with optimally chosen m with regard to the average quality of the clustering.

4.3.6 Multi-task Learning

In this section, we evaluate the utility of OGAHC within a Hierarchical RL agent employing the full architecture of Figure 4.1 in a multi-task learning scenario. The scenario is a 23×23 maze world consisting of four rooms with four "special" states (see Figure 4.14). In each time step, the agent obtains a reward of -1. The agent has to learn to solve

[1] The average accordance is computed as $\overline{\text{acc}_{gt}} = 60^{-1} \sum_{i=1}^{60} \text{acc}_{gt}\{\mathcal{P}_{500i}\}$, where $\mathcal{P}_j$ denotes the partition obtained after j steps. Note that this formula exploits that graph clustering is performed every 500 steps. For off-line graph clustering and $j < m$, graph clustering has not been performed yet and the partition is set to a single cluster containing all nodes.

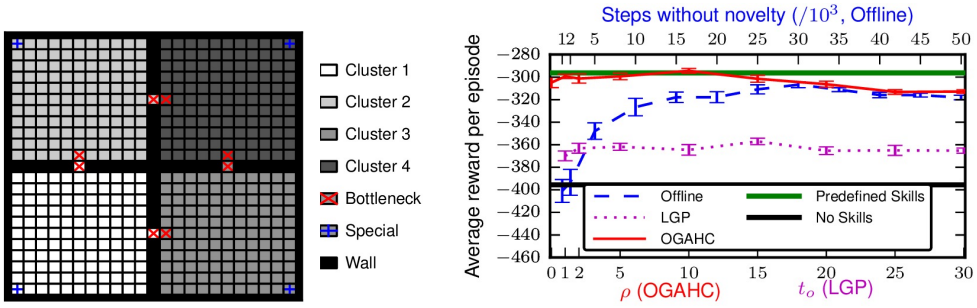


Figure 4.14 – *OGAHC in the Multi-Task Maze*. Left: Structure, ground-truth partitioning, and bottlenecks of the multi-task maze world. Right: Reward per episode averaged over the first 1000 episodes (R_{avg}^{1000}) for different skill discovery strategies. The parameters ρ and t_o are varied over the same value range (bottom x-axis). Note that the relative scale of the horizontal axes is arbitrary. Shown are mean and standard error of the mean for 15 repetitions. See Figure 4.15 for detailed learning curves.

12 different tasks in this environment; each task is defined by a pair (s_0, s_t) of special states where s_0 is the start state and s_t is the goal state of the task. In each episode, a task is chosen at random and the specific task is communicated to the agent as a state space dimension.[1] This dimension is used by the agent for policy learning but is not taken into account during skill discovery and skill learning. The acquired skills can thus be transferred between tasks and may give a useful exploration bias for the agent. We have chosen this domain for our evaluation since one can define a ground-truth partition easily in this domain by identifying the doorways as bottlenecks and the domain allows studying the potential of the discovered skills for transferring procedural knowledge between different tasks.

The agent uses a 3-layer hierarchy (see Section 2.3.4): the lowest layer consists of the primitive actions, the middle layer of the acquired skills and a special option discussed below. On the upper layer, the agent may choose among these skills but not choose a primitive action directly, i.e., the action space is abstracted not augmented (see Section 2.3.4). 1-step intra-option Q-Learning (cf. Section 2.3.4.3) is used for skill learning; no experience replay is conducted to avoid intermixing the contributions of skill discovery and experience replay (see Jong et al. (2008) for a discussion). A special low-level option prototype is provided to the agent: this option can be invoked in any state, terminates in any state with probability $\beta = 0.01$, and uses the external reward signal. This option allows the agent thus to learn a monolithic global policy. The reasons for this option are twofold: on the one hand, it guarantees that the agent can learn a global optimal policy eventually despite the abstraction of the action space. Moreover, it makes the agent less susceptible to the broken exploration symmetry that is caused by temporal abstraction than pure augmentation of the action space (cf. Section 3.2.1 and Figure 3.3).

[1] Note that we thus inform the agent explicitly about the current task, which makes recognizing the same task later on trivial. Perkins and Precup (1999) presented a Bayesian framework in which the agent need not be informed about the current task but can identify it on its own.

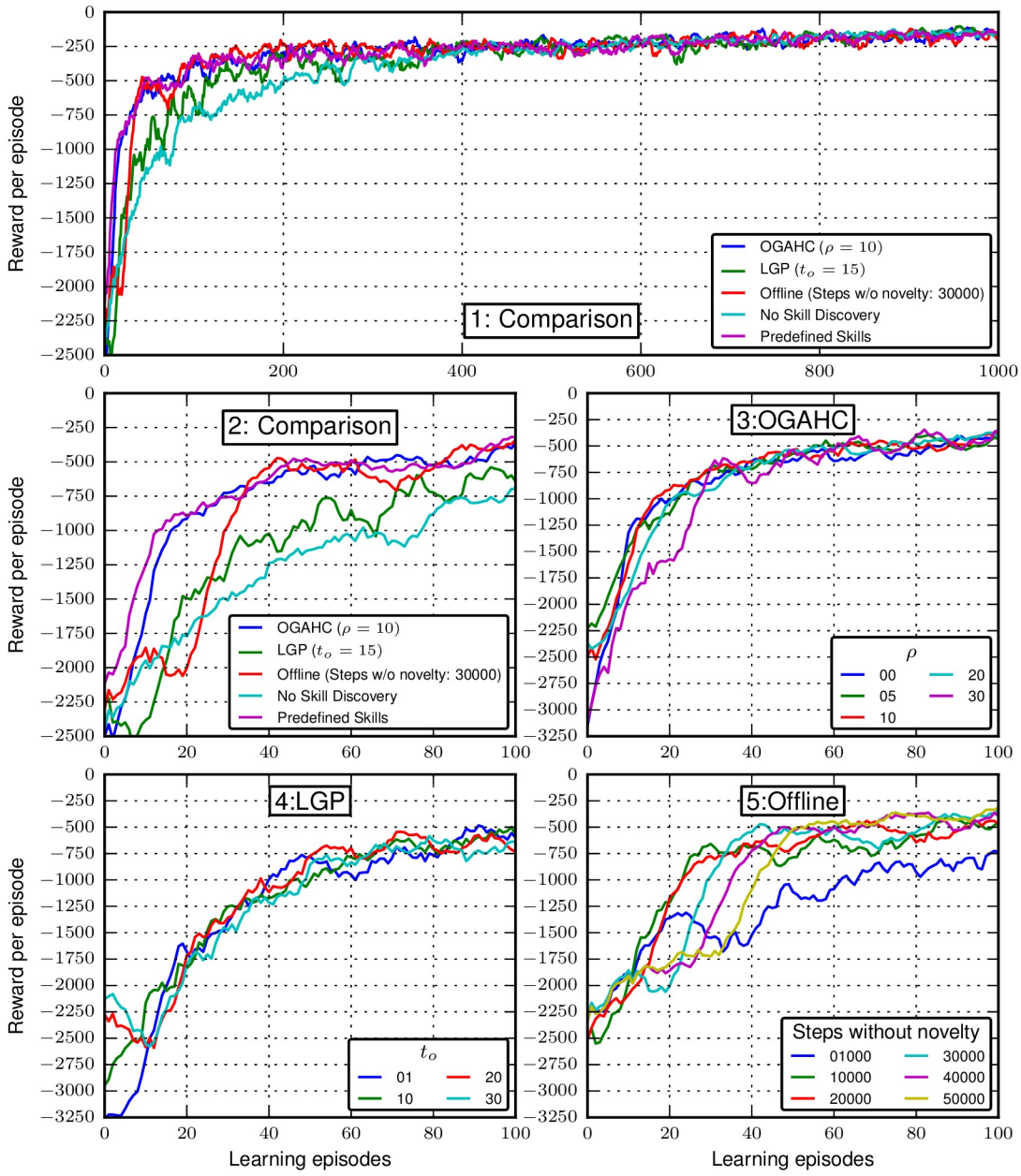


Figure 4.15 – Plot 1 shows learning curves during the first 1000 episodes for the different methods with optimal values for ρ, t_o and "steps without novelty". Plot 2 shows the same comparison zoomed-in to the first 100 episodes. Plot 3, 4, and 5 show the learning curves of OGAHC, LGP, and Offline Clustering for different values of ρ, t_o and "steps without novelty", respectively. Shown is the mean of $R^5_{\text{mwa}}(t)$ over 15 repetitions.

We compare OGAHC to the Off-line Clustering approach proposed by Mannor et al. (2004) and the Local Graph Partitioning (LGP) approach by Şimşek et al. (2005). As baseline, we evaluate the agent using no skill discovery (corresponding to *monolithic* in Section 3.4.2 and denoted by "no skills"), and the baseline *predefined skills* where the agent is provided with the ground-truth partition (see Figure 4.14) from the beginning. $|\psi|$ has been set to 0.0375 for all approaches. For LGP, the required hit-ratio has been set to $t_p = 0.2$, the option lag to $l_o = 20$, the window length to $h = 1000$, and the update frequency to 250. These parameters have been chosen based on preliminary experiments. The discount factor of the agent has been set to $\gamma = 0.99$, the learning rate to $\alpha = 1.0$, the value functions have been initialized optimistically to 0.0, and the penalty for failing to fulfill a skill's objective has been set to $r_p = -100$. For OGAHC, the parameter ρ has been varied between 0 and 30, for LGP the minimum number of state observations t_o has been varied between 1 and 30, and the Off-line Clustering has been conducted after m steps in which no novel state has been encountered (where m has been varied between 5000 and 50000). Each setting has been evaluated 15 times for 1000 episodes. As performance metric, we use the average return per episode R^t_{avg} (see Section 3.4.1) since this metric allows investigating the effect of multi-task transfer onto learning speed.

The results of the empirical evaluation are depicted in Figure 4.14. The agent provided with the predefined skill prototypes obtains an average reward per episode of $R^{1000}_{\text{avg}} = -296.3 \pm 1.6$ and the agent using no skills of $R^{1000}_{\text{avg}} = -395.5 \pm 0.9$. The maximal gain of average reward per episode that can be achieved by biasing exploration using skills is thus approximately $+100$. Using LGP for skill discovery does not achieve an average reward of more than $R^{1000}_{\text{avg}} = -350$ for any choice of t_o. Closer inspection showed that for small values of t_o, the resulting partitioning was far from being optimal while for larger values of t_o, the skills have been introduced too late to provide a useful exploration bias. Using the Off-line Clustering approach, the optimal time for performing the clustering is after $m^* = 30000$ steps without observing any novel state which results in $R^{1000}_{\text{avg}} = -306.8 \pm 2.6$. Thus, performing Off-line Clustering at the right time does already realize 90% of the possible reward gain. However, for the same reasons as in Subsection 4.3.5 the choice of m is crucial: performing clustering too early ($m \ll m^*$) results in suboptimal partitions while performing it too late ($m \gg m^*$) limits the benefits of the acquired skills for learning.

OGAHC achieves $R^{1000}_{\text{avg}} > -305$ for any choice of $\rho \leq 15$ and $R^{1000}_{\text{avg}} = -294.5 \pm 2.1$ for $\rho = 10$. The approach allows thus acquiring more than 90% of the possible reward gain (and thus more than obtained by Off-line Clustering for any choice of m) for a broad range of values for ρ, making the specific choice of ρ less important. For the optimal choice $\rho^* = 10$, OGAHC achieves an average reward that is on a par with what can be obtained with the predefined skill prototypes. This can be explained by the observation that for this choice of ρ, OGAHC discovers skills that are close to the optimal ones very early during learning (typically during the first 3 episodes). Since skill learning takes several episodes, the predefined skill prototypes offer the agent an efficient exploration bias starting after approximately 5 episodes; thus, OGAHC can be

on a par with the predefined skill prototypes. In summary, the results show that using OGAHC for skill discovery allows identifying reusable skills at an early stage of learning without requiring to fine-tune the parameter ρ.

4.4 DISCUSSION

This chapter has proposed the novel skill discovery approach OGAHC. OGAHC generates a transition graph, which captures the dynamics of a discrete domain, based on experience and discovers reusable and versatile skills by identifying bottlenecks of this graph using agglomerative hierarchical clustering. OGAHC is fully incremental and allows identifying bottlenecks of a domain robustly. The four main features of OGAHC that set it apart from related approaches are the following:

1. Use maximum likelihood estimates for the sample transition graph's weights and correct for the bias of the sampling policy by means of importance sampling (see Section 4.2.1). These "off-policy" weights are robust with regard to stochastic domains and the agent's exploration behavior.
2. Use the $\hat{N}_{cut}$ linkage for defining bottlenecks since it has a straightforward connection to random walks on a graph and it's parameter ψ has a simple probabilistic interpretation (see Section 4.2.2).
3. Use agglomerative hierarchical clustering with consistency constraints for incremental, graph-based skill discovery. Consistency constraints allow performing graph clustering several times without obtaining inconsistent skill prototypes (see Section 4.2.3.1).
4. Perform graph smoothing prior to graph clustering in order to reduce the number of false positives in bottleneck detection (see Section 4.2.3.2).

Note that OGAHC is modular, i.e., every of these 4 components could be replaced by alternatives while leaving the rest unchanged. Section 4.3 has presented an empirical evaluation where each component was compared with reasonable alternatives. The following list summarizes the main empirical findings that support the choice of the specific structure of OGAHC:

- The $\hat{N}_{cut}$ linkage in combination with the off-policy weights is more robust with respect to both domain stochasticity and exploration behavior of the agent than alternatives proposed by other authors (see Section 4.3.5.2).
- The specific choice of the value of parameter ψ for the $\hat{N}_{cut}$ linkage is not very critical for the resulting partitions (see Section 4.3.2).
- Agglomerative hierarchical clustering obtains empirically good approximations of the optimal partition with a comparatively small amount of computation (see Section 4.3.1).
- Graph smoothing is crucial for reducing false positives in bottleneck detection when the agent uses a sample transition graph rather than the true transition graph (see Section 4.3.3)

- In contrast to the on-policy edge weights, the off-policy edge weights converge to the true state transition probability under biased sampling policies (see Section 4.3.4).
- Incremental graph clustering approaches like OGAHC with consistency constraints and graph smoothing can handle the trade-off between false positives and false negatives better than non-incremental "off-line" clustering approaches. Furthermore, the choice of OGAHC's parameter ρ is less critical than the choice of the single point in time for performing the off-line clustering (see Section 4.3.5.3).
- Hierarchical RL agents using OGAHC can considerably outperform other skill discovery approaches in terms of sample-efficiency in a multi-task problem. Moreover, an OGAHC-based agent with appropriate ρ performs nearly as good as an agent that is provided with the domain bottlenecks from the very beginning (see Section 4.3.6).

Thus, there is considerable empirical evidence supporting OGAHC. OGAHC exhibits 4 of the 5 desirable properties for skill discovery identified in Section 3.2.3: it is incremental (P1), identifies task-independent skills (P2), decides automatically how many skills are created (P3), and—as the empirical results show—is very sample-efficient (P5). In summary, OGAHC is a contribution that achieves the first subgoal stated in Section 1.2:

SUBGOAL S1 Develop an incremental, graph-based skill discovery approach that can identify skills at any time and allows an agent to acquire a collection of skills which increases in both size and sophistication over time.

The author would also like to note that some situations have been identified where OGAHC is not expected to work well: for instance, hierarchical agglomerative clustering is not well suited for domains with a dense transition graph since it becomes computationally expensive and yields worse partitions than other clustering approaches. Furthermore, the generation of the transition graph is based on the assumption that the dynamics $P^a_{ss'}$ of the domain do not change over time. While this is the standard "time invariance" assumption of MDPs, it is not realistic for actual, real-world scenarios, where $P^a_{ss'}$ would typically be non-stationary (cf. Section 3.2.1). One possible way to take this non-stationarity into account during transition graph generation would be to discount $n^a_{vv'}$ over time. For instance, after each transition, one could set $n^a_{vv'} \leftarrow \kappa n^a_{vv'}$, where $\kappa \in [0,1]$ is an appropriate discounting term and n^a_v needs to be renormalized accordingly. For $\kappa < 1$, the influence of old transitions, which may no longer be possible under the changing dynamics, diminishes over time and G may track the actual domain's dynamics. Using a hierarchical policy based on skills acquired based on OGAHC, on the other hand, may allow adapting more easily to (small) changes in the environment as discussed in Section 3.2.1. Please refer to Digney (1998) and Stolle and Precup (2002), where this has been shown empirically for similar approaches. Corresponding experiments for OGAHC remain to future work.

Moreover, the author acknowledges that the empirical evaluation in this chapter was performed in small, discrete "toy" problems. Nevertheless, the obtained results

are instructive and—as will be seen in the next chapters—generalize to more complex domains. By extending OGAHC to domains with continuous state spaces and action spaces in Chapter 6, a skill discovery approach that also exhibits property P4 will be obtained.

5 Learning Graph-Based Representations

"I am not discouraged, because every wrong attempt discarded is another step forward."

Thomas A. Edison

5.1 Introduction

In Chapter 4, we have proposed the incremental graph-based skill discovery approach OGAHC. We have evaluated OGAHC empirically in small and discrete domains. Since our motivation stems mainly from robotic applications, which are typically continuous, one natural question is: How can OGAHC (or any other graph-based skill discovery approach) be extended to continuous domains? We address this question in this chapter. Note that in this chapter we focus on non-incremental clustering; we combine the proposed methods with OGAHC in Chapter 6.

The main challenge when applying graph-based methods in continuous domains is how to create the sample transition graph. In small and discrete domains, constructing the sample transition graphs from experience for unknown MDPs is straightforward (cf. Section 4.2.1): one graph node is created for each observed domain state and an edge is created for any pair of states between which a direct transition has been observed. For domains with continuous state spaces, employing graph-based approaches is considered to be much more difficult (Konidaris and Barto, 2009). The reason for this is that graphs are inherently discrete structures and thus, there cannot be a one-to-one correspondence between states and graph nodes since there exists an infinite number of states in continuous domains. Thus, several states need to be aggregated into one node, i.e., $V \subsetneq \mathcal{S}$. Accordingly, one has to choose how many nodes there should be and how states should be assigned to graph nodes.

Prior work on graph-based approaches in continuous domains (see Section 5.2.1 for a more detailed discussion) has typically considered continuous domains where the

state space is a real-valued vector space, i.e., $\mathcal{S} \subseteq \mathbb{R}^{n_s}$. In this situation, states can be assigned to graph nodes based on the euclidean distance, i.e., by assigning each state to the nearest graph node. Furthermore, the number of graph nodes v_{num} is typically specified externally. The main challenge is thus to decide where the graph nodes should be placed in the euclidean space. Prior work has either discretized the domain, i.e., placed graph nodes at a regular grid over the state space (Mannor et al., 2004), or placed graph nodes at a subset of the observed states (Mahadevan and Maggioni, 2007). While the former suffers from the "curse of dimensionality" (see Section 2.2.4), the later allows exploiting situations where the effective dimensionality of the state space is smaller.

However, both approaches focus purely on covering the state space as uniformly as possible and neglect the dynamics of the environment. We argue that the construction of the sample transition graph should take the dynamics into account since a transition graph can be seen as a model of the environment. That is, typical transitions encountered in the domain should be representable by the graph. The hypothesis evaluated in this chapter is that a graph, which models the dynamics of its (continuous) environment well, will yield superior results with regard to bottleneck identification. We propose a new heuristic called `FIGE`, which allows learning transition graphs that have high likelihood under a given set of state transitions.

The outline of this chapter is as follows: In Section 5.2, we discuss how graph-based skill discovery approaches can be extended to domains with continuous state space and review related works. In Section 5.3, we define the likelihood of a graph for a given set of transitions sampled according to the domain's dynamics. Thereupon, we propose the `FIGE` heuristic for learning transition graphs of continuous environments, which is derived from the maximum graph likelihood formulation under simplifying assumptions (see Appendix A.2). In Section 5.4, we compare `FIGE` with other graph learning heuristics empirically with regard to the likelihood and the learning performance of the resulting skill discovery method. Section 5.5 presents an excursus which shows that `FIGE` is not only useful for skill discovery but also for representation learning. We summarize and discuss the results of this chapter in Section 5.6.

5.2 Graph-Based Skill Discovery in Continuous Domains

When applying graph-based methods to MDPs with continuous state space, the typical approach is to generate a (discrete) graph that captures the properties of the domain well, analyze the graph, and generalize the identified properties from the graph to the continuous domain. This is shown in Figure 5.1 for graph-based skill discovery (see Section 5.5 for a further example): transitions T from the continuous domain are sampled and a sample transition graph G is generated. This transition graph generation entails an implicit discretization. Thereupon, graph clustering as discussed in Section 4.2.3 is performed to obtain a graph partition $\mathcal{P}$. This partition is used to generate skill

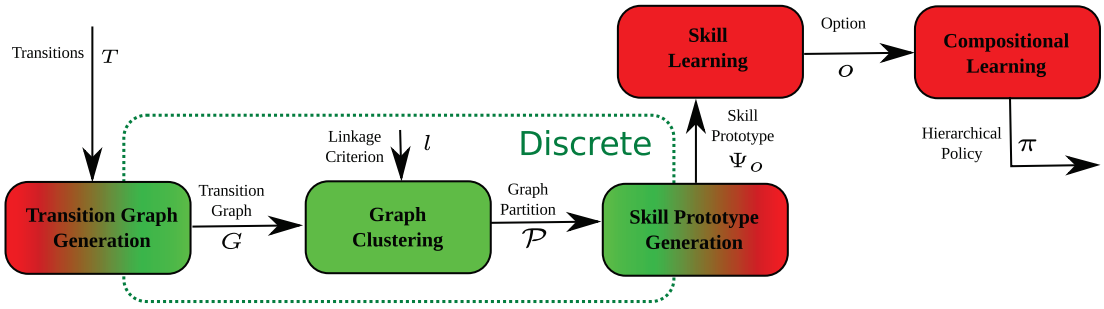


Figure 5.1 – *Data flow diagram of graph-based skill discovery for continuous domains* (cf. Figure 4.1). The diagram shows the transition between parts of the data flow that deal with the continuous domain (dyed in red) and a discretized version of it (dyed in green).

prototypes Ψ_o. However, in contrast to Section 4.2.4 these skill prototypes must not be discrete since skill learning and compositional learning take place in the original, continuous domain. Thus, skill prototype generation is the part where generalization of graph properties to the continuous domain takes place.

The main challenges are thus to generate the transition graph and to define the skill prototypes. We review prior works for transition graph generation in Section 5.2.1 and propose a new method for this in Section 5.3.2. Skill prototype generation is discussed in Section 5.3.3. Note that in this chapter, we only consider non-incremental approaches for transition graph generation and graph clustering. For an extension to incremental approaches, we refer to Chapter 6.

5.2.1 Prior Work

In this subsection, we review heuristics for generating sample transition graphs in unknown continuous MDPs, where the agent has to generate the graph based on experience it has collected during exploring its environment. In domains with continuous state spaces $S \subset \mathbb{R}^{n_s}$, there cannot be a one-to-one correspondence between states and graph nodes since there exists an infinite number of states. Thus, several states need to be aggregated into one node, i.e., $V \subsetneq S$. Accordingly, one has to choose how many nodes there should be and where in the state space these nodes should be placed. Prior work on choosing the positions of the graph nodes has mainly focused on covering the state space uniformly with nodes and neglected the domain's dynamics $P^a_{ss'}$. We summarize three heuristics that have been proposed for choosing graph nodes based on a set of transitions sampled from the domain. These heuristics are all parametrized by the parameter v_{num}, which determines the number of nodes of the generated graph.

One straightforward choice for the graph node position is to use a uniform *grid* over the state space. This approach has been used in the context of graph-based skill discovery, e.g., by Mannor et al. (2004). For an n_s-dimensional state space where the range of possible values in each dimension has been scaled to $[0,1]$, the resolution in each dimension is set to $r = \lfloor \sqrt[n_s]{v_{num}} \rfloor$ and $V = \{(i+0.5)/r \mid 0 \leq i < r\}^{n_s}$. An evident

disadvantage is that the approach will not scale to domains with many dimensions since the resolution in each dimension declines exponentially with n_s.

A second heuristic is the *on-policy sampling* heuristic (also denoted as "random subsampling" by Mahadevan and Maggioni (2007)), which samples the graph node positions uniform randomly from the set of states S' encountered during exploration. In contrast to the grid-based heuristic, this heuristic depends not directly on the state space's dimensionality n_s, but rather on the "effective" dimensionality of the manifold of feasible states. If there is redundancy in the dimensions of the state space, this effective dimensionality might be considerably lower than n_s. The heuristic is on-policy, i.e., regions of the state space that are often visited by the sampling policy are represented by more graph nodes.

The ε-*net* heuristic, also denoted as "trajectory-based subsampling" (Mahadevan and Maggioni, 2007), aims at covering the set of states encountered during exploration as uniformly as possible. It follows a greedy strategy: the first graph node v_0 is picked at random from S'. By induction, for $k \geq 1$ suppose the graph nodes $v_0, \ldots, v_{k-1}$ have already been selected and their pairwise euclidean distance is at least ε. Search for $s \in S'$ whose distance to each of the $v_0, \ldots, v_{k-1}$ is at least ε: if there are such states, pick one at random and add it to the set of graph nodes. If there is no such candidate, return the current set of graph nodes. This set corresponds to a locally maximal set of graph nodes with pairwise distance at least ε. The advantage of this approach compared to the on-policy sampling method is that the effective state space is covered more uniformly. For parameterizing the heuristic by v_{num} instead of ε, we perform binary search for a value of ε that yields a set of v_{num} graph nodes. As discussed in Section 3.3.4, Bacon and Precup (2013) have proposed to use the ε-*net* heuristic for skill discovery.

Once a finite set of graph nodes has been chosen with any of the discussed heuristics, the states of the original MDP can be associated with their closest graph nodes. By this, one can use the approach discussed in Section 4.2.1 to create graph edges and their weights by replacing an observed transition between two states by a transition between the two respective closest graph nodes. More sophisticated strategies for mapping state transitions onto node transitions would be possible: for instance, one could associate each state transition with a weighted sum of node transitions such that the weights sum to one and the weighted sum of node transitions is maximally close to the original state transition. This would, however, reduce the sparsity of the graph's connectivity and by this increase the runtime of agglomerative graph clustering approaches as discussed in Section 2.4.3.3. Note that by constructing a state transition graph, one effectively creates a discrete version of the MDP that is embedded into the continuous state space. See Section 5.3.2.1 for an illustration of the three discussed heuristics.

5.3 METHODS

While the heuristics discussed in Section 5.2.1 focus on covering the state space uniformly, they do not take the domain's dynamics into account. Thus, for many valid state transitions $s \to s'$ of the domain, there may not be any pair of graph nodes $v_1, v_2 \in V$

such that $v_1 \to v_2$ is a good representation of $s \to s'$. Accordingly, the graph may not be able to capture the domain's dynamics $P^a_{ss'}$ accurately. In this section, we propose a generative model which defines how probably a set of observed transition has been generated from a transition graph. We then propose the heuristic `FIGE` which is derived from this generative model as maximum likelihood solution under simplifying assumptions. Thereupon, we illustrate different approaches for transition graph generation and discuss how skill prototypes can be generated for a given transition graph in a continuous domain.

5.3.1 Likelihood of Transition Graph

As in Section 4.2.1, we propose to consider a transition graph as a generative model for transitions and to choose the graph such that its likelihood $L_T(G) = p(T|G)$ becomes maximal for a set of observed transitions $T = \{(s_i, a_i, s'_i)\}_{i=1}^n$. In contrast to Section 4.2.1, however, the likelihood $L_T(G)$ in a domain with continuous state space depends not solely on the graph's weights but also on the graph node positions V. We consider transitions to have been sampled from the graph using the following generative model:

1. Sample a graph node $v \in V$ uniform randomly, i.e., $p(v) = 1/|V|$.
2. Sample a state s for a given node v according to $p(s|v) = N_b \exp(-\frac{1}{b^2}\|s - v\|_2^2)$ where b controls how closely centered $p(s|v)$ is on v and N_b is a normalization constant, which only depends on b.
3. Sample an action a uniformly from the action space A, i.e., $p(a|s) = p(a) = 1/|A|$.
4. Sample the "successor node" v' according to the graph's edge weights, i.e., $p(v'|v,a) = w_{vv'}$.
5. Finally, sample the successor state s' with the same b and N_b as before according to the distribution $p(s'|v',v,s) = N_b \exp(-\frac{1}{b^2}\|s' - (s + (v' - v))\|_2^2)$. This distribution encourages that the state transition $s \to s'$ is close to parallel to the given node transition $v \to v'$.

Note that step (1), (3), and (4) correspond to the generative model proposed in Section 4.2.1. This 5-step generative model can be derived as follows:

$$p(T|G) = \prod_{i=1}^n p((s_i, a_i, s'_i)|G) = \prod_{i=1}^n p(s_i)p(a_i|s_i)p(s'_i|s_i, a_i)$$

Under the independence assumptions $I = \{v \perp a|s; v' \perp s|v,a; s' \perp a|v', v', s\}$, we have

$$\begin{aligned}
p(s'|s,a) &= \sum_v p(v, s'|s,a) = \sum_v p(v|s,a)p(s'|v,s,a) \\
&= \sum_v p(v|s,a) \sum_{v'} p(v', s'|v,s,a) \\
&= \sum_v p(v|s,a) \sum_{v'} p(v'|v,s,a)p(s'|v',v,s,a) \\
&\overset{I}{=} \sum_v p(v|s) \sum_{v'} p(v'|v,a)p(s'|v',v,s)
\end{aligned}$$

Inserting this in $p(T|G)$ and using Bayes rule $p(v|s) = p(s|v)p(v)/p(s)$ yields

$$p(T|G) = \prod_{i=1}^{n} p(s_i)p(a_i|s_i) \left[\sum_{v \in V} \frac{p(s_i|v)p(v)}{p(s_i)} \sum_{v' \in V} p(v'|v,a)p(s'_i|v',v,s_i) \right]$$
$$= \prod_{i=1}^{n} \underset{(3)}{p(a_i|s_i)} \left[\sum_{v \in V} \underset{(1)}{p(v)} \underset{(2)}{p(s_i|v)} \sum_{v' \in V} \underset{(4)}{p(v'|v,a)} \underset{(5)}{p(s'_i|v',v,s_i)} \right].$$

This formula corresponds to the 5-step generative model given above.

5.3.2 FIGE: Force-Based Iterative Graph Estimation

Given this generative model, the maximum likelihood estimate of the transition graph for a given set of transitions T would be $G^* = \arg\max_G L_T(G)$. Unfortunately, solving this problem directly is hard; we propose the FIGE heuristic, which aims at finding close-to-optimal transition graphs iteratively and is computationally tractable. FIGE is an iterative algorithm whose update equations are derived from the maximum likelihood objective using two simplifying assumptions (see Appendix A.2): (A1) For each transition $(s,a,s') \in T$, assume $p(v'|v,a) = 1$ if $v = \mathrm{NN}_V(s) \wedge v' = \mathrm{NN}_V(s')$ else 0, where $\mathrm{NN}_V(s) = \arg\min_{v \in V} ||s - v||^2$. This assumption implies that whenever action a is executed in any state of the Voronoi cell $Vo(v) = \{s \in \mathcal{S} | \mathrm{NN}_V(s) = v\}$ the successor state will be with probability 1 in $Vo(v')$. (A2) Assume $p(T|V) = \prod_{v \in V} p(T|v)$. This assumption implies that the choice of the positions of the graph nodes $v \in V$ can be made independently. Both assumptions are typically oversimplifying; A1 is more oversimplifying for domains whose dynamics are less locally smooth. The Assumption A2 on the other hand is more simplifying in strongly connected domains where many transitions from the Voronoi cell of one node to the Voronoi cell of another node occur. To account for some of the errors made because of the oversimplifications of A1 and A2, FIGE iteratively refines the graph node positions by applying the derived update equations several times. Note that FIGE is a heuristic and no guarantee for converging to G^* is given.

FIGE is summarized in Algorithm 5.1: The set of graph nodes V is initialized such that it covers the set of states contained in T uniformly by, e.g., maximizing the distance of the closest pair of graph nodes (line 3). Afterwards, for K iterations, the graph nodes are moved according to two kind of "forces" that act on them (see Figure 5.2): The "sample representation" force F_S (line 6-7) pulls each graph node v to the mean of all states S^v for which it is responsible, i.e., the states s for which it is the nearest neighbor $\mathrm{NN}_V(s)$ in V. Thus, this force encourages node positions that capture the on-policy state distribution well and corresponds to an intrinsic k-means clustering. The "graph consistency" force F_G (line 8-10) pulls each graph node v to a position where for all $(s,a,s') \in T$ with $\mathrm{NN}_V(s) = v$ there is a vertex v' such that $v' - v$ is similar to $s' - s$, i.e., both vectors are close to parallel. Thus, this force encourages node positions which can represent the domain's dynamics well. The nodes are then moved according to the

Algorithm 5.1 Force-Based Iterative Graph Estimation (FIGE)

1: **Input:** Transitions $T = \{(s_i, a_i, s_i')\}_{i=1}^{n}$, parameters v_{num}, K
2: # Choose initial node positions V from states in T s.t. the distance of closest pair is maximized
3: $V = \text{INITIALIZE}(T, v_{num})$ # $|V| = v_{num}$
4: **for** $i = 0$ to $K-1$ **do**
5: **for all** $v \in V$ **do**
6: $S^V[v] = \{s \mid \exists (s,a,s') \in T : \text{NN}_V(s) = v\}$ # Observed states in Voronoi cell $Vo(v)$
7: $F_S[v] = \text{MEAN}(S^V[v]) - v$ # Sample representation force
8: $T^{\rightarrow}(v) = \{\text{NN}_V(s') - s' + s \mid \exists (s,a,s') \in T : \text{NN}_V(s) = v\}$ # Transitions starting in $Vo(v)$
9: $T^{\leftarrow}(v) = \{\text{NN}_V(s) - s + s' \mid \exists (s,a,s') \in T : \text{NN}_V(s') = v\}$ # Transitions ending in $Vo(v)$
10: $F_G[v] = 0.5 \cdot [\text{MEAN}(T^{\rightarrow}(v)) + \text{MEAN}(T^{\leftarrow}(v))] - v$ # Graph consistency force
11: $V = V + \alpha_i \cdot 0.5(F_S[V] + F_G[V])$ # Update node positions (vector notation)
12: # Count transitions from Voronoi cell $Vo(v)$ to Voronoi cell $Vo(v')$ under action a
13: $n^a_{vv'} = |\{(s,s') \mid \exists\, (s,a,s') \in T : \text{NN}_V(s) = v \wedge \text{NN}_V(s') = v'\}|$
14: $E = \left\{(v,v') \mid v,v' \in V\ \exists a \in A : n^a_{vv'} > 0\right\}$ # Edge between v and v'
15: $w_{vv'} = \frac{1}{|A|} \sum_{a \in A} \frac{n^a_{vv'}}{\sum_{\tilde{v}} n^a_{v\tilde{v}}}$ # Off-policy edge weights from Section 4.2.1
16: **return** (V, E, w)

two forces (line 11), where the parameter $\alpha_i \in (0,1]$ controls how greedily the node is moved to the position where the forces would become minimal. In order to ensure convergence of the graph nodes, α_i should go to 0 for i approaching K. If not explicitly stated, we use $\alpha_i = \lceil i/5 \rceil^{-1}$ and $K = 15$. An edge is added between two nodes v and v' if there exists at least one transition $(s,a,s') \in T$ with s being in the Voronoi cell of $Vo(v)$ and s in $Vo(v')$ (line 14). Moreover, the off-policy edge weights derived in Section 4.2.1 are used (line 15).

The derivation of FIGE from the maximum likelihood objective is given in Appendix A.2. FIGE's property of first choosing the graph node positions V and afterwards choosing E and w is a direct consequence of Assumption A1. Similarly, Assumption A2 allows that FIGE can ignore node interactions within an iteration and chooses each graph node's position greedily.

FIGE's time complexity is dominated by the nearest neighbor queries: in every of the K iterations and for any s and s' occurring in T, the nearest neighbor in V need to be determined. Using $|T| = n$ and assuming that "naive", linear nearest-neighbor search is used, this requires $O(nv_{num}n_s)$, where n_s is the dimensionality of the state space. Thus, the time complexity of FIGE is in $O(Knv_{num}n_s)$.

Note that FIGE uses the assumption of a discrete action space solely in lines 13-15 for the importance sampling required for computing the off-policy edge weights. FIGE could be extended to continuous action spaces by using the on-policy edge weights $w_{on} = n_{vv'}/n_v$ with $n_{vv'} = |\{(s,s') \mid \exists\, (s,a,s') \in T : \text{NN}_V(s) = v \wedge \text{NN}_V(s') = v'\}|$ and $n_v = \sum_{v'} n_{vv'}$. Alternatively, other means for estimating the sampling policy could be employed which extend to policies over continuous action spaces.

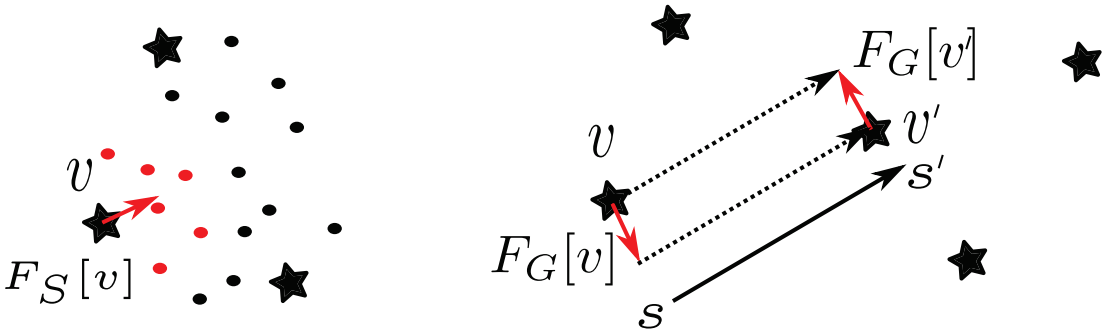


Figure 5.2 – *Illustration of the forces in FIGE.* Shown are the forces that act on graph nodes (depicted as stars) in FIGE. The left diagram depicts the sample representation force F_S acting on node v. This force is exerted by the set of states S^V (red dots) for which v is the closest graph node and pulls v to the mean of S^V. The left diagram shows the graph consistency force F_G exerted by the transition from state s to s' onto the graph nodes $v = \mathrm{NN}_V(s)$ and $v' = \mathrm{NN}_V(s')$. The force F_G pulls node v to position $v' - (s' - s)$ and node v' to position $v + (s' - s)$.

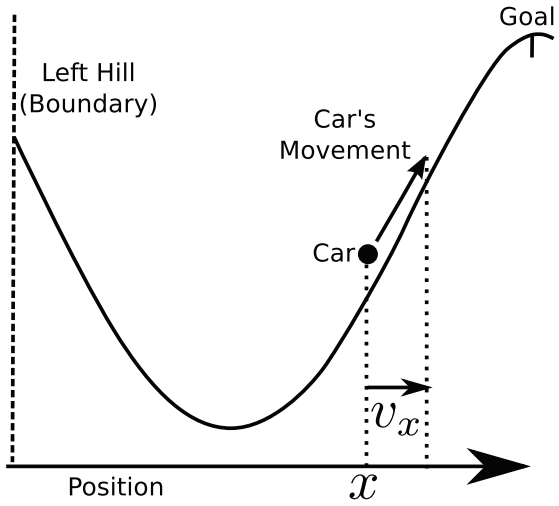


Figure 5.3 – *Illustration of the mountain car domain.* The car is denoted by a black dot; its movement is restricted to a one-dimensional surface. The objective of the car is to reach the top of the right hill. Since the car is underpowered, it cannot reach the goal directly but must first build sufficient energy by oscillating back and forth between the two hills. We refer to Sutton and Barto (1998, Chapter 8.4) for more details.

5.3.2.1 Illustration

We illustrate the different heuristics for transition graph generation in the *mountain car* domain (see Figure 5.3). In mountain car, the agent controls a car that is placed in a one-dimensional valley and must reach the top of the right hill. Since the car is underpowered, it cannot reach the goal directly but must first build sufficient energy by oscillating back and forth between the two hills. The agent observes two continuous state variables, its position x and velocity v_x, and can choose among the three discrete

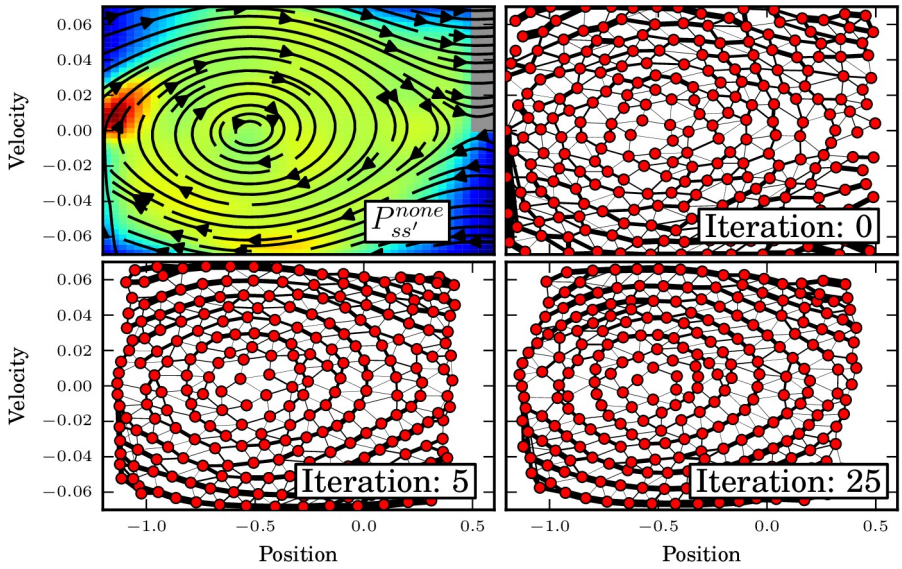


Figure 5.4 – *Illustration of the transition graphs during FIGE's iterations.* Every point in the upper left plot corresponds to one state and the streamlines show the dynamics of the domain when no force is applied to the car. Background colors show the density of the on-policy state distribution for a policy that selects actions uniform randomly and the gray area corresponds to the terminal region of the state space. The other plots visualize the graphs generated by FIGE after 0, 5, and 25 iterations for $v_{num} = 250$.

actions `left`, `none`, and `right`, which add -0.001, 0, and 0.001 to v_x, respectively. At each time step, x is incremented by v_x and due to gravity $-0.0025cos(3x)$ is added to v_x. The velocity v_x is constrained to a maximal absolute value of 0.07 and set to 0 if the top of the left hill is reached. The mountain car domain is well suited for illustration because of its two-dimensional state space.

The domain's dynamics for the `none` action are visualized in the upper left plot of Figure 5.4. The other plots show the graphs generated by FIGE after 0, 5, and 25 iterations for $v_{num} = 250$ and $|T| = 40000$. Because of its initialization of graph node positions, FIGE covers the state space of the mountain car domain already quite well before the first iteration of the force-based updates. Unfortunately, the dynamics of the domain can hardly be represented by edges of the graph for this choices of graph node positions. However, because of the graph consistency forces that act on the graph node positions, the representability of the domain's dynamics increases considerably with the number of iterations. At the same time, the sample representation forces ensure that all relevant parts of the state space remain covered by graph nodes. After 25 iterations, the graph structure reflects nicely the domain's dynamics.

Figure 5.5 illustrates the graphs generated by FIGE after 25 iterations and by the three other heuristics discussed in Section 5.2.1. While the transition graphs generated by the grid and the ε-net heuristics cover the state space close to uniformly, the domain's dynamics are hardly recognizable. Even worse, the on-policy heuristic generates graphs that do not cover the state space well because independent sampling does not take

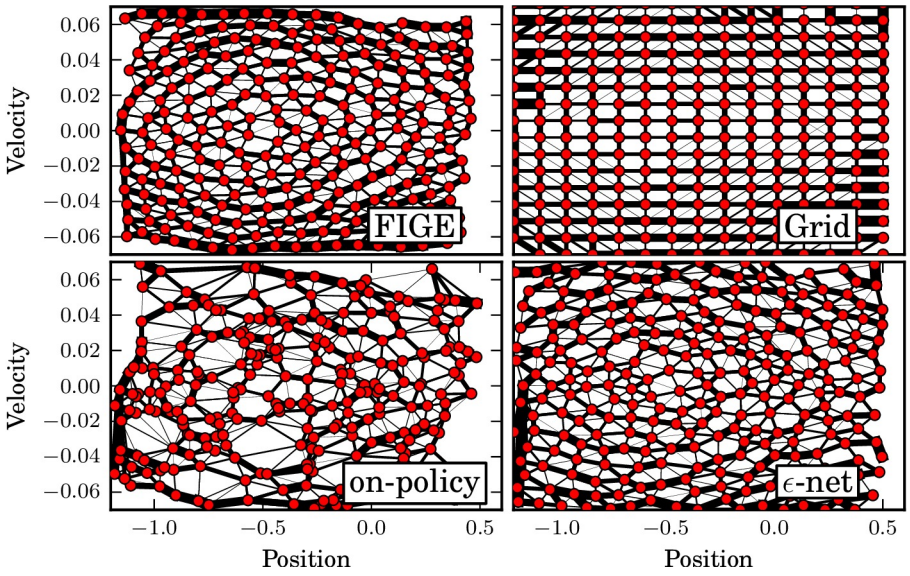


Figure 5.5 – *Illustration of four different heuristics for transition graph generation.* Shown are the generated transition graphs in the mountain car domain (cf. Figure 5.4).

into account the euclidean nature of the state space. In contrast, the transition graph generated by `FIGE` nicely reflects the domain's dynamics.

5.3.3 Skill Prototype Generation

Plugging `FIGE` in the graph-based skill discovery approach shown in Figure 5.1 allows generating a transition graph $G = (V, E, w)$ and by means of graph clustering its partition $\mathcal{P}_G$. For learning an option o based on a newly identified bottleneck, we need to choose an appropriate skill prototype $\Psi_o = (I_o, \beta_o, R_o)$ based on the identified partition $\mathcal{P}_G$. For this, the partition $\mathcal{P}_G = \{p_1, \ldots, p_n\}$ of the transition graph is generalized to a partition $\mathcal{P}_\mathcal{S} = \{\Pi_\mathcal{S}(p_1), \ldots, \Pi_\mathcal{S}(p_n)\}$ of the entire state space $\mathcal{S}$ by a nearest-neighbor based generalization $\Pi_\mathcal{S}$: for this, we set $\Pi_\mathcal{S}(p_i) = \{s \in \mathcal{S} \mid \mathrm{NN}_V(s) \in p_i\}$, i.e., we assign each state s to the cluster of its closest vertex $\mathrm{NN}_V(s)$.

Similar to Section 4.2.4, we can now create skill prototypes for each pair of connected clusters $A, B \in \mathcal{P}_G$: For each cluster A, one skill is generated for each adjacent[1] cluster B. The corresponding skill prototype $\Psi_{A\to B} = (I_{A\to B}, \beta_{A\to B}, R_{A\to B})$ is defined as:

$$I_{A\to B} = \Pi_\mathcal{S}(A) \qquad \beta_{A\to B}(s) = 0 \text{ if } s \in I_{A\to B} \text{ else } 1$$
$$R_{A\to B}((s,a,r,s')) = -1 \text{ if } s' \in \Pi_\mathcal{S}(A \cup B) \text{ else } r_p - 1.$$

The prototype $\Psi_{A\to B}$ corresponds to a skill that can be invoked anywhere in cluster A, terminates successfully everywhere in cluster B, and terminates unsuccessfully in all other clusters. The parameter $r_p \ll 0$ of the algorithm determines the penalty for such

[1] Two clusters A and B are adjacent in $G = (V, E)$ if there exists $v_a \in A, v_b \in B$ such that $(v_a, v_b) \in E$.

an unsuccessful termination. Otherwise a reward of -1 is given. Thus, the optimal policy corresponds to traversing the bottleneck area between A and B as fast as possible while not leaving the clusters A and B.

Additionally, for each cluster that contains nodes in which an episode has terminated, a special skill Ψ_{As_t} is created that can be invoked in any state of the cluster, terminates successfully when the episode terminates, and terminates unsuccessfully (i.e., obtains the penalty r_p) if the clusters is left (cf. Section 4.2.4). Note that in contrast to Mannor et al. (2004), the generalization of the graph partition to the entire state space allows performing the learning of skills and higher-level policies in the original MDP and not in a discretized version of it.

5.4 RESULTS

In this section, we present an empirical evaluation of the transition graphs generated by `FIGE` (see Section 5.4.1) and of a hierarchical RL architecture which uses skill discovery based on `FIGE` internally. Skill discovery based on `FIGE` is evaluated with regard to the quality of the obtained partitions in Section 5.4.2.2 and with regard to the learning performance of the whole hierarchical RL architecture in Section 5.4.2.3. We refer to Figure 4.1 for information on which parts of the overall architecture correspond to these evaluations.

5.4.1 Graph Likelihood

We empirically compare different heuristics for transition graph generation (see Section 5.2.1) with regard to the obtained likelihood $L_T(G)$ of the generated graphs G (cf. Section 5.3.1). We perform this analysis in the mountain car domain (see Section 5.3.2.1).

In the left graph of Figure 5.6, the likelihood $L_T(G)$ with $b = 0.02$ is shown for different heuristics and different number of graph nodes v_{num}. Since the likelihood depends on v_{num}, we plot the ratio of the method's likelihood relative to the likelihood obtained by the data-independent grid heuristic. Furthermore, we use a logarithmic scale for plotting this ratio. Note that the likelihood has been evaluated on test transitions T_{test} which were different from the training transitions T_{train} that have been used for the optimization of graph node positions ($|T_{train}| = 5000$, and $|T_{test}| = 10^5$).

Regardless of v_{num}, the largest likelihood $L_T(G)$ is achieved by `FIGE` and the smallest by the grid heuristic (for $v_{num} > 75$). The on-policy sampling heuristic performs slightly better than ε-net for $v_{num} < 150$ while both perform similarly for larger v_{num}. A possible explanation for the stronger deterioration of ε-net is that for small v_{num}, the minimal node distance ε gets larger than the typical distance of states and their successors and thus, $P^a_{ss'}$ cannot be represented in any part of the state space. In contrast, on-policy sampling allocates more graph nodes in densely sampled parts of the state space and thus allows modeling at least these parts of the state space. `FIGE` can achieve a

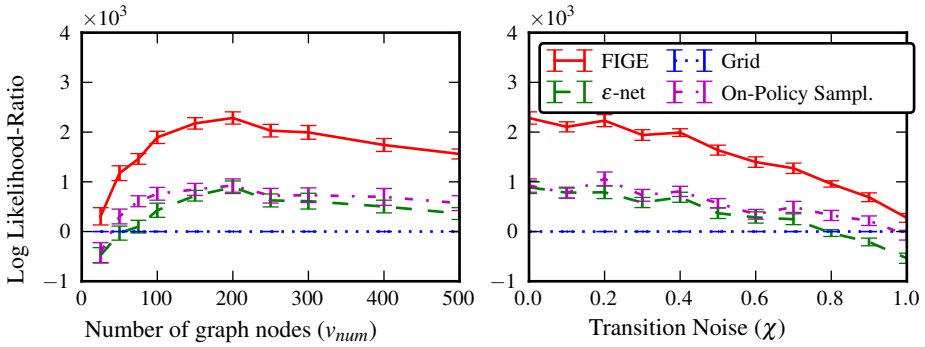


Figure 5.6 – *Log-likelihood of transition graphs.* Left graph: Graph Log Likelihood-Ratio relative to grid heuristic in deterministic mountain car for different values for the number of graph nodes. Right graph: Graph Log Likelihood-Ratio relative to grid heuristic in the stochastic mountain car domain for varying stochasticity χ and $v_{num} = 200$. Shown are mean and standard error of the mean over 20 repetitions.

considerably larger likelihood $L_T(G)$ than both by taking the domain's dynamics into account explicitly.

In a second experiment, we evaluate how robust the different heuristics are with regard to increasing stochasticity χ in the domain's state transition probability $P^a_{ss'}$. For this, we modify any transition from state s to s' governed by the domain's deterministic dynamics such that the i-the dimension of the actual successor state is changed to $s'_i \leftarrow s'_i + \chi_i(s'_i - s_i)$ with χ_i sampled uniformly from $[-\chi, \chi]$. Note that this is not purely observation noise since the actual state of the environment is altered. The quantity χ controls how "strong" the stochasticity of the domain is, with $\chi = 0$ corresponding to the deterministic domain. The same amount of transition noise was also used for generating the test transitions T_{test}. The results are shown in the right graph in Figure 5.6. As expected, the grid-based heuristic deteriorates less with increasing stochasticity as it does not take the observed transitions into account. Nevertheless, the other data-dependent heuristics achieve better likelihood $L_T(G)$ for $\chi < 0.8$ with `FIGE` remaining the best heuristic for the whole investigated range of $\chi \in [0, 1]$. This shows that `FIGE` is also suited for stochastic domains.

5.4.2 Skill Discovery

In this section, we present an empirical evaluation of the proposed skill discovery approach in the 2D Multi-Valley domain (see Section 5.4.2.1). We consider the quality of the obtained partitions in Section 5.4.2.2 and the learning performance in Section 5.4.2.3.

5.4.2.1 2D Multi-Valley Domain

The 2D Multi-Valley environment (see Figure 5.7) is an extension of the basic mountain car domain. The car the agent controls is not constrained to a one-dimensional surface but to a two-dimensional one. This two-dimensional surface consists of $2 \times 2 = 4$ valleys,

whose borders are at $(\pi/6 \pm \pi/3, \pi/6 \pm \pi/3)$. The agent observes $n_s = 4$ continuous state variables: the positions in the two dimensions (x and y) and the two corresponding velocities (v_x and v_y). The agent can choose among the four discrete actions `northwest`, `northeast`, `southwest`, `southeast` which add $(\pm 0.001, \pm 0.001)$ to (v_x, v_y). In each time step, due to gravity $0.004\cos(3x)$ is added to v_x and $0.004\cos(3y)$ to v_y. The maximal absolute velocity in each dimension is constrained to 0.07. The agent is faced with a multi-task scenario: in each episode, the agent has to solve one out of twelve tasks. Each task is associated with a combination of two distinct valleys; e.g., in task $(0,1)$ the agent starts in the floor[1] of valley 0 and has to navigate to the floor of valley 1 and reduce its velocity such that $\left|\left|(v_x, v_y)\right|\right|_2 \leq 0.03$. In each time step, the agent receives a reward of $r = -1$. Once a task is solved, the next episode starts with the car remaining at its current position and one of the tasks that starts in this valley is drawn at random. The current task is communicated as an additional state space dimension to the agent; the agent uses it for compositional learning but ignores it during graph generation, graph clustering, and skill learning such that skills are reusable in different tasks.

We have chosen this domain for the empirical evaluation in this section for two reasons: (1) there exists a ground-truth partition of the domain into 4 clusters, which correspond to the domain's valleys. This has the advantage that we can compare both the performance of the learning system as a whole and the quality of the obtained partitions themselves with regard to the ground-truth partition. (2) The domain consists of multiple tasks and is thus a good choice for analyzing if the discovered skills allow transferring procedural knowledge across tasks and in which situations this transfer becomes important. This was stated in Section 3.2.1 as one of the potential benefits of skill discovery.

5.4.2.2 Cluster Accordance Analysis

In this section, we present an empirical evaluation of the quality of the partitions generated by non-incremental clustering of graphs that have been generated with different heuristics for transition graph generation. The quality of a partition $\mathcal{P}$ is measured using the accordance ratio $\text{acc}_{GT}(\mathcal{P})$, cf. Section 3.4.1. The corresponding ground truth partition $\mathcal{P}_{GT}$ consists of 4 clusters corresponding to the 4 valleys (see Figure 5.7), i.e., the boundaries of the clusters are at $x = \pi/6$ and $y = \pi/6$. The accordance ratio acc_{GT} is defined only for discrete domains since it involves averaging over all pairs of states; we extend it to the continuous domains by approximating $\text{acc}_{GT}(\mathcal{P}) \approx \frac{1}{N}\sum_{i=0}^{N-1} \delta(\delta([s_i]_{\mathcal{P}}, [\bar{s}_i]_{\mathcal{P}}), \delta([s_i]_{\mathcal{P}_{GT}}, [\bar{s}_i]_{\mathcal{P}_{GT}}))$, where s_i and $\bar{s}_i$ are sampled uniform randomly from the state space.

Figure 5.8 shows the accordance ratio with the ground truth acc_{GT} for partitions resulting from different heuristics for transition graph generation and varying number of graph nodes (for $n = 30000$ transitions and $\psi = -0.015$). Independent of the number graph nodes v_{num}, `FIGE` is the most robust heuristic and obtains an accordance of

[1] The floor of valley 0 (see Figure 5.7) corresponds to the region $((-1/6 \pm 2/15)\pi, (-1/6 \pm 2/15)\pi)$.

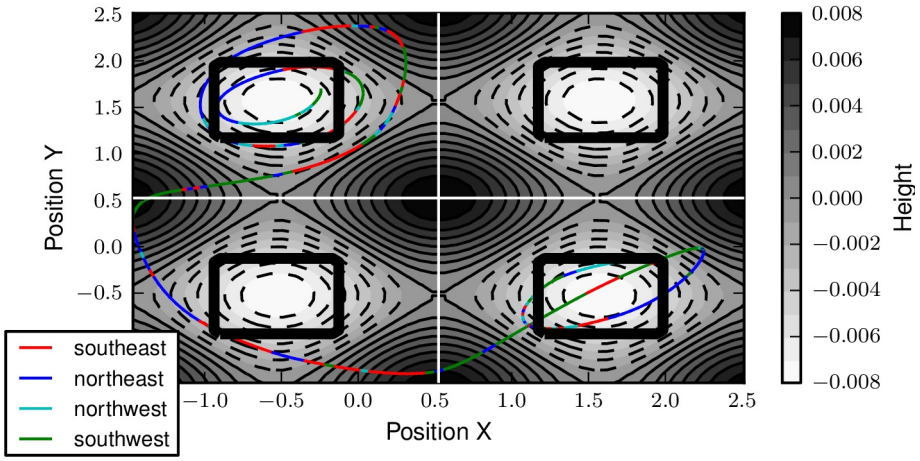


Figure 5.7 – *Illustration of the 2D Multi-Valley domain.* Gray-scale contours depict the height of the two-dimensional surface. The black boxes denote the target regions of the different tasks and the white lines the boundaries of the ground truth partitions of the state space. Shown is one example trajectory with color-coded actions.

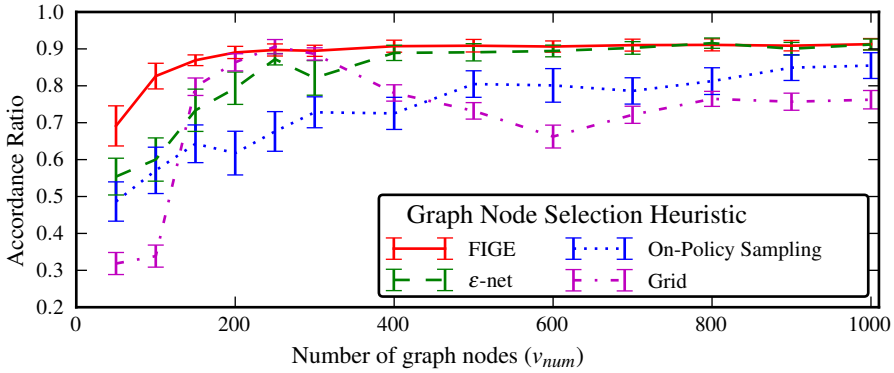


Figure 5.8 – *Accordance ratio of graph partitions.* The diagram depicts the accordance ratio acc_{GT} of graph partitions with ground-truth for varying number of graph nodes and different heuristics for transition graph generation. Graphs have been estimated based on $n = 30000$ transitions. Shown are mean and standard error of the mean over 20 repetitions.

$\text{acc}_{GT} \geq 0.8$ for any $v_{num} \geq 100$ and $\text{acc}_{GT} \geq 0.89$ for any $v_{num} \geq 200$. For sufficiently many graph nodes, the ε-net heuristic achieves comparable results. However, for $v_{num} \leq 200$, the ε-net performs significantly worse ($p < 0.05$). This indicates that taking the domain's dynamics into account is particularly important when a small number of graph nodes is desired. The on-policy sampling heuristic performs typically worse than the ε-net heuristic, with statistically significant differences for $v_{num} \geq 200$. This shows that it is important to cover the effective state space uniformly with graph nodes to obtain a good partition of the state space. The grid heuristic shows the most complex pattern: while it can achieve high accordance for small graphs ($\text{acc}_{GT} \geq 0.85$ for $200 \leq v_{num} \leq 300$), it performs considerably worse for smaller and larger v_{num}, with, e.g., an accordance of $\text{acc}_{GT} \approx 0.66$ for $v_{num} = 600$. Apparently, there is a complex,

non-monotonic relationship between the grid resolution and the quality of the partition which is probably governed by how well the grid nodes align to the valley borders. This reduces the usability of the grid-based heuristic since the choice of an appropriate grid resolution becomes difficult.

5.4.2.3 Learning Performance

This section presents an empirical comparison of the learning performance of the entire hierarchical RL architecture shown in Figure 5.1. Skill discovery has been performed after $n = 10^5$ state transitions have been observed in the environment and graphs with $v_{num} \in \{50, 100, 200, \ldots, 500\}$ nodes have been generated. These graphs have been clustered with non-incremental agglomerative clustering (cf. Section 2.4.3.3) for $\psi = -0.015$. Each option's value function has been represented by a CMAC function approximator (see Section 2.3.2) consisting of 10 independent tilings with $7^2 \cdot 5^2$ tiles, where the higher resolutions have been used for the x and y dimensions. The penalty of an unsuccessful option has been set to $r_p = -1000$ and the value functions have been initialized to -100. For learning the compositional option π, a lower resolution of $5^2 \cdot 3^2$ tiles has been used and the value functions have been initialized to -1000. The discounting factor has been set to $\gamma = 1$ and all policies were ε-greedy with $\varepsilon = 0.01$. The value functions were learned using Q-Learning and updated only for currently active options with a learning rate of 1. Episodes have been interrupted after 10^4 steps without solving the task and a new task was chosen at random. All parameters have been chosen based on preliminary investigations.

Two baselines from Section 3.4.2 have been evaluated in the same setting: (i) a *monolithic* approach which learns a flat policy for every task without using skills, and (ii) the same hierarchical RL framework with *predefined* skill prototypes $\Psi_o = (I_o, \beta_o, R_o)$. These prototypes have been generated in the same way as those discovered using graph clustering but are based on the ground-truth partition $\mathcal{P}_G$ of the domain (see Figure 5.7). Thus, baseline (ii) presents an upper boundary for the performance that any bottleneck-based skill discovery method can achieve within the given hierarchical RL architecture.

Figure 5.9 shows the accumulated reward R^t_{acc} (see Section 3.4.1) obtained during the first 2400 learning episodes (the phase of learning during which the explorative bias provided by skills has the strongest impact) for different transition graph generation heuristics and different number of graph nodes v_{num}. For too small v_{num}, e.g., $v_{num} = 50$, no heuristic for transition graph generation was able to obtain good results. Moreover, one can see that the grid-based heuristic obtains poor results for any choice of v_{num}. When using many graph nodes ($v_{num} = 500$), no considerable differences between the other heuristics exist. However, for intermediate values of v_{num}, e.g., $v_{num} \in \{100, 150\}$, `FIGE` obtains significantly better results than the ε-net and the on-policy sampling heuristic ($p < 0.001$, Mann-Whitney U-test). In contrast to the other heuristics, the accumulated reward obtained by the `FIGE` heuristic does not deteriorate considerably for $v_{num} \in \{100, 150\}$ compared to $v_{num} = 500$. In summary, `FIGE` allows creating skills that can provide a useful explorative bias during learning based on smaller graphs than

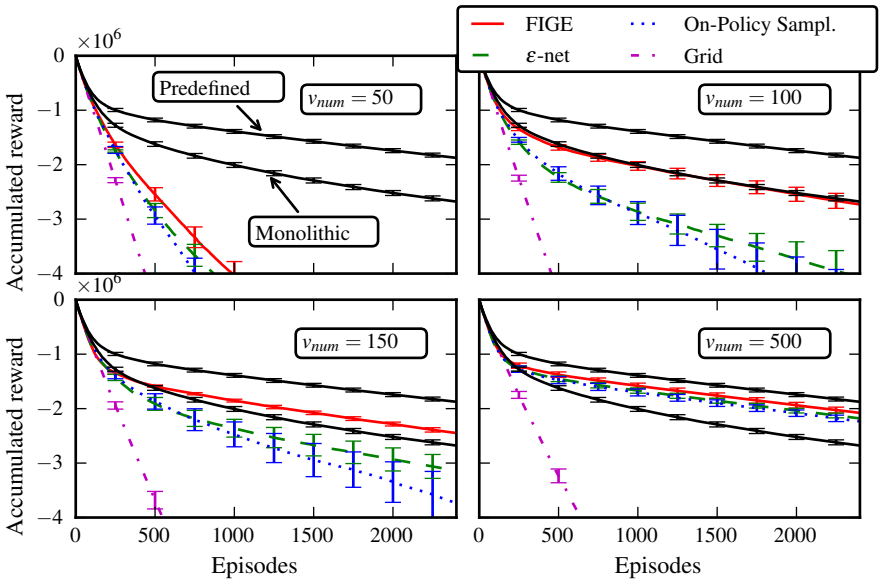


Figure 5.9 – *Evaluation of FIGE-based skill discovery in the 2D multi-valley.* The diagram depicts the accumulated reward R^t_{acc} in the 12-task 2D Multi-Valley domain during 2400 episodes of learning for different graph generation heuristics and graph sizes. Shown are mean and standard error of the mean over 10 repetitions.

other heuristics which allows reducing computation time during graph clustering and learning.

Comparing FIGE to the baselines, the accumulated reward obtained by FIGE for $v_{num} \geq 150$ is slightly larger than when learning a flat policy with the monolithic baseline. For $v_{num} = 500$, this difference becomes considerable. Thus, the proposed approach can effectively transfer knowledge between tasks and outperform the monolithic approach. Furthermore, the performance of FIGE ($v_{num} = 500$) is only slightly worse than the performance of the predefined skills baseline. This "lost" reward compared to the predefined skills occurs entirely in the first 500 episodes and can thus be attributed to the overhead of graph generation and skill discovery. In later episodes, both curves are nearly parallel, i.e., a similar amount of reward is accumulated. Thus, the skills identified and the learned hierarchical policy are equally good for both FIGE and the predefined, ground-truth skills.

In order to investigate the trade-off between the overhead of learning a hierarchical policy and skill discovery and the performance gain that can be obtained by reusing skills, we have performed a second experiment. In this experiment, the number of tasks the agent has been faced with was varied. For this, in each run a subset of the total of 12 tasks of the domain has been subsampled at random. The results of the evaluation are shown in Figure 5.10. Shown is the accumulated reward in the first 200 learning episodes of each task averaged over the tasks for $v_{num} = 500$.

The performance of the monolithic approach stays approximately constant since it must learn a separate policy for each task from scratch because it cannot transfer

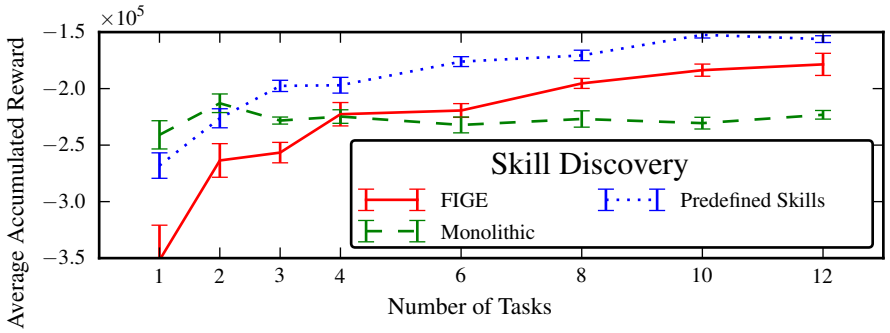


Figure 5.10 – *Skill discovery in the 2D Multi-Valley domain for different number of tasks.* Shown is the average accumulated reward R_{acc}^{200} of 200 learning episodes per task in the 2D Multi-Valley domain. The diagram depicts a comparison of a monolithic approach, a hierarchical approach provided with the ground-truth partition of the domain (predefined), and a hierarchical approach that learns this partition concurrently to learning a policy based on a transition graph generated with `FIGE`.

procedural knowledge in the form of skills between tasks.[1] If only one task exists, the hierarchical approaches perform worse than the monolithic one ($p < 0.025$, t-test for related samples). This is due to the overhead of learning a hierarchical policy which does not pay off in a single-task problem. However, when the number of tasks increases, the hierarchical approaches perform significantly better than the monolithic one ($p < 0.004$ for 3 or more tasks for predefined skill prototypes and 8 or more tasks for skill discovery based on `FIGE`). This is because the hierarchical approaches can reuse options across tasks and thus transfer learned knowledge. This supports the hypothesis from Section 3.2.1 that one of the major advantages of hierarchical RL approaches (also for continuous domains) is that they can efficiently transfer knowledge between different but related tasks. Furthermore, one can also see that the difference between the hierarchical approach using predefined skill prototypes and the one where skill discovery is based on `FIGE` becomes smaller for more tasks. This indicates that the overhead of skill discovery, i.e., transition graph generation and graph clustering, becomes less relevant the more different tasks the agent has to solve.

5.5 EXCURSUS: REPRESENTATION LEARNING

Besides graph-based skill discovery, representing domain models in the form of a transition graph is also used in other subfields of RL, most notably in *representation learning* (Mahadevan and Maggioni, 2007). In the context of RL, representation learning refers to a situation where the agent learns an appropriate representation for policies and value functions autonomously. This contrasts with the standard approach to RL in which a function approximator with fixed parametrization is given and the goal of RL is "solely" to learn the optimal parameters for this representation. In this section, we

[1] Recall that the task is a discrete state space dimension over which the agent does not generalize.

Algorithm 5.2 Representation Policy Iteration (RPI)

Input: Transitions $T = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{n}$, parameters v_{num}, p_{num} (number of proto-value functions)
Create transition graph with one of the heuristics discussed in Section 5.2.1
Note: Euclidean connectivity is used for choosing graph edges and their weights
$G = \text{TRANSITIONGRAPHGENERATION}(T, v_{num})$
Compute diffusion operator on graph (normalized or combinatorial Laplacian)
$\mathcal{O} = \text{LAPLACIAN}(G)$
Use p_{num} smoothest eigenvectors of $\mathcal{O}$ as basis functions; $\Phi : v_{num} \times p_{num}$ matrix
$\Phi = \text{EIGENDECOMPOSITION}(\mathcal{O}, p_{num})$
Perform Nyström extension to generalize Φ to proto-value functions $\hat{\Phi}$
$\hat{\Phi} = \text{NYSTRÖMINTERPOLATION}(G, \Phi)$
Determine policy using least-squares policy iteration
return $\pi = \text{LSPI}(T, \hat{\Phi}, \gamma, \varepsilon)$

extend one method for representation learning in RL by incorporating `FIGE` and show that `FIGE` can also improve performance of this method considerably. This section can be seen as an excursus and may be skipped by the reader.

5.5.1 Method

Representation Policy Iteration (RPI) is an approach proposed by Mahadevan and Maggioni (2007) that aims at solving MDPs by jointly learning representations and optimal policies. In contrast to most other RL algorithms, RPI does not require an a-priori specification of basis functions. The main idea for learning basis functions is to first learn a transition graph of the domain and to construct a symmetric diffusion operator on this graph. The unnormalized combinatorial graph Laplacian $\mathcal{L} = D - W$ and the symmetrized graph Laplacian matrix $\mathcal{L}_{sym} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ are examples for such diffusion operators—with W being the graph's symmetrized weight matrix and D being a diagonal matrix whose entries are the row sums of W (see Section 2.4.3.1). The smoothest eigenvectors (those with the smallest associated eigenvalues) of these operators, the so-called proto-value functions (PVFs), are used as basis functions Φ for representing value functions. Least-squares policy iteration (LSPI, cf. Section 2.3.1) is used for learning the parameters w^{π} of the action value function $Q^{\pi} = w^{\pi}\Phi$ of an ε-optimal policy π within the linear span of the basis functions Φ.

RPI can also be used in MDPs with continuous state space. In such continuous domains, one challenge is to choose the node position of the transition graph ("to subsample a set of states" in the terminology of Mahadevan and Maggioni). The authors discuss the usage of the on-policy sampling and the ε-net heuristics (cf. Section 5.2.1). However we will show that considerable better results can be achieved by using `FIGE`. In RPI, each graph node is connected to its k nearest neighbor nodes in the euclidean space and the edge weight between nodes v_i and v_j is $w((v_i, v_j)) = \tau(i)\exp(-\frac{||v_i - v_j||_2^2}{\kappa})$ where $\tau(i)$ and κ are parameters to be specified. Note that this way of connecting graph nodes has the potential disadvantage that proximity of nodes in the euclidean space does not necessarily imply that a transition between these nodes is possible, e.g., if an

obstacle lies between those states. Choosing graph edges based on observed transitions between states (see Section 5.2.1) lessens this issue and seems thus preferable. However, for consistency with the original approach of Mahadevan and Maggioni (2007), we adhere in this section the "euclidean" connectivity.

Once a transition graph for the continuous domain has been generated, the eigendecomposition of its Laplacian can be used for computing PVFs. However, since these eigenvectors only define the value of the PVFs at the positions of the graph nodes, a means for generalizing these values over the entire state space is required. As suggested by Mahadevan and Maggioni (2007) we use the Nyström interpolation method for this. The resulting PVFs can now be used in LSPI for computing a policy. Pseudo-code for RPI is given in Algorithm 5.2 and a corresponding data-flow diagram is shown in Figure 5.11. For details we refer to Mahadevan and Maggioni (2007).

5.5.2 Illustration

We illustrate the combination of `FIGE` with RPI in the mountain car domain (cf. Section 5.3.2.1). For this, a transition graph has been generated using `FIGE` for $|T| = 40000$ and $v_{num} = 250$. The eigen-decomposition of the graph's Laplacian yields 250 PVFs. Figure 5.12 shows three exemplary PVFs with high smoothness, i.e., small eigenvalues. The first PVF ϕ_1 exhibits large positive activation for nodes v which correspond to a car moving away from the goal ($\phi_1((x,v_x)) \gg 0$ for $v_x \ll 0$) and large negative activation for nodes v which correspond to a car moving towards the goal ($\phi_1((x,v_x)) \ll 0$ for $v_x \gg 0$). The second PVF ϕ_2 exhibits large positive activation for nodes v which correspond to the car being close to the goal ($\phi_1((x,v_x)) \gg 0$ for $x > 0$) and large negative activation for nodes v which correspond to the car being far away from the goal ($\phi_1((x,v_x)) \gg 0$ for $x < -0.5$). The fourth PVF ϕ_4 exhibits large positive activation for nodes v which correspond to a car in a low energy state, i.e., with velocity close to 0 and position close to the bottom of the valley. Correspondingly, ϕ_4 exhibits large negative activation for a car in a high energy state. Thus, these three PVFs encode most of the relevant information: how close the car is to the goal (ϕ_2), in which direction it is moving (ϕ_1), and whether it has gathered sufficient energy to reach the goal on the top of the hill (ϕ_4). In particular, ϕ_4 encodes a non-trivial concept for this domain: the current energy

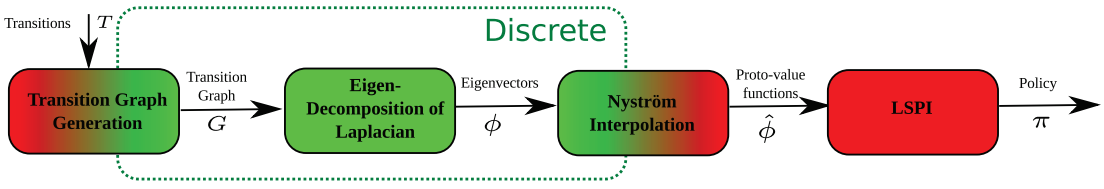


Figure 5.11 – *Data flow diagram of Representation Policy Iteration for continuous domains* (cf. pseudo-code in Algorithm 5.2). The diagram shows the transition between parts of the data flow that deal with the continuous domain (dyed in red) and a discretized version of it (dyed in green).

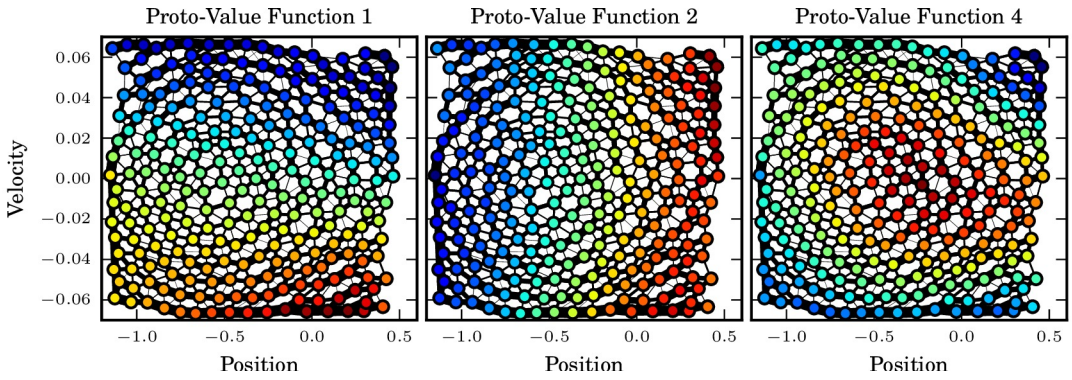


Figure 5.12 – *Illustration of Representation Policy Iteration in mountain car*. Shown are three proto-value functions determined on a transition graph generated using `FIGE` (cf. Figure 5.4). Colors encode the proto-value functions activation with red corresponding to strong positive activation and blue corresponding to strong negative activation. Note that the Nyström interpolation is not shown in this diagram.

state of the car. Learning a policy based on these PVFs is thus considerably simplified compared to learning with standard function approximators like CMACs.

5.5.3 Evaluation

We evaluate RPI in combination with `FIGE` in three RL benchmark problems with continuous state spaces: mountain car, inverted pendulum, and the octopus arm. In the original paper (Mahadevan and Maggioni, 2007), at the end of each episode an additional set of samples is collected either on- or off-policy. We skip this additional sampling and use the samples collected during control learning also for representation learning to show that some heuristics for transition graph generation can deal with this better than others. In order to initialize representation and control learning, the agent explored the environment uniform randomly during the first 10 episodes. Thereupon, RPI was performed at the end of each episode and the policy obtained was followed ε-greedily.

5.5.3.1 Mountain Car

In a first experiment, we evaluate the performance of RPI for different transition graph generation heuristics and different degrees of transition noise χ in mountain car (see Section 5.3.2.1) for $v_{num} = 50$. For all heuristics, we obtained the best results when setting the number of proto-value functions equal to the number of graph nodes, i.e., $p_{num} = v_{num}$. Moreover, in accordance with Mahadevan and Maggioni (2007) we set the discount factor γ to 0.99 and the exploration rate to $\varepsilon = 0.01$. We used the symmetrized graph Laplacian $\mathcal{L}_{sym}$ as graph operator. The results are shown in Figure 5.13. The left plot shows learning curves of RPI in the deterministic mountain car domain: RPI performs best when combined with `FIGE` and worst when combined with the grid heuristic while on-policy sampling and ε-net achieve approximately the same

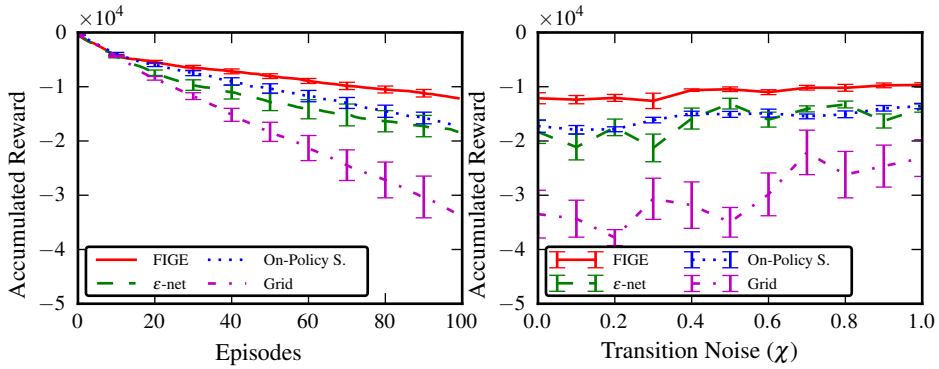


Figure 5.13 – *Evaluation of Representation Policy Iteration in mountain car.* The diagrams depict the accumulated reward R_{acc} obtained by RPI for different heuristics for transition graph generation. Left plot: learning curves of R^t_{acc} versus number of episodes t in the deterministic domain. Right plot: accumulated reward R^{100}_{acc} after 100 episodes for different degrees of transition noise χ. Shown are mean and standard error of mean over 10 repetitions.

results (no significant differences). This can be attributed to the fact that due to the randomly chosen start states of episodes, the on-policy state distribution (over several episodes) does not vary too strongly over the effective state space. Thus, sampling from the on-policy distribution yields in this domain graph nodes that cover the effective state space close to uniform. The worse results of the grid heuristic show that even for low-dimensional domains, a uniform discretization can be detrimental.

The right plot shows how the reward accumulated after 100 episodes changes for different degrees of stochasticity of the domain, where the same stochasticity model as in Section 5.4.1 has been used. In general, increasing transition noise seems to make the task easier as the performance increases for all heuristics; probably because the value function becomes smoother and thus better representable. However, the relative order of different methods remains the same. This reinforces that `FIGE` can be used in stochastic domains as well.

5.5.3.2 Inverted Pendulum

In the inverted pendulum problem, an agent has to learn to balance a pendulum that is attached via an unactuated joint to a cart by applying a force to this cart. The vertical angle θ of the pendulum and its angular velocity $\dot{\theta}$ make up the two-dimensional state space of the system. According to Lagoudakis and Parr (2003), the nonlinear dynamics of the system are given by

$$\ddot{\theta} = (g\sin(\theta) - \alpha ml\dot{\theta}^2\sin(2\theta)/2 - \alpha\cos(\theta)u)/(4l/3 - \alpha ml\cos(\theta)),$$

where $u \in \{-50\,\mathrm{N}, 0\,\mathrm{N}, 50\,\mathrm{N}\}$ denotes the force applied by the agent to the cart, the gravity is $g = 9.81\,\mathrm{m/s^2}$, $m = 2.0\,\mathrm{kg}$ is the mass of the pendulum, $M = 8.0\,\mathrm{kg}$ is the mass of the cart, $l = 0.5\,\mathrm{m}$ is the length of the pendulum, and $\alpha = (m+M)^{-1}$. Each episode starts with $\theta = .1$ and $\dot{\theta} = 0$. The agent obtains a reward of $+1$ for every time

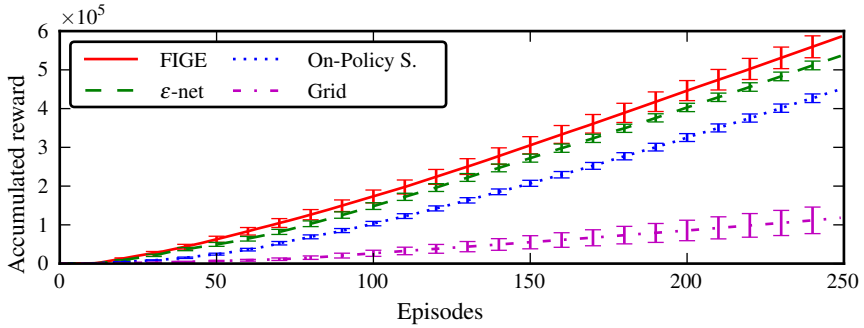


Figure 5.14 – *Evaluation of Representation Policy Iteration in inverted pendulum.* The diagram depicts the accumulated reward R^t_{acc} obtained by RPI for different transition graph generation heuristics and $v_{num} = 25$. Shown are mean and standard error of the mean over 30 repetitions.

step it manages to keep θ in a range of $[-\pi/2, \pi/2]$ around the vertical orientation $\theta = 0$. The episode ends once the agent fails to balance the pendulum or after 3000 steps when it is assumed that the agent could balance the pendulum indefinitely. The discount factor was set to $\gamma = .95$ and the exploration ratio to $\varepsilon = 0.025$. Based on prior investigations, the number of graph nodes v_{num} and proto-value functions p_{num} were set to 25. We used the symmetrized graph Laplacian $\mathcal{L}_{sym}$ as graph operator.

Figure 5.14 visualizes the accumulated reward obtained during learning for different transition graph generation heuristics. Similar to the results in the mountain car domain, the worst results are obtained by the grid heuristic and the best results by the `FIGE` heuristic.[1] In contrast to the results in mountain car, the ε-net heuristic achieves considerably more accumulated reward than the on-policy sampling heuristics. This is probably due to the fact that successful balancing typically corresponds to staying in the same part of the state space for approximately 3000 steps. The on-policy sampling heuristic samples uniformly among the visited states and will thus typically choose too many graph nodes in this part of the state space and too few in other parts after the first episode of successful balancing. Both ε-net and `FIGE` avoid this by having the objective to cover the effective state space uniformly and not the on-policy distribution. The small but significant advantage of `FIGE` compared to ε-net emphasizes that taking the domain's dynamics into account is also useful in the inverted pendulum domain even though the effect is not as strong as in other domains.

5.5.3.3 Octopus Arm

In a third experiment, we investigate the performance of RPI using `FIGE` for transition graph generation in the octopus arm domain[2] (Yekutieli et al., 2005). The dynamics of this domain are based on a two-dimensional biomechanical model of the octopus arm, in which the arm is modeled as a multi-segment structure. Each segment of this

[1] All pairwise comparisons yield significant differences with $p < 0.001$ for a Mann-Whitney U-test.

[2] Source code available via `http://cs.mcgill.ca/~dprecup/workshops/ICML06/octopus.html`.

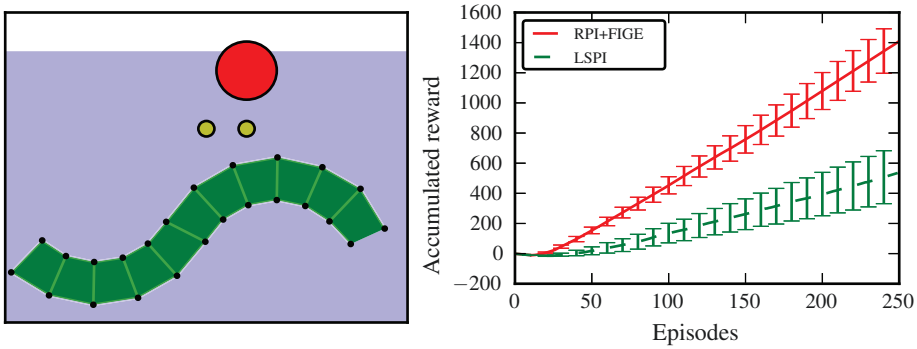


Figure 5.15 – *Evaluation of Representation Policy Iteration in octopus arm.* Left plot: Visualization of the octopus arm task. Right plot: Accumulated reward R^t_{acc} obtained by LSPI and RPI using FIGE for transition generation in the octopus arm domain. Shown are mean and standard error of mean over 20 repetitions.

structure contains longitudinal and transverse muscles and must maintain a constant volume, which is a prominent feature of muscular hydrostats (Kier and Smith, 1985). The input to the model is the degree of activation of each of its muscles. Furthermore, the model takes external forces such as gravity, buoyancy and water drag forces as well as internal forces by the arm muscles and the forces responsible for maintaining a constant volume of the arm into account.

The specific task is depicted in the left plot of Figure 5.15: the agent has to control the octopus arm such that it moves two food items (small yellow circles) into its mouth (large red circle). The base of the arm is restricted and cannot be actuated directly. The agent may control the arm in the following way: elongating or contracting the entire arm, bending the first half of the arm in either of the two directions, and bending the second half of the arm in either of the two directions. In each time step, the agent can control the elongation, the first half, and the second half of the arm independently, resulting in 8 discrete actions. The agent observes the positions x_i, y_i and velocities $\dot{x}_i$, $\dot{y}_i$ of the food items and of 24 selected parts of its arm (denoted by small black dots in Figure 5.15) and the angle and angular velocity of the arm's base. Thus, the state space is continuous and consists of 106 dimensions. Each dimension is normalized such that its values fall into $[0,1]$. The agent obtains a reward of -0.01 per time step, a reward of 5 for moving the left food item into its mouth, and a reward of 7 for the right food item. The episode ends after 100 time steps or once both food items have been eaten. Because of the high-dimensional state space and the complex dynamics of the domain, the octopus arm problem is a challenging benchmark task for RL.

We compare RPI combined with FIGE for $v_{num} = 75$ and $p_{num} = 5$ to LSPI using 75 radial basis functions (RBFs) as features ($\gamma = 0.99$, $\varepsilon = 0.01$). In the first 10 episodes, pure exploration without learning was conducted. The RBF centers c_i have been set to observed states such that the pairwise distance of the centers becomes maximal; the feature activation of center c_i for state s is computed as $\phi_i(s) = \exp(10||c_i - s||_2^2)$. The right plot of Figure 5.15 summarizes the results: using FIGE-based proto-value functions performs considerably better than standard RBF features; in particular, the

agent learns in each run to move at least one food item into its mouth, which is not the case for LSPI. The main difference between the two approaches is that RBFs are local while proto-value functions can also capture more global properties. We suppose that since FIGE allows capturing the dynamics of a domain well, it allows learning non-local proto-value functions that provide a useful bias to LSPI. In summary, the results suggest that FIGE can also support learning in high-dimensional problems.

5.6 DISCUSSION

This chapter has presented a new view on graph-based RL in domains with continuous state spaces. Based on interpreting transition graphs as generative models of the domain's dynamics, a novel formulation for the likelihood of a graph for a given set of transitions was proposed (see Section 5.3.1). Based on this, the new heuristic FIGE has been derived from the maximum likelihood objective under simplifying assumptions (see Section 5.3.2 and Appendix A.2). FIGE allows generating transition graphs that capture the domain's dynamics better than other heuristics. This is also reflected in the performance of skill discovery and representation learning methods that are built upon transition graphs: in both kind of methods and across different domains, FIGE has achieved superior and more robust results than prior heuristics for transition graph generation:

- FIGE generates transition graphs which explain observed transitions better, i.e., which have a higher likelihood (see Section 5.4.1).
- Graph clustering based on transition graphs generated with FIGE allows identifying bottlenecks of a domain more reliably and with smaller graphs than with other heuristics (see Section 5.4.2.2).
- Skills identified based on transition graphs generated with FIGE increase the accumulated reward obtained by a hierarchical RL agent compared to other heuristics (see Section 5.4.2.3).
- A hierarchical RL agent which uses autonomous, graph-based skill discovery based on FIGE can outperform a monolithic learner in a multi-task problem. While monolithic learning is superior in single task problems because of the overhead of skill discovery and hierarchical learning, hierarchical approaches become the better, the more related tasks need to be solved (see Section 5.4.2.3).
- The overhead of graph-based skill discovery compared to optimal, predefined skills is constant and becomes less significant in multi-task problems (see Section 5.4.2.3).
- Representation learning based on FIGE allows accruing more reward than with other heuristics (see Section 5.5.3.1 and Section 5.5.3.2) or with standard function approximators like RBFs (see Section 5.5.3.3).
- FIGE can cope with domain stochasticity (see Section 5.4.1 and Section 5.5.3.1).

In general, the empirical results show that it makes a considerable difference how transition graphs are generated; for instance, using a grid-based discretization often had a catastrophic effect on the performance, even for low-dimensional domains.

Non-incremental graph-based skill discovery based on FIGE as outlined in Section 5.2 exhibits 4 of the 5 desirable properties for skill discovery identified in Section 3.2.3: it identifies task-independent skills (P2), decides automatically how many skills are created (P3), is suited for domains with continuous state and action[1] spaces (P4), and—as the empirical results show—is sample-efficient (P5). In summary, FIGE is a contribution that achieves subgoal S2 stated in Section 1.2, namely "to extend graph-based approaches for skill discovery to continuous domains".

FIGE is a non-incremental, batch algorithm that requires considerable amounts of computation. This is less critical when it is combined with other off-line approaches like RPI, LSPI, and non-incremental skill discovery based on graph-clustering, which are even more expensive in terms of computation. However, for making use of transition graphs in incremental methods like, e.g., OGAHC for skill discovery, it is required to develop an incremental method for graph generation that aims at similar objectives as FIGE. This issue will be addressed in Chapter 6. By this, a skill discovery approach will be obtained that exhibits all 5 desirable properties identified in Section 3.2.3.

The author would also like to note that the generative a model and the maximum likelihood formulation $G^* = \arg\max_G L_T(G)$ stated in this chapter could give rise to other methods for transition graph generation in the future: for instance a gradient descent-based or expectation maximization-based approach could yield viable alternatives to FIGE. However, this remains to future work and is not further examined in this thesis.

[1] Section 6.4.2 will show an experiment where FIGE is applied in a domain with continuous action space.

6
Lifelong Learning and Intrinsic Motivation

> "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education, one would obtain the adult brain. [...] Our hope is that there is so little mechanism in the child brain that something like it can be easily programmed. The amount of work in the education we can assume, as a first approximation, to be much the same as for the human child."
>
> Alan Turing, Computing Machinery and Intelligence, 1950, pp.456

6.1 Introduction

We have stated in Chapter 1 the goal to develop an incremental, self-motivated approach for skill acquisition which is based on identifying and exploiting the structure of a problem and which can be used in developmental and lifelong learning settings in continuous and stochastic domains. In Chapter 4, we have proposed the incremental skill discovery approach `OGAHC`, which allows discovering new skills at any time during learning (subgoal S1 of Chapter 1.2). In Chapter 5, we have proposed `FIGE` which allows identifying and exploiting the structure of continuous domains for skill discovery (subgoal S2). However, `OGAHC` is not suited for continuous domains and `FIGE` is not incremental. In this chapter, we combine the two approaches into a novel skill discovery approach which achieves both subgoals at the same time, i.e., which is incremental and suited for continuous domains.

Furthermore, we address subgoal S3, which aims at being able to perform skill acquisition in developmental or lifelong learning settings. As discussed in Chapter 1, developmental settings correspond to situations where an artificial agent—before it is confronted with tasks that are imposed on him externally—is granted a developmental period in which it can act freely without external tasks and rewards. On the one hand, such a developmental setting is well suited for skill acquisition since the agent can focus entirely on skill discovery and skill learning without having to solve external tasks.

On the other hand, there is also no external reward signal which guides the agent's behavior. Because of this, the agent needs to have an internal mechanism which governs its behavior. Such a mechanism is called an *intrinsic motivation* system. According to Barto et al. (2004), such an intrinsic motivation system should be a sophisticated system that need not be redesigned for different problems anew. In this way, an intrinsic motivation system can facilitate the acquisition of skills that can form reusable building blocks for behavioral hierarchies (Singh et al., 2010).

In this chapter, we propose computational approaches to intrinsic motivation which govern the agent's priorities in skill acquisition. This means that at any point in time the intrinsic motivation system decides if the agent focuses on skill learning, more specifically which skill out of a set of skills is learned, or on discovering novel skills which requires further exploration of the domain. In the two proposed methods, we consider intrinsic motivation to reward the agent for (a) exploring novel parts of the environment and for (b) engaging in learning skills whose predictive model exhibits large error. We define novelty with regard to a set of observed states and predict skill effects based on a learned skill model which allows predicting state transitions conditioned on the specific skill.

We present an empirical analysis of the proposed approach in two domains with continuous and high-dimensional state space and complex dynamics. We evaluate empirically to which extent the agent can benefit from reusing skills, which influence the specific skill discovery approach and the definition of intrinsic motivation have onto the agent's performance, and how the length of the agent's developmental period affects the task performance. Moreover, we present evidence that the intrinsic motivation mechanisms can identify how much time should be spent on learning specific skills.

The chapter is structured as follows: Section 6.2 provides the necessary background on lifelong learning and intrinsic motivation. Section 6.3 gives details of the main methodological contributions of this chapter. In Section 6.4, we present and discuss the results obtained in the empirical analysis. In Section 6.5, we draw a conclusion and provide an outlook.

6.2 Lifelong Learning and Intrinsic Motivation

In this section, we give a summary of lifelong learning, shaping, and intrinsic motivation and present some related work on computational intrinsic motivation systems.

6.2.1 Lifelong Learning and Shaping

Thrun (1996) suggested the notion of *lifelong learning* in the context of supervised learning for object recognition. In lifelong learning, a learner experiences a sequence of different but related tasks. Due to this relatedness, learned knowledge can be transferred across multiple learning tasks, which can allow generalizing more accurately from less training data. The concept of lifelong learning was extended to RL by, e.g., Sutton et al. (2007) and by Ring (1997) under the term "continual learning".

In RL, lifelong learning is often combined with *shaping* (Randløv and Alstrøm, 1998), which denotes a process where a trainer rewards an agent for a behavior that progresses towards a desired target behavior which solves a complex task. Thus, shaping can be seen as a training procedure for guiding the agent's learning process. Shaping was originally proposed in psychology as an experimental procedure for training animals (Skinner, 1938) and has been adopted for training of artificial systems later on (Randløv and Alstrøm, 1998). Shaping takes often the form of *progressive tasks*, where the trainer first states relatively simple tasks to the agent. Once these simple tasks can be solved by the agent, more complex tasks are presented by the trainer. The central idea here is that learning to solve complex tasks should become easier if related but simpler tasks have already been learned. This process is inspired by learning in biological systems: for instance, a human baby typically learns first how to roll, then how to crawl, then how to walk. One early success of applying shaping in the form of progressive tasks to robot learning was given by Asada et al. (1996): in this work, a goal shooting task was learned by means of Q-learning in progressively more complex tasks and "learning from easy missions".

One disadvantage of shaping via progressive tasks is that an external teacher is required which selects tasks of a specific complexity carefully by taking the current developmental state of the agent into account. This reduces the agent's autonomy.

6.2.2 Intrinsic Motivation

A different approach to lifelong learning, in which no external teacher is required, is to provide the agent with a means for *intrinsic motivation*. The term "intrinsically motivated" stems from biology and one of its first appearances was in a paper by Harlow (1950) on the manipulation behavior of rhesus monkeys. According to Baldassarre (2011) "extrinsic motivations guide learning of behaviors that directly increase [evolutionary] fitness" while "intrinsic motivations drive the acquisition of knowledge and skills that contribute to produce behaviors that increase fitness only in a later stage." Thus, similar to shaping, intrinsic motivations contribute to learning not as a learning mechanism per se, but rather as a guiding mechanism which guides learning mechanisms to acquire behaviors that increase fitness. According to Baldassarre "[intrinsic motivations] drive organisms to continue to engage in a certain activity if their competence in achieving some interesting outcomes is improving, or if their capacity to predict, abstract, or recognize percepts is not yet good or is improving...". Accordingly, learning signals produced by intrinsic motivations tend to decrease or disappear once the corresponding skill is acquired.

Computational approaches to intrinsic motivation (see Oudeyer and Kaplan (2007) for a typology) have become popular in hierarchical RL in the last decade resulting in the area of Intrinsically Motivated Reinforcement Learning (IMRL) (Barto et al., 2004). Work on intrinsic motivation in RL, however, dates back to the early 1990s (Schmidhuber, 1991). IMRL often employs a *developmental setting* (see, e.g., Stout and Barto (2010) and Schembri et al. (2007)), which differs from the usual RL setting where the objective is to maximize the accumulated external reward. In the developmental

setting, the agent is given a developmental period, which can be considered as its "childhood", in which no external reward is given to the agent. This allows the agent to explore its environment freely without having to maximize the accumulated reward, i.e., without exploitation. On the other hand, the agent is not guided by external reward but needs to have a means for intrinsic motivation. The objective in the developmental setting is to learn skills which allow learning high-quality policies quickly in tasks that are later on imposed onto the agent. Thus, the objective can be seen as a kind of optimal exploration for skill learning (Şimşek and Barto, 2006), in contrast to finding the optimal balance between exploration and exploitation typical for non-developmental RL.

6.2.3 Related Work

Different mechanisms for intrinsic motivation have been proposed. A complete review is beyond the scope of this chapter, we discuss a selected subset of methods and refer to Oudeyer et al. (2007) for a review.

Barto et al. (2004) and Singh et al. (2004) investigate how a hierarchically organized collection of reusable skills can be acquired based on intrinsic reward. Their notion of intrinsic reward is based on the novelty response of dopamine neurons. More precisely, the intrinsic reward for a *salient event* is proportional to the error of predicting this salient event based on a learned skill model for this event. This skill model is not only a passive model of the environment but it is also dependent on the agent's action preferences. As a result of the intrinsic reward, once the agent encounters an unpredicted salient event, it is driven to attempt to achieve this event until it has learned to predict it satisfactorily. Note that the assumption that the agent has a hardwired notion of interesting or salient events limits the autonomy of the agent since it effectively circumvents the necessity of skill discovery. The approach is evaluated empirically in the discrete playroom domain. The results show that an agent employed with skills learned using intrinsic motivation can learn a multitude of externally defined tasks more easily than an agent learning task solutions from scratch.

An extension of this work to a robotic scenario is given by Soni and Singh (2006): the authors present a study of novelty-based intrinsic motivation on the Sony AIBO robot. A set of salient events is predefined and one option is created for achieving each of the salient events. For each of these options, a model of state transition probabilities is learned and unexpected state transitions are rewarded since they are considerer to be novel.

Şimşek and Barto (2006) consider the optimal exploration problem, i.e., how an agent can learn efficiently how to accumulate high reward in the future without having to collect high reward during the learning process. The authors propose a method which is based on learning a policy for a derived MDP. In this derived MDP, the agent has to maximize an intrinsic reward instead of an external one; however, the intrinsic reward is based on the external reward. The authors propose that this optimal exploration perspective could be adopted for skill acquisition: once a new skill is discovered, the agent may focus on learning a satisfactory skill policy and may be indifferent to external

reward during this "developmental" phase. The authors present empirical results which suggest that the proposed approach can acquire skills which improve performance in the target task compared to using only primitive actions. This advantage is the larger, the longer the developmental phase lasts.

Singh et al. (2010) investigate the interplay of learning and evolution. Their hypothesis is that in nature, intrinsic reward mechanisms are selected by evolution based on their utility for speeding up learning in a magnitude of environments. The authors conduct a computational experiment in which they adopt an evolutionary perspective. In this perspective, reward functions are evaluated according to the expected fitness of learning agents using them in a range of environments given some explicit fitness function. By searching over a large set of reward functions, they find reward functions that allow agents to acquire a considerably larger evolutionary fitness than when using reward mechanisms that are directly coupled to evolutionary success. This is possible because these reward functions allow guiding exploration in a way that is reasonable for a large range of environments; the selected reward functions must thus capture regularities across environments. According to Singh et al., a major function of intrinsic rewards is thus to compensate for agent limitations, such as short agent lifetimes or non-Markovian nature of environments, by allowing a transfer of an explorative bias by means of the reward function.

Oudeyer et al. (2007) propose an intrinsic motivation system that encourages a robot to explore situations in which its current *learning progress* is maximized. More specifically, the robot obtains a positive intrinsic reward for situations in which the error rate of internal predictive models decreases and a negative one for situations in which it increases. By this, the robot focuses on exploring situations whose complexity matches its current stage of development, i.e., situations which are neither too complex (too unpredictable) nor too simple (too predictable).

Hester and Stone (2012) propose a model-based approach for a developing, curious agent called TEXPLORE-VANIR. This approach uses two kinds of intrinsic reward that are derived from the learned model. The first one rewards the agent for exploring parts of the environment for which the variance in the model's prediction is large while the second one rewards the agent for exploring parts of the environment that are *novel* to the agent. The authors show empirically that these intrinsic rewards are helpful for an agent in a developmental setting. Furthermore, the intrinsic rewards also improve the performance of an agent faced with an external task from the very beginning by providing a reasonable explorative bias.

Stout and Barto (2010) propose "competence progress motivation", which generates intrinsic rewards based on the skill competence progress, i.e., how strongly the agent's competence to achieve self-determined goals progresses. The authors show on a simple problem that the approach is able to focus learning efforts onto skills that are neither too simple not too difficult at the moment. While the authors predefine the set of skills that shall be learned, they note that "identifying what skills should be learned is a very important problem and one that a complete motivational system would address". This problem is addressed in this chapter.

Note that intrinsic motivations need not be the only source of motivation in a biologically-inspired robotic control architecture such as the one shown in Figure 1.1; rather, *homeostatic* need regulation and prediction of fitness-enhancing visceral-body changes, which are considered to be internal but extrinsic motivations (Baldassarre, 2011), should be taken into account as well. However, since we focus on the "decision" layer of the architecture, we do not consider these kinds of motivations in detail here.

6.3 METHODS

In Section 6.1, we have motivated the need for lifelong learning agents that can be used in developmental settings. In this section, we present an agent architecture for an IMRL agent that can be used in such developmental settings and is based on skill discovery and intrinsic motivation. Thereupon, we propose a new method for incremental skill discovery based on `FIGE` and `OGAHC` that is suited for developmental settings and continuous state and action spaces. Moreover, we define two different intrinsic motivation mechanisms.

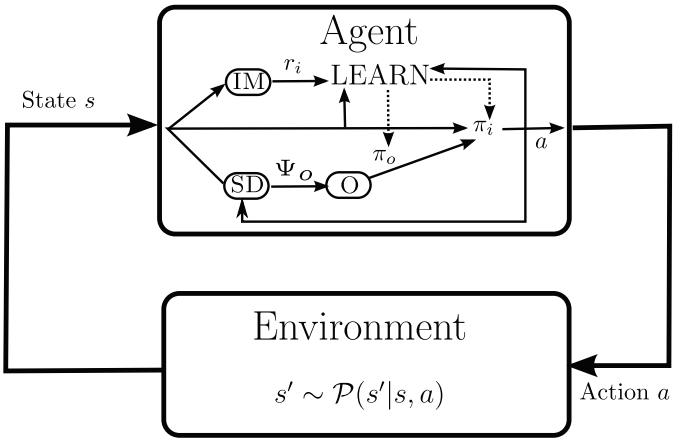
6.3.1 Agent Architecture

We consider an agent situated in an environment with state space $\mathcal{S}$ and action space $\mathcal{A}$. We are particularly interested in problems where the state and/or the action space are continuous, more specifically where $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ and/or $\mathcal{A} \subseteq \mathbb{R}^{n_a}$. We assume that the state transitions (the effects of executing an action in a state) have the Markov property. During its lifetime, the agent may be faced with different tasks in this environment; we assume that each task $\mathcal{T}_j$ is specified by a reward function $(R^a_{ss'})_j = R_j$ and the agent needs to maximize a long-term notion of this reward. Note that each task thus corresponds to an MDP[1] $\mathcal{M}_j = (\mathcal{S}, \mathcal{A}, P^a_{ss'}, R_j)$, where all tasks share $\mathcal{S}$, $\mathcal{A}$, and $P^a_{ss'}$.

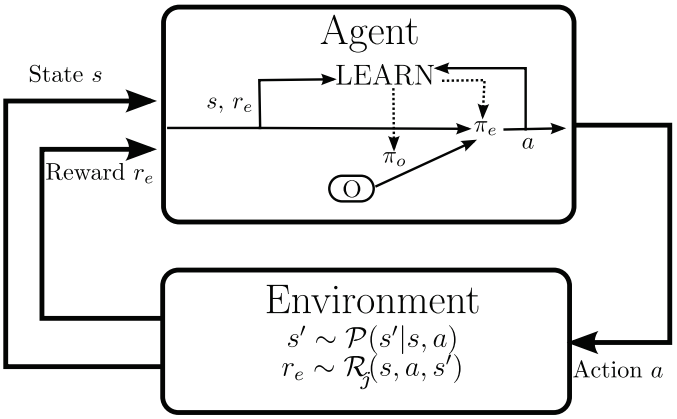
We adopt the developmental setting of IMRL (see Section 6.2), i.e., we assume that the agent is granted a developmental period before it is faced with an external task. The agent-environment interaction during the developmental period can be modeled as an MDP without reward $\mathcal{M}_{\setminus R} = (\mathcal{S}, \mathcal{A}, P^a_{ss'})$. Thus, we implicitly assume that the developmental period takes place in the same environment where the agent has to solve tasks later on, i.e., we assume $\mathcal{S}$, $\mathcal{A}$, and $P^a_{ss'}$ to be identical. While no external objective is imposed on the agent, the agent should use the developmental period nevertheless for learning a repertoire of skills O that can later on help in solving tasks $\mathcal{T}_j$. Furthermore, we do not provide the agent with a set of subgoals or salient events but require the agent to identify these on its own.

For this, two questions need to be addressed: (a) how are useful and task-independent skills identified autonomously? and (b) how does the agent handle the implicit trade-off of skill acquisition between skill discovery and skill learning when no external reward is available? We address these questions in Section 6.3.2 and Section 6.3.3 respectively.

[1] Note that we skip S_0 and S_T here in the definition of the MDP for notational brevity, see Section 2.2.2.



(a) *Agent architecture employed during the developmental period.* No external reward is provided but the motivational system IM creates an intrinsic reward r_i. In parallel, new skill prototypes Ψ_o are identified using the skill discovery module SD and added to the skill pool O. Compositional learning is used to learn a hierarchical policy π_i which select skills such that the intrinsic reward is maximized. Both π_i and the policy π_o of the active skill o are learned concurrently.



(b) *Agent architecture for learning to solve external tasks* $\mathcal{T}_j$. A hierarchical policy π_e is learned based on the fixed set of skills O, which have been discovered during the developmental period, such that the external reward r_e is maximized. The policy π_o of the active skill is improved concurrently based on R_o but not r_e.

Figure 6.1 – *Agent architectures for developmental period and for task learning.*

For now, we assume that two modules for intrinsic motivation (IM) and skill discovery (SD) exist where IM generates an intrinsic reward signal r_i which the agent uses in place of external reward and SD identifies new skill prototypes $\Psi_o = (I_o, \beta_o, R_o)$ which are added to the skill repertoire O and whose policies π_o are learned later on by the agent using option learning such that R_o is maximized. It is important to note that the pseudo reward function R_o is different from the intrinsic reward r_i: r_i governs the high-level behavior of the agent, which is encoded in π_i, during the developmental period while R_o encodes the objective of a single skill (typically reaching a bottleneck) and should

thus be independent of both the intrinsic and the external reward. The agent's internal architecture during its developmental period is depicted in Figure 6.1a.

Once an external task $\mathcal{T}_j$ is imposed onto the agent, the intrinsic reward and the skill discovery modules are disabled, and the agent learns a hierarchical policy π_e over the set of discovered skills O that maximizes the external reward r_e (see Figure 6.1b). Note that the agent continues to learn option policies π_o based on experience collected; however, the external reward r_e is not taken into account in skill learning such that options remain task-independent and focus on maximizing the pseudo reward R_o.

6.3.2 Incremental Graph-Based Skill Discovery

A skill discovery method which can be used in the outlined architecture needs to exhibit the properties P1-P4 identified in Section 3.2.3: it needs to be incremental (P1), it needs to be applicable in developmental settings, i.e., it must not require that an external reward signal or task exists (P2), it should decide automatically how many skills are identified (P3), and it needs to be suited for continuous domains (P4).

In this section, we propose `IFIGE`, an incremental extension of `FIGE` (cf. Section 5.3.2), which is combined with an extension of `OGAHC` (cf. Section 4.2.3.2) to continuous domains. This combination exhibits all of the properties given above. The key idea of the approach is that a transition graph, which captures the domain's dynamics, is learned incrementally from experience using `IFIGE` and that the generated graphs are clustered into densely connected subgraphs using `OGAHC`. These clusters correspond to subareas of the domain's state space and the connections between these subparts form bottlenecks of the domain.

6.3.2.1 Incremental Transition Graph Estimation in Continuous Domains

The main drawbacks of `FIGE` (see Section 5.3.2) are that v_{num}, the number of nodes of the transition graph, needs to be pre-specified and that `FIGE` is a batch algorithm and thus not suited for incremental skill discovery. We present now Incremental `FIGE` (`IFIGE`) which is an extension of `FIGE` that does not suffer from these drawbacks. `IFIGE` updates the graph's node positions after every experienced transition. Moreover, `IFIGE` stores for every graph node v a set of exemplar states $S[v] = \{s_i \mid i = 1,\dots,n_v\}$ and exemplar transitions $T[v] = \{(s_i, a_i, s_i') \mid i = 1,\dots,n_v\}$, with all s_i being "similar" to v and n_v being set typically to 25. Note that n_v can be seen as a memory capacity; larger values improve performance but lead to increased memory consumption and runtime of the algorithm.

`IFIGE` (see Algorithm 6.1) starts with a single graph node $V = \{s_0\}$ (line 2) and $S[s_0] = T[s_0] = \emptyset$ (line 3), where s_0 is the start state. For any encountered transition (s, a, s'), the most similar graph node $v = \mathrm{NN}_V(s)$, i.e., the nearest neighbor of s in V, is determined (line 6), s is added to the set of state exemplars $S[v]$, and (s, a, s') to $T[v]$. If the size of $S[v]$ or $T[v]$ exceeds n_v, old exemplars are deleted (line 7 and 8). Afterwards, the position of vertex v is updated according to the standard `FIGE`

Algorithm 6.1 Incremental Force-Based Iterative Graph Estimation (IFIGE)

1: **Input:** start state s, maximal node diameter ζ, maximal number of exemplars n_v
2: $V = \{s\}$ # Start with single graph node at start state
3: $S[s] = T[s] = \emptyset$ # No exemplars initially
4: **loop**# Let the agent act indefinitely
5: $(s,a,s') = \text{ACT(s)}$ # Choose action from $\pi(s,a)$ and successor state according to $P^a_{ss'}$
6: $v = \text{NN}_V(s)$ # Determine graph node responsible for state s
7: $S[v] = \text{ADDSTATE}(S[v],s)$ # Add state s to $S[v]$, remove old exemplar if $|S[v]| > n_v$
8: $T[v] = \text{ADDTRANS}(T[v],(s,a,s'))$ # Add transition, remove old exemplar if $|T[v]| > n_v$
9: $F_S = \text{MEAN}(S[v]) - v$ # Sample representation force on node v
10: $F_G = \{\text{NN}_V(s') - s' + s \mid \exists (s,a,s') \in T[v]\}$ # Forward graph consistency force on node v
11: $v = v + \alpha \cdot 0.5(F_S + F_G)$ # Update position of node v
12: $\text{REASSIGN}(S[v],T[v],V)$ # Check exemplars in $S[v]$, $T[v]$ for a better representative in V
13: **if** $\text{DIAMETER}(S[v]) > \zeta$ **then** # If node v covers too larger area of state space
14: $v_1, v_2 = \text{kMEANS}(S[v],2)$ # Choose two new representatives for state set $S[v]$
15: $V = [V \cup \{v_1,v_2\}] \setminus \{v\}$ # Remove old node v, add new nodes v_1 and v_2
16: $\text{REASSIGN}(S[v],T[v],\{v_1,v_2\})$ # Assign exemplars in $S[v]$ and $T[v]$ to v_1 and v_2
17: **end if**
18: $s' = s$ # Continue in next state
19: **end loop**# Note: graph edges and weights are created on demand

forces[1] for $T = T[v]$ and $S^V[v] = S[v]$ (line 9-11). This changes the position of v; thus, IFIGE checks afterwards for all state exemplars in $S[v]$ and transition exemplars in $T[v]$ whether any other node in V would be a better representative and moves the exemplars if required (line 12). Afterwards, IFIGE checks whether v is responsible for a too large area of the state space by computing the distance of the farthest pair in $S[v]$. If this distance is above a threshold ζ (line 13), v is removed from V and two new nodes v_1 and v_2 are added to V (line 15). The nodes v_1 and v_2 are chosen as the cluster centers of a k-means clustering of $S[v]$ for $k = 2$ (line 14). The sets $S[v]$ and $T[v]$ are split into two subsets accordingly (line 16). Splitting nodes ensures that the number of graph nodes grows with the size of the state space explored by the agent.

The computational complexity of a single iteration of IFIGE (line 5-16) is the following: line 6 requires $\mathcal{O}(|V|n_s)$, line 7 and 8 require $\mathcal{O}(n_v)$, line 9 requires $\mathcal{O}(n_v n_s)$, line 10 and 12 require $\mathcal{O}(n_v|V|n_s)$, line 11 requires $\mathcal{O}(n_s)$, line 13 requires $\mathcal{O}(n_v^2 n_s)$, line 14 and 16 require $\mathcal{O}(n_v n_s)$, and line 15 requires $\mathcal{O}(1)$. In summary and assuming $n_v < |V|$, IFIGE requires in the worst case $\mathcal{O}(n_v|V|n_s)$.

For generating the graph based on the determined node positions V, IFIGE adds for all graph nodes v and any transition $(s,a,s') \in T[v]$ an edge between v and the node $v' = \text{NN}_{V\setminus\{v\}}(s')$ for action a. Since this requires at least $\Omega(|V|^2)$ computations, the graph edges are not updated after every step but only when the graph is actually required externally (for instance for graph-based skill discovery). In domains with discrete action

[1] Note that only the forward part $T^{\rightarrow}(v)$ of the graph consistency force for v is considered here since the backward part $T^{\leftarrow}(v)$ could depend on all exemplars of all other nodes. This is prohibitive since the worst-case complexity of a single update would be $\mathcal{O}(|V|^2 n_v n_s)$ since the nearest graph node for all $\mathcal{O}(|V|n_v)$ exemplars would have to be computed. We have not observed any considerable effects of this simplification onto the resulting graph.

spaces, the off-policy edge weights as in Algorithm 5.1 are used while in domains with continuous action spaces the on-policy edge weights are preferable since they do not require that the sampling policy is estimated, which would be difficult in continuous action spaces.

6.3.2.2 Extension of OGAHC to Continuous Domains

Based on the transition graph constructed incrementally with IFIGE, we can identify task-independent and thus reusable skills using OGAHC (see Section 4.2.3.2). In this section, we discuss how OGAHC can be extended to domains with continuous state and action spaces.

The main hindrance for applying OGAHC in domains with continuous state space is the constraint "two nodes that have been assigned to different clusters in partition $\mathcal{P}_{t_1}$ at time t_1 must not be assigned to the same cluster in a subsequent partition $\mathcal{P}_{t_2}$ for any time $t_2 > t_1$", cf. Section 4.2.3.2. This assumes that graph nodes do not change over time, which is not the case when OGAHC is applied on top of IFIGE where graph nodes might change their position or might even be split into two new nodes. This property of IFIGE needs to be taken into account when defining the constraints.

For this, the current partition is adapted to changes in the graph prior to the invocation of OGAHC as follows: let $P^*(V)$ be the partition of the graph nodes V of the last invocation of OGAHC and V' be the current node positions. We extend $P^*(V)$ to a (pre-)partition P_{pre} of V' by assigning nodes $v'_a, v'_b \in V'$ to the same cluster if the nearest neighbors[1] $\mathrm{NN}_V(v'_a)$ and $\mathrm{NN}_V(v'_b)$ are in the same cluster in $P^*(V)$. Now, OGAHC can be invoked with the usual constraints that nodes which are in different clusters in $P_{pre}(V')$ must be in different clusters in $P^*(V')$.

Some of the nodes $v' \in V'$ may be in areas of the state space that have not been explored before and were thus not covered by V. Accordingly, constraints based on $P^*(V)$ should not be extended to these nodes. For this, we identify all nodes $v' \in V'$, whose nearest neighbor $\mathrm{NN}_V(v')$ is further away than $2d_{nn}$, where d_{nn} is the average distance of two connected nodes in V. For these nodes, the constraint is relaxed such that they can be assigned to any cluster in $P^*(V')$. This corresponds to a situation where the agent has visited a particular area of the state space for the first time and has thus no prior knowledge from previous invocations of OGAHC about the bottlenecks in this novel area.

Since graph clustering and linkage criteria are defined solely on the graph's edge weights, no adaptations of these parts are required in domains with continuous action spaces. Graph smoothing as defined in Section 4.2.3.2, however, requires that statistics of node-action pairs like n_v^a are determined. This becomes impractical for continuous action spaces because of the infinite number of actions. Different ways of solving this problem are conceivable, for instance histograms could be used instead of n_v^a and graph smoothing be performed based on the number of samples per action bin. We leave

[1] Note that the robustness of this approach could be increased by taking not only the nearest neighbor but the k nearest neighbors into account.

this to future work, however, and do not perform graph smoothing in domains with continuous action spaces in this work.

Based on the partitions obtained by applying OGAHC to the transition graphs estimated with IFIGE, bottlenecks of the graph and thus also of the domain can be identified. Based on the identified bottlenecks, skill prototypes Ψ_o can be defined as discussed in Section 5.3.3.

6.3.2.3 Illustration

We present an illustration of the proposed skill discovery approach in Figure 6.2. For this, we use the 1D Multi-Valley domain. As the 2D Multi-Valley domain (see Section 5.4.2.1), this domain is an extension of the mountain car domain, which consists of two valleys. State and action space and dynamics are essentially the same as in mountain car with the one exception that there are two valleys instead of one. The hill which separates the two valleys is a typical bottleneck of the domain as the agent would cross this hill very unlikely under a random policy. This can also be seen in the plot of the domain's dynamics in Figure 6.2. Because of this and the domain's two-dimensional state space, the domain is well suited for illustration of bottleneck-based skill discovery in continuous domains.

The figure shows the generated transition graphs and their clustering after 5000, 10000, 25000, and 50000 steps. After 5000 steps, the agent has visited both valleys but only in "high-energy" states, i.e., states with either high velocity or positions close to the top of the hills. Because of this, the separating hill has not yet been identified as a bottleneck as it can be traversed easily in these high-energy states. After 10000 steps, the agent has continued exploring the western valley and the graph generated by IFIGE has "grown" into these novel area. However, because of the many virtual transitions which are added by OGAHC, no bottleneck is identified at this point in time. After 25000 steps, the agent has explored most areas of the state space and the transition graph is grown into areas corresponding to low-energy states. Accordingly, OGAHC identifies a domain decomposition which captures the bottleneck on the top of the hill correctly. After 50000 steps, the generated graph covers most of the domain and encodes its dynamics well; however, no qualitative changes in the domain's decomposition occur as there is only one bottleneck in this domain which had already been identified.

6.3.3 Intrinsic Motivation

In the context of this chapter, intrinsic motivation refers to the process of mapping a transition from state s under option o to successor state s' onto an intrinsic reward r_i. We investigate two different intrinsic motivation mechanisms, one based on the novelty of a state under a skill and one based on the prediction error of a learned skill model.

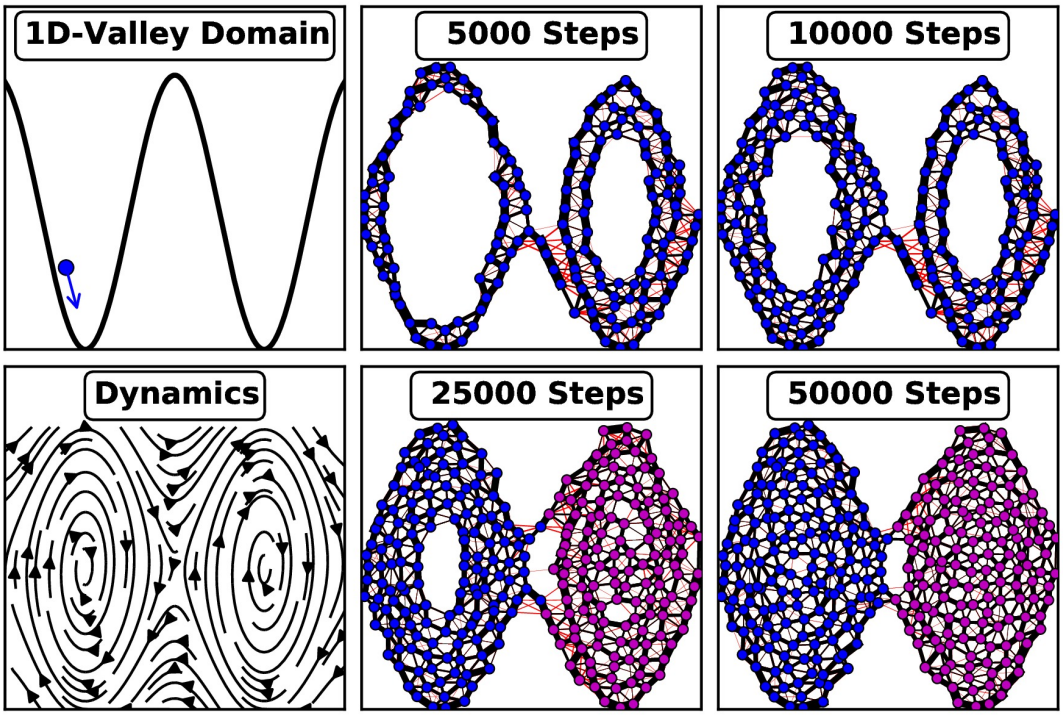


Figure 6.2 – *Illustration of the proposed skill discovery approach in the 1D Multi-Valley.* The x-axis shows the car's position and the y-axis its velocity. The upper left plot shows the two valleys of the domain and the lower left plot the dynamics $P^a_{ss'}$ of the domain for a =None. The four other plots show the transition graphs generated by `IFIGE` after 5000, 10000, 25000, and 50000 steps under a random policy. The node colors correspond to the cluster labels of the node partition determined by `OGAHC` and the red edges correspond to virtual edges added by graph-smoothing in `OGAHC`.

For the *novelty-based motivation* criterion, the agent stores for each option o the states it has encountered under this option so far in the set S_o.[1] When transitioning to state s' under option o, the intrinsic reward is computed via

$$r_i = -\sum_{j \in \mathrm{NN}^{10}_{S_o}(s')} \exp\left(-\frac{||s' - s_j||_2^2}{b^2}\right), \tag{6.1}$$

where $\mathrm{NN}^{10}_{S_o}(s')$ denotes the indices of the 10-nearest neighbors of s' in S_o and b is a domain-dependent scale parameter. Thus, the intrinsic reward is upper-bounded by 0 with values close to 0 if the 10 nearest neighbor are very different (large euclidean distance) from s' and very small values when s' is similar to several states in S_o. Thus, the novelty criterion discourages to execute options in regions of the state space where

[1] In order to keep the size of S_o limited, we remove states from S_o once $|S_o| > 2500$. The heuristic for selecting the state that is removed is to remove one of the states of the (approximate) closest state pair in S_o. This results in covering the effective state space of the problem approximately uniform.

this option has been executed already several times. This mechanism is similar to the mechanism proposed by Hester and Stone (2012); however, in contrast to their work, it is also suited for domains with continuous state spaces.

For the *prediction-error motivation* criterion, the agent learns for each option a model $\hat{P}_o$ that predicts the successor state of states s when following option o. The intrinsic reward is determined based on the error of the model's prediction via

$$r_i = -1 + \tanh(\sigma ||s' - \hat{P}_o(s)||_2^2), \tag{6.2}$$

where σ is a domain-dependent scale parameter. The intrinsic reward r_i is large (close to 0) when the difference of predicted successor $\hat{P}_o(s)$ and actual successor s' is large. The intrinsic reward becomes small (close to -1) when the model correctly predicts the effect of executing option o in state s. Thus, the prediction error criterion encourages to execute options whose effects are unknown or unpredictable in the current area of the state space. Note that in contrast to the novelty criterion, for the prediction error criterion the intrinsic reward in a state depends on the option's policy.

The option model $\hat{P}_o$ stores internally a set $T_o = \{(s_j, s'_j)\}$ of transitions encountered under option o. The model's prediction is based on 10-nearest neighbors regression:

$$P_o(s) = s + \frac{1}{10} \sum_{j \in \mathrm{NN}_{T_o}^{10}(s)} (s'_j - s_j), \tag{6.3}$$

where $\mathrm{NN}_{T_o}^{10}(s)$ denotes the indices of the 10-nearest neighbors of s in the start states in T_o. If the size of T_o exceeds a threshold (in the experiments 2500) and a transition from s to s' is added, the oldest transition among $\mathrm{NN}_{T_o}^{10}(s)$ is removed. This is required to keep the memory consumption limited and, more importantly, to track the non-stationarity in the target function that is induced by learning the option o concurrently and thus changing o's policy.

6.4 RESULTS

In this section, we present an empirical evaluation of the proposed methods in two continuous and challenging RL domains. We evaluate both the behavior of the agent during the developmental period and its performance in external tasks.

6.4.1 2D Multi-Valley

For similar reasons as in the previous chapter, we use the 2D Multi-Valley environment (see Section 5.4.2.1) as first RL benchmark domain: the domain consists of multiple tasks and is thus a good choice for analyzing if the skills discovered during the developmental period allow transferring procedural knowledge across tasks and in which situations this transfer becomes important.

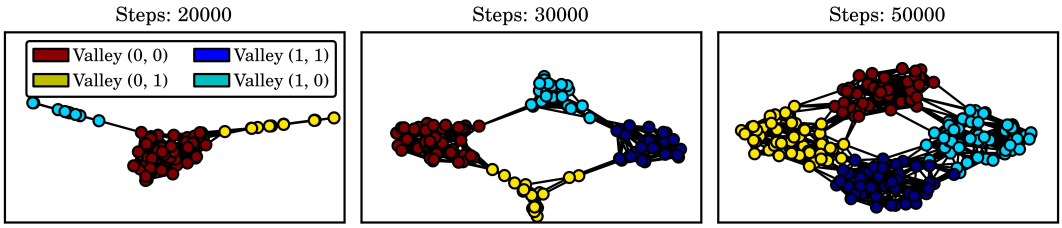


Figure 6.3 – *Visualization of generated transition graphs in 2D Multi-Valley.* Shown is a two-dimensional embedding (determined using Isomap) of the learned transition graphs. The densely connected subgraphs correspond to the four valleys.

6.4.1.1 Developmental Period

During its developmental period, the agent can explore the domain freely while engaging in skill discovery and following its intrinsic motivations. Initially, the agent has only a single option o_e in its skill pool O, which can be invoked in any state of the environment, i.e., $I_{o_e} = \mathcal{S}$, and terminates with probability $\beta_{o_e}(s) = 0.05$. This option can be considered to be the agent's exploration option, which can always be invoked if the agent prefers to explore the environment over learning a specific skill. We set the greediness of `IFIGE` to $\alpha_i = 0.25$ and the maximal node diameter to $\zeta = 0.3$. For `OGAHC`, we set the maximal linkage to $\psi = -0.0375$ and performed skill discovery every 5000 steps.

Each option's value function has been represented by a CMAC function approximator consisting of 10 independent tilings with $7^2 \cdot 5^2$ tiles, where the higher resolutions have been used for the x and y dimensions. The penalty r_p of the options' pseudo-reward functions R_o has been set to $r_p = -1000$. An agent obtains this penalty if an option terminates unsuccessfully, i.e., leaves its initiation set I_o without reaching its goal cluster (see Section 5.3.3). Value functions have been initialized to -100. For learning the higher-level policy π_i, a lower resolution of $5^2 \cdot 3^2$ tiles has been used and the value functions have been initialized to 0. The discounting factor has been set to $\gamma = 0.99$ and all policies were ε-greedy with $\varepsilon = 0.01$. The value functions were learned using Q-Learning and updated only for currently active options with a learning rate of 1. The scale-parameters of the intrinsic motivation mechanisms have been set to $b = 0.1$ (novelty) and $\sigma = 10^4$ (prediction error). All parameters have been chosen based on preliminary investigations.

Figure 6.3 shows the transition graphs generated by `IFIGE` after 20000, 30000, and 50000 developmental steps. The two-dimensional embeddings of the graphs have been determined using Isomap (Tenenbaum et al., 2000). The four valleys of the domain clearly correspond to four densely connected subgraphs of the transition graph. The figure also shows that it would be difficult to determine a single point in time at which skill discovery should be performed: for instance, are the valleys $(0,1)$ and $(1,0)$ explored sufficiently after 30000 steps to perform graph clustering? Since skill discovery with `OGAHC` is incremental, i.e., can be performed several times during learning, this choice need not be made.

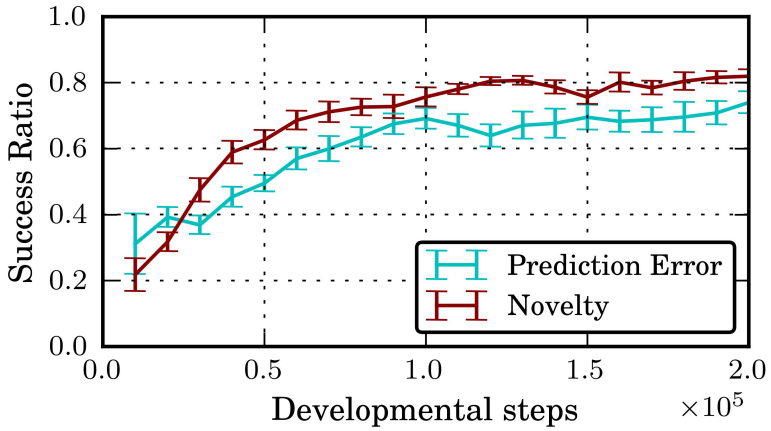


Figure 6.4 – *Success ratio of learned skills during developmental period.* Shown are mean and standard error of the mean averaged over 10 independent runs.

Figure 6.4 shows the success ratio, i.e., how often a skill reaches its goal cluster, of the skills discovered during the developmental period. Initially, skills are unlikely to reach their goal area, with success ratios of approximately 0.25. Under both intrinsic motivation systems, the agent invests time in learning skill policies and the success ratio increases to 0.7 for the prediction error and 0.8 for the novelty criterion after approximately 10^5 steps of development. Note that success ratios of 1.0 are not possible since for some states in $s \in I_o$, there is no way of reaching the option's goal area without leaving the initiation set, e.g., when the agent is moving with high velocity in the direction of the wrong neighbor valley. A possible explanation for the different performance under the two motivational systems is given below.

Figure 6.5 shows the ratio of selecting the option o_e ("Exploration") or any of the other, discovered options in O ("Skill Learning") under the policy π_i for different intrinsic motivations. Initially, no skills have been discovered and the agent thus has to explore. Once the first skills have been discovered, the agent focuses onto learning these skills. Over time, as the skill policies converge, a better predictive model for these skills can be learned. Similarly, the more time is spend on learning a skill, the less novel states are encountered under this skill. Accordingly, both intrinsic motivation mechanisms reduce the ratio of skill learning and focus on exploration again in order to discover new skills. Note that at this point in time, there are no further skills to be discovered in this domain but this is unknown to the agent.

In general, the prediction error-based motivation chooses the exploration option more often and reduces skill learning more abruptly than the novelty criterion. This can be explained by the fact that the exploration policy changes more strongly over time and it is thus harder to learn a model of this option. Once the policies of the other skills have settled, they are chosen only rarely. However, the results in Figure 6.4 suggest that this happens too early as the final "fine-tuning" of the skill policies is not finished and the success ratio is smaller than for the novelty criterion. Thus, the results indicate that using the prediction error for intrinsic motivation can be detrimental in situations

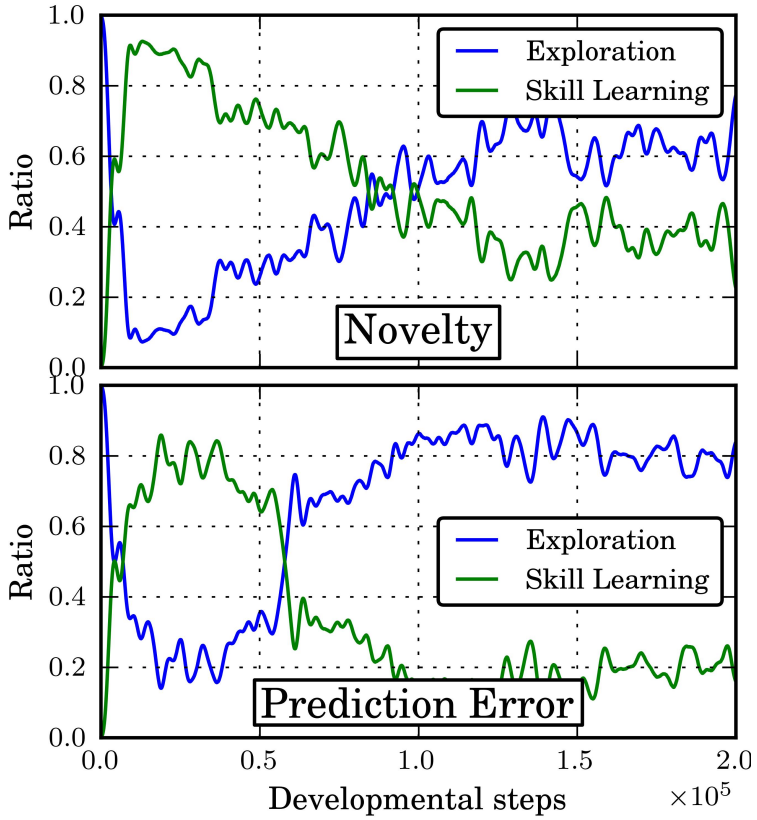


Figure 6.5 – *Ratio of skill learning and exploration during developmental period.* A ratio of 0.8 for skill learning means that an agent performs skill learning 80% of the time and only in 20% of the cases it explores. Shown is the mean over 10 independent runs.

where different option policies explore to different degrees since the prediction error criterion will favor the options with stronger exploration. Thus, it is recommended to base motivation on criteria like novelty or on the *change* of prediction error rather than on the error itself.

6.4.1.2 Task Performance

In its "adulthood", the agent is faced with the multi-task scenario discussed in Section 5.4.2.1: in each episode, the agent has to solve one out of twelve tasks. Each task is associated with a combination of two distinct valleys; e.g., in task $(0,1)$ the agent starts in the floor of valley 0 and has to navigate to the floor of valley 1 and reduce its velocity such that $\left|\left|(v_x, v_y)\right|\right|_2 \leq 0.03$. In each time step, the agent receives an external reward of $r_e = -1$. Once a task is solved, the next episode starts with the car remaining at its current position and one of the tasks that starts in this valley is drawn at random. Episodes are interrupted after 10^4 steps without solving the task and a new task is chosen at random. The current task is communicated as an additional state space dimension to the agent. The agent uses this task information and the reward r_e

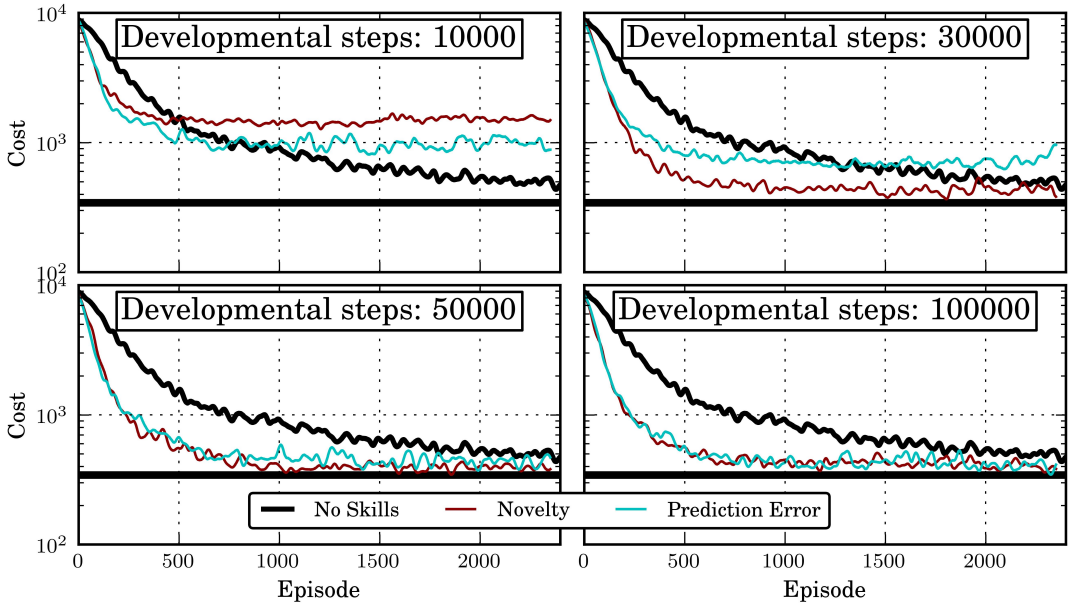


Figure 6.6 – *Performance of IMRL agent in 2D Multi-Valley.* Shown is cost (negative return $R^{25}_{\text{mwa}}(t)$) of the agent in the 12 tasks 2D Multi-Valley domain for different intrinsic motivation systems and different lengths of the developmental period. "No Skills" shows the performance of a monolithic agent that does not learn skills and has no developmental period. The horizontal black line shows the average cost of the policy learned by the monolithic agent after 5000 episodes. Shown is the mean over 10 independent runs that have been smoothened by a moving window average with window size 51.

for learning the task policy π_e but ignores these information when improving π_o such that skills remain reusable in different tasks. The exploration option o_e used in the developmental period was removed from the skill set O such that the agent can only choose among self-discovered skills.

Figure 6.6 shows the results for different intrinsic motivation mechanisms and different lengths of the developmental period. As baseline, "No Skills" shows the performance of an agent that learns a monolithic policy for each task separately (cf. Section 3.4.2). For a very short developmental period of 10000 steps, the hierarchical agent, which uses skills learned in the developmental period, learns initially faster than the monolithic agent. However, it converges to considerably worse policies. This is probably due to the fact that not all relevant skills have been discovered in the short developmental period. This is an example of the effect discussed by Jong et al. (2008) that an incomplete set of skills might have a detrimental effect on an agent's performance because their implicit explorative bias breaks the symmetry of random exploration (see Section 3.2.1).

For 30000 developmental steps, the skills acquired under the novelty motivation allow achieving close-to-optimal performance already while the ones from the prediction-error motivation do not. This corresponds to the different qualities of the learned skills under the two motivation systems (see Figure 6.4). For 50000 or more developmental

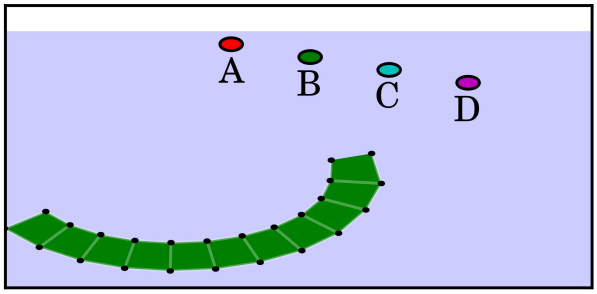


Figure 6.7 – *Visualization of the Octopus arm task.* The circles represent target objects used in different tasks which yield an external reward when touched.

steps, the performance of the hierarchical agent approaches the optimal performance considerably faster than the monolithic agent, irrespective of the intrinsic motivation system used. This is interesting since after 50000 steps, the learned skills are far from optimal (see Figure 6.4). Apparently, also skills with sub-optimal policies can help the agent considerably. This reinforces results obtained by Kirchner (1999), who showed that higher-level behavior can benefit from suboptimal lower-level skills in a fixed behavioral hierarchy. It should also be noted that even though a close-to-optimal performance is reached relatively fast, the performance remains slightly below the optimum which is reached by the monolithic agent after 5000 episodes. This is probably due to the (temporal) abstraction introduced by the skills which, on the one hand, helps the agent in learning faster but, on the other hand, also reduces the class of representable policies.

6.4.2 Octopus Arm

The second domain is a modified version of the Octopus arm domain which was introduced in Section 5.5.3.3. The base of the arm is restricted and cannot be actuated directly. The agent may control the arm in the following way: elongating or contracting the entire arm, bending the first half of the arm in either of the two directions, and bending the second half of the arm in either of the two directions. In each time step, the agent can set the elongation and the bending of the first and second half of the arm to an arbitrary value in $[-1,1]$, resulting in 3 continuous action dimensions. The agent observes the positions x_i, y_i and velocities $\dot{x}_i$, $\dot{y}_i$ of 24 selected parts of its arm (denoted by small black dots in Figure 6.7) and the angle and angular velocity of the arm's base. Thus, the state space is continuous and consists of 98 dimensions. Because of the high-dimensional state space and the complex biomechanically realistic dynamics of the domain, the octopus arm problem is a challenging benchmark task for RL.

6.4.2.1 Developmental Period

Similar to the developmental period in the 2D multi-valley domain, the agent can explore the domain freely while engaging in skill discovery and following its intrinsic

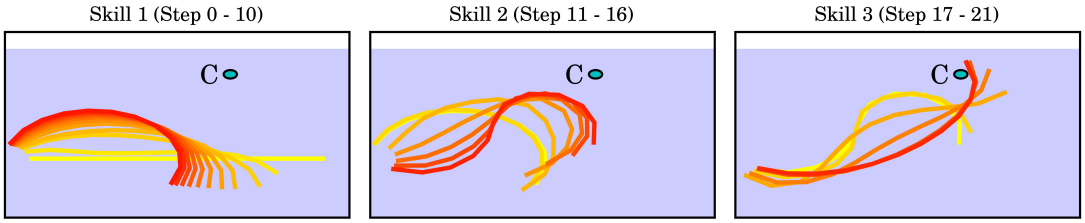


Figure 6.8 – *Example trajectory of the Octopus arm controlled by the IMRL agent.* The trajectory corresponds to a sequence of three skills. Yellowly colored arms correspond to states at the beginning of skill execution while redly colored arms correspond to states at the end of skill execution.

motivations. However, the basis for skill discovery is not to identify bottlenecks (there are no bottlenecks in this domain) but to cluster the transition graph into regions which correspond to similar qualitative states. Thus, a different linkage criterion l_G has been used: for two subgraphs A and B of the transition graph G, the linkage is set to $l_G(A,B) = 1/|A \cup B|^2 \sum_{v,v' \in A \cup B} d_G(v,v')$, i.e., the average geodesic distance d_G between two nodes in $A \cup B$. This linkage results in clusters with similar states in the sense that the agent can traverse from one state of the cluster to the other with a small number of steps. The maximum linkage ψ of a cluster in `OGAHC` has been set to 3.0 and skill discovery with `OGAHC` was performed every 10000 steps. The greediness of `IFIGE` has been set to $\alpha_i = 0.25$, the maximal node diameter to $\zeta = 7.5$, and the on-policy edge weights have been used because of the domain's continuous action space. Intrinsic motivation was based on the novelty mechanism with $b = 1$ and the length of the developmental period was set to 50000 steps.

Because of the continuous action space, we have used direct policy search based on evolutionary computation for learning option policies π_o, see Section 2.3.3. The value for j-the action dimension is determined via $a_j = \tanh(\sum_{k=0}^{98} \theta_{jk} s_k)$, where s_k is the value of the k-th state dimension and $s_{98} = 1$ is a bias. The policy's weights θ_{jk} have been optimized using a 16+40 evolution strategy (Beyer and Schwefel, 2002) and each weight vector has been evaluated 10 times. The penalty r_p of the options' pseudo-reward functions R_o, which an agent obtains if an option terminates unsuccessfully (see Section 5.3.3), has been set to $r_p = -100$. The evolution strategies' objective is to maximize the pseudo-reward accumulated in 10 steps, after which the option is interrupted.

As in the multi-valley domain, the agent has initially only a single option o_e in its skill pool O, which can be invoked in any state of the environment, i.e., $I_{o_e} = \mathcal{S}$, and terminates with probability $\beta_{o_e}(s) = 0.1$. The policy π_{o_e} selects actions uniform randomly from the action space. The higher-level policy π_i, which determines the option that is executed, has been learned using Q-Learning with discounting factor $\gamma = 0.99$ and exploration rate $\varepsilon = 0.01$. Because of the high dimensionality of the state space, the value function was not represented using a CMAC function approximator but using a linear combination of state values, i.e., $Q(s,o) = \sum_{k=0}^{98} \theta_{ok} s_k$. The learning rate has been set to 0.1.

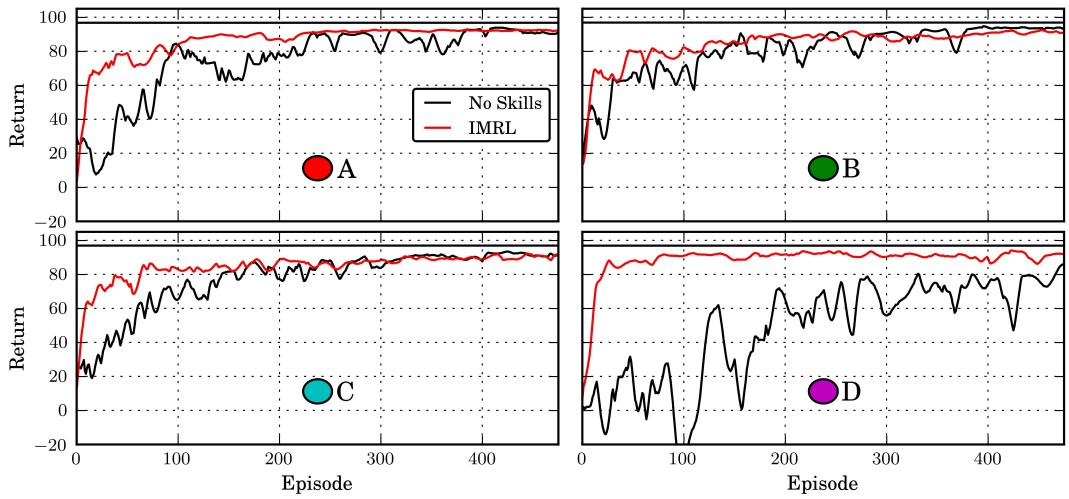


Figure 6.9 – *Performance of IMRL agent in the octopus arm domain.* Shown is return $R^{12}_{\text{mwa}}(t)$ of the agent under the "novelty" motivation after 50000 developmental steps. The circle patches indicate the respective targets used in the runs (see Figure 6.7). "No Skills" shows the performance of an agent that does not learn skills and has no developmental period. The horizontal black line shows the average cost of the policy learned by the monolithic agent after 2500 episodes. All curves show median performance over 5 independent runs and have been smoothened by a moving window average with window length 25.

6.4.2.2 Task Performance

Different tasks can be imposed onto the agent; in this chapter, we require that the agent learns to reach for certain objects that are located at different positions (see Figure 6.7). The agent obtains an external reward of -0.01 per time step and a reward of 100 for reaching the target object. The episode ends after 1000 time steps or once the target object is reached.

Figure 6.8 depicts an example trajectory of the octopus arm learned by the IMRL agent for reaching a target located at position C: the goal is reached after 22 steps and the agent invokes three different skills during this trajectory. The skill executed in the first 11 steps contracts the arm and brings it into an $\cap$-shape. The skill chosen for the next 6 steps unrolls the first part of the arm until an S-shape is reached. The skill executed in the last 5 steps unrolls the second half of the arm such that the target object is reached by an $\cup$-shape. Note that bending the arm directly into an $\cup$-shape would not be successful but result in a state like the one depicted in Figure 6.7.

Figure 6.9 shows the learning curves of the IMRL agent and a monolithic agent, which learns a flat global policy with the same parametrization as the skill policies, for different target positions in the Octopus domain. Given sufficient time, the monolithic agent can learn policies of similar quality as the IMRL agent. Thus, close-to-optimal behavior can be represented by a flat global policy. However, in general, the IMRL agent learns close-to-optimal policies faster and the learning curves exhibit less variance across all tasks. Thus, the temporal abstraction of the skills that were learned in

the developmental period seem to make learning close-to-optimal behavior easier by providing a useful explorative bias. On the other hand, as in the multi-valley domain these abstractions may impair performance slightly in the long run.

6.5 DISCUSSION

This chapter has presented a novel approach for skill acquisition in continuous domains that can be used by an IMRL agent in its developmental period. This skill acquisition approach is based on incremental, graph-based skill discovery and intrinsic motivation. The four main contributions of the proposed approach can be summarized as follows:

1. The agent uses an IMRL architecture that allows applying hierarchical RL in developmental settings based on intrinsic motivation (see Section 6.3.1).
2. A transition graph of the domain is generated incrementally using `IFIGE` (see Section 6.3.2.1). `IFIGE` is an extension of the method `FIGE` proposed in Chapter 5 which allows "growing" a transition graph over time as the agent explores the environment consecutively. Thus, in contrast to `FIGE`, `IFIGE` does neither require specifying a single point in time where the transition graph is generated nor specifying the number of graph nodes.
3. Skills are discovered based on the generated transition graph using an extension of the method `OGAHC`, which was proposed in Chapter 4, to continuous domains. The extension proposed in Section 6.3.2.2 allows handling situations where the underlying transition graph changes over time, which is the case for instance when using `IFIGE` for generating the graph.
4. Two intrinsic motivation mechanisms have been proposed which are based on novelty and the prediction-error of an option model (see Section 6.3.3).

The empirical results for the proposed IMRL approach in two continuous RL domains support the following findings:

- The proposed IMRL approach based on incremental skill discovery and intrinsic motivation allows acquiring skills in a developmental period that can improve performance in external tasks considerably compared to a monolithic learning approach (see Section 6.4.1.2 and Section 6.4.2.2). This means, close-to-optimal behaviors can be learned in less trials because of the explorative bias provided by the temporal abstractions of the skill hierarchy.
- The long-term performance of the IMRL agent remains slightly below those which can be reached by a monolithic agent ultimately (see Section 6.4.1.2 and Section 6.4.2.2). This is to be expected for hierarchical approaches: on the one hand, the (temporal) abstraction introduced by the skills helps the agent in learning faster. On the other hand, however, the abstractions on lower layers also reduce the class of representable high-level policies as they mask certain potentially relevant details of the environment from the upper levels. Carefully choosing the temporal abstractions—which is exactly the objective of skill discovery—should allow speeding up initial learning considerably while still allowing close-to-optimal

long term performance. The empirical results suggest that the proposed skill discovery approach handles this trade-off satisfyingly.

- The length of the developmental period is important: while a sufficiently long developmental period allows improving task performance considerably, a too short developmental period might even deteriorate the performance of an IMRL agent compared to an agent which learns a monolithic policy from scratch (see Section 6.4.1.2).
- Even skills which have not been learned completely during the developmental period, i.e., whose policies are sub-optimal, can speed-up the learning of task solutions considerably (see Section 6.4.1.2). This reinforces and extends results of Kirchner (1999) to continuous domains and behavioral hierarchies that have been built autonomously by the agent by means of skill discovery.
- Intrinsic motivation based on novelty allows acquiring better skills in a shorter developmental period than prediction-error based intrinsic motivation. The skills acquired under novelty motivation are preferable because they are executed more often successfully and increase performance in external task compared to prediction-error based motivation (see Section 6.4.1).
- The reason for the better performance of novelty-based intrinsic motivation is that it is able to handle the trade-off between skill discovery and skill learning, which is intrinsic to skill acquisition, better than prediction-error based intrinsic motivation. This means, it adjusts the ratio of skill learning to exploration more appropriate, cf. Figure 6.5.
- The proposed skill acquisition approach scales to the high-dimensional Octopus arm problem and is able to discover skills in this problem which allow decomposing a complex movement into simpler parts (see Section 6.4.2.2).

In summary, incremental graph-based skill discovery based on `IFIGE` and `OGAHC` as outlined in Section 6.3.2 exhibits all desirable properties identified in Section 3.2.3: it is incremental (P1), it identifies task-independent skills (P2), decides automatically how many skills are created (P3), is suited for continuous domains (P4), and—as the empirical results show—is sample-efficient (P5). By combining this skill discovery approach with means for intrinsic motivation, subgoal S3 stated in Section 1.2 has been achieved, i.e., an intrinsic motivation mechanism has been developed that allows incremental skill acquisition in the absence of external reward signals such as in developmental settings.

This work can be extended in numerous ways: for instance, instead of performing skill discovery only in the developmental period, the agent could also discover novel skills and learn based on intrinsic motivation while it is faced with an external task. This, however, requires trading off intrinsic and external rewards and facing the exploration-exploitation dilemma. This is left to future work; however, the author would like to emphasize that the proposed skill discovery approach is in no way restricted to developmental settings. A further direction of future work is to combine the proposed skill discovery approach with more sophisticated intrinsic motivation mechanisms such as competence progress intrinsic motivation (Stout and Barto, 2010) or other means for empirically estimating the learning progress (see, e.g., Lopes et al. (2012)).

7 Conclusion and Outlook

> "Only those who have the patience to do simple things perfectly will acquire the skill to do difficult things easily."
>
> Friedrich Schiller

7.1 SUMMARY

THIS thesis has proposed a novel approach for autonomous skill acquisition in artificial, lifelong learning agents. The *central idea* of this thesis is that structure, which is inherent in an environment, can be used as basis for identifying reusable and versatile skills. This idea was motivated by the observation that the structure of an environment is typically independent of an agent's task and is thus a good basis for the identification of reusable skills. Moreover, the structure of a domain can be uncovered incrementally and allows the agent thus to acquire an increasing repertoire of skills step-by-step. The three main *contributions* of this thesis are the following:

1. Chapter 4 proposed the novel skill discovery approach `OGAHC`. `OGAHC` generates a transition graph, which captures the dynamics of a discrete domain, based on experience and discovers reusable and versatile skills by identifying bottlenecks of this graph using agglomerative hierarchical clustering. `OGAHC` is fully incremental and allows identifying bottlenecks of a domain robustly in the face of domain stochasticity and different explorative behavior of the agent.

2. Chapter 5 presented `FIGE`, a new method for generating transition graphs from experience that is is suited for continuous domains. `FIGE` is based on formalizing the problem as a probabilistic generative model and using maximum likelihood estimation for determining the transition graph. Using `FIGE`, graph-based skill discovery and representation learning approaches developed for discrete domains can be extended to continuous domains. An incremental version of `FIGE` denoted `IFIGE` was proposed in Chapter 6.

3. The third contribution presented in Chapter 6 consists of the development of an intrinsic motivation mechanism based on novelty. This intrinsic motivation mechanism allows incremental skill acquisition in the absence of external reward signals, e.g., in developmental or lifelong learning settings. It governs the behavior of an agent during skill acquisition such that the agent is able to allocate its time dynamically between skill learning and skill discovery.

By integrating this novelty-based intrinsic motivation mechanism with `OGAHC` and `IFIGE`, an incremental skill acquisition approach is obtained that achieves the goal that was stated in Section 1.2:

> **Goal:** *Develop an incremental, self-motivated approach for skill acquisition which is based on identifying and exploiting the structure of a problem and which can be used in developmental and lifelong learning settings in continuous and stochastic domains.*

Our *empirical evaluation* of the resulting approach shows that: (1) `OGAHC` is able to identify bottlenecks earlier and more reliable than other graph-based skill discovery approaches which are non-incremental or based on repeated sampling (see Section 4.3). (2) `FIGE` generates transition graphs which capture the dynamics of a continuous MDP better than other heuristics proposed in prior work. Due to this, skill discovery and representation learning based on `FIGE` outperform related approaches that are based on other heuristics for transition graph generation (see Section 5.4). (3) The overall skill acquisition approach allows an agent to acquire reusable and versatile skills in developmental settings, which considerably improve task performance compared to monolithic learning approaches (see Section 6.4). Moreover, the proposed approach is able to scale to continuous, high-dimensional problems and multi-task settings.

7.2 INSIGHTS

The main insights that have been obtained based on the empirical evaluation of the proposed approach are the following:

1. Domain structure is a good basis for discovering reusable and versatile skills. Identifying and exploiting domain structure, e.g., in the form of bottlenecks, has proven to be an effective basis for skill discovery. While some related works (cf. Section 3.3) have made similar observations in discrete and deterministic domains, this thesis provides evidence that this finding remains valid in continuous, stochastic, and high-dimensional problems. The resulting skills are reusable since they have been identified solely based on properties of the domain and are independent of the tasks that have been imposed onto the agent. Moreover, the discovered skills are versatile, i.e., they facilitate learning solutions for different tasks.

2. Skill discovery must be an incremental, open-ended endeavor. The experiments have repeatedly shown that it is impractical to choose a single point in time at which the agent shall discover all useful skills. If this point in time would be too early, some skills might be missed and an incomplete set of skills might be harmful. Being conservative by performing skill discovery very late during learning is undesirable since the skills might be acquired too late to be useful for the agent. Thus, incremental skill discovery is important since it allows discovering skills at any time during learning once enough evidence is collected that the respective skill may be useful. Please refer to Chapter 4 for more details.

3. A behavioral hierarchy can benefit from suboptimal skills. While it is in general considered to be desirable to form a behavioral hierarchy in which the policies of lower-level skills have converged to (locally) optimal policies, also skills with suboptimal policies can be beneficial for increasing learning performance on higher levels. This has already been observed and discussed by Kirchner (1999), who showed that higher-level behavior can benefit from suboptimal lower-level skills in a fixed behavioral hierarchy. The empirical results of this thesis provide additional evidence that this finding extends to continuous domains and behavioral hierarchies that have been built autonomously by the agent by means of skill discovery. A possible explanation for this finding it that also suboptimal skills offer a temporal abstraction that can be a useful explorative bias. Please refer to Section 6.4.1.2 for more details.

4. Intrinsic motivation can guide skill acquisition in developmental settings. In an incremental skill acquisition approach, there is an implicit trade-off between skill discovery and skill learning: the agent might on the one hand focus on learning the policy of an already discovered skill or on the other hand explore the domain to identify its structure and discover further useful skills. In a developmental setting, in which the agent is not constrained by external tasks, the agent is free to decide at any point in time if it wants to engage in skill learning or skill discovery. In the experiments, intrinsic motivation systems have proven to be an effective means for allocating the agent's time dynamically between skill learning and skill discovery: initially, they guide the agent to explore the environment for discovering new skills. Once the first skills have been discovered, intrinsic motivation encourages the agent to focus on learning policies for these skills. As the skill policies converge, they guide the agent to increase its explorative behavior again to discover further skills. Please refer to Section 6.4.1.1 for details.

5. The overhead of building a behavioral hierarchy pays off in multi-task problems. The experiments have shown that forming a behavioral hierarchy based on autonomous skill acquisition becomes particularly useful in multi-task problems in which the agent has to solve several different but related tasks. In single-task problems, it is often more sample-efficient to learn a monolithic policy from scratch since skill discovery and skill learning only make the learning task more complex in this case.

However, if the agent is faced with a multi-task learning problem, the acquired skills facilitate effective inter-task transfer of procedural knowledge. This overcompensates for the overhead of skill acquisition already for a medium number of tasks. Similar findings have been made in related works in small and discrete domains (cf. Section 3.3). However, this thesis presents evidence that this result holds true also for continuous domains where function approximation is required and intra-option Q-learning is not feasible. Please refer to Section 5.4.2.3 for more details.

7.3 OUTLOOK

This thesis has focused on basic research in hierarchical RL in a definite theoretical framework, namely in MDPs. This has allowed studying the proposed methods in isolation, performing reproducible experiments, and systematically varying properties of agent and environment. The thesis concludes by giving some possible routes for future work that builds upon the presented approaches. The outlined topics are considered to be essential for open-ended, lifelong learning of behavior in autonomous agents such as robots. Please refer also to Barto et al. (2013) for a related discussion.

Complex Behavioral Hierarchies This thesis has focused on hierarchical architectures consisting of three layers. These architectures contain the primitive actions on the lowest layer, the acquired skills on the middle layer, and a high-level policy on the top layer, which selects among the skills. Skill discovery changes the number of skills on the middle layer. The number of layers, however, remains constant. The same is true for most related work. Future work on learning deep architectures, in which skills can invoke other skills, is desirable since the ability to form such deep architectures is commonly considered to be a prerequisite for actual open-ended lifelong learning. The dendrogram generated by the hierarchical clustering of the transition graph (see Section 2.4.3.3) could be an interesting starting point for this.

Intrinsic Motivation Lifelong learning does not only require an agent to know *how* to learn but also to decide *when* and *what* to learn. Knowing when to learn is important since learning of behavior involves exploration which may not be feasible in all situations, e.g., when an agent is in a dangerous situation or has to fulfill a time-critical task. Deciding what to learn is important since complex and dynamic environments offer such a multitude of experience that it becomes impossible for an agent to learn about every aspect of the world. Thus, the agent must decide what aspects of the world are relevant and which behavior is worth the effort of being learned. An intrinsic motivation system can address these issues. However, a lifelong learning agent will require a sophisticated motivation system that need not be redesigned for different problems anew. The intrinsic motivation systems proposed in this thesis as well as related works are typically problem-specific or focus solely on specific settings like the developmental

one. Future work on more general intrinsic motivation systems will thus be important for lifelong learning in robotics.

Parametrized Options This work has modeled skills as options, which are essentially closed-loop policies. However, as pointed out by Barto et al. (2013), what is commonly denoted as a skill is actually more flexible than an option: for instance, the skill "throwing an object" would correspond to many different options whose policies depend on "contextual information" such as the target positions, the type of object, or the desired trajectory of the throw. As proposed by da Silva et al. (2012), skills can be considered as family of options which are parameterized by the context. Transfer learning can be used to generalize from contexts for which option policies have been learned to novel but related contexts. Ongoing work by the author not covered in this thesis extends these parameterized options to skill templates, which take the uncertainty of the generalization into account during transfer, and aims at applying hierarchical RL in robotic manipulation tasks (Metzen and Fabisch, 2013; Metzen et al., 2014).

Related to this is contextual policy search (Deisenroth et al., 2013), a multi-task learning approach in which several options for different contexts are learned concurrently and a high-level policy generalizes experience of these low-level options over different contexts. Thus, there is some recent work on learning such "contextual" options. However, we do not know any work on the *discovery* of such options. The work of Daniel et al. (2012) could be an interesting starting point.

Undirected Behavior This thesis has focused on goal-directed skills, i.e., skills which terminate once a specific region of the state space is reached. While such skills are useful behavioral building blocks, for instance in the area of object manipulation, they are not suited for all kinds of behavior. For instance, they are not applicable to rhythmic and repetitive behavior such as walking, running, swimming, swinging, or stirring. For such behavior, rather than identifying goal regions, it is important to identify specific patterns in the dynamic behavior such as desired limit cycles. Future work on acquiring reusable building blocks for this kind of behavioral would be an interesting complement to the works discussed and proposed in this thesis.

Real-world Robotic Applications The most important future work will be to show the potential of hierarchical RL in real-world problems, e.g., in robotic applications. In this thesis—as in nearly all related work—the empirical evaluation has been conducted in simulated environments. This has the advantage that strengths and weaknesses of methods can be systematically explored, for instance by varying the stochasticity of a domain or the explorative behavior of the agent. While some of the problems considered in this thesis, e.g., the Octopus arm problem, should not be considered as "toy" problems since they are challenging for both humans and conventional control approaches due to their complex dynamics, future work that shows the potential of hierarchical RL in important real-world tasks is much needed (Barto et al., 2013). The following list gives some of the most severe challenges and how they could be addressed:

- Real-world problems in robotics are *noisy, partially observable, and potentially non-Markovian*. Furthermore, learning must *not impair the robot*. One way of addressing these challenges is to integrate hierarchical RL into a robotic control architecture such as the one shown in Figure 1.1. By this, the learning component could be provided with a more abstracted and curated view onto the problem that is more amenable to RL. For instance, adding low-level reflexes and behavior supervision modules can reduce the risk that explorative behavior of the agent impairs the system. Perception modules can reduce noise in the sensors and estimate unobserved components of the environment's state based on the sensory input. By this, the level of noise and partial observability may be reduced.

- Real-world problems in robotics require learning in *high-dimensional state and action spaces*. Traditional RL approaches like temporal difference learning suffer from the curse of dimensionality and do not scale easily to this kind of problems. Direct policy search approaches with problem-specific policy representations are considered to be more promising (Deisenroth et al., 2013). However, the combination of direct policy search with hierarchical RL is an area where further research is required and promises considerable progress for robot learning.

- Real-world problems require that agents *learn novel and adapt existing knowledge with few trials*. This implies that behavior is not learned from scratch but that reusable, modular building blocks of procedural knowledge are acquired which simplify learning of novel and adaptation of existing behavior. This thesis has focused on and contributed to the discovery and learning of these building blocks. Future work will require developing means which allow the efficient utilization of the building blocks in robotic tasks.

7.4 Closing Words

Artificial agents with the capacity for lifelong learning of new behaviors would be of considerable scientific and economical interest. Yet, the astounding capability of biological beings for learning and adaptation is not easily reproduced in artificial systems. Autonomous skill acquisition is considered to be one important component of the capacity for lifelong learning. The author believes that the novel approach for skill acquisition proposed in this thesis is one small step in the direction of artificial, lifelong learning agents; albeit, there remain several further steps to be taken before we may see artificial agents that come close to the capabilities of biological beings in this regard. Alan Turing's following words remain all too true:

> "We can only see a short distance ahead, but we can see plenty there that needs to be done."
>
> Alan Turing, Computing Machinery and Intelligence, 1950, pp.456

A Derivations

A.1 Derivation of Transition Graph Weights

We give a derivation of the off-policy and on-policy weights as the maximum likelihood estimate $w^* = \arg\max_w L_T(w)$ for a given set of transitions $T = \{(s_i, a_i, s_i')\}_{i=1}^n$ (see Section 4.2.1). Recall that the edge attribute $n_{vv'}^a$ stores the number of occurrences of the transition (v, a, v') in T and the node attribute n_v^a the number of times that action a has been invoked in v. Using the shorthand notation $w_{vv'} = w((v, v'))$, we can derive for $L_T(w) = p(T|w)$:

$$\begin{aligned} p(T|w) &= \prod_{i=1}^n p((v_i, a_i, v_i')|w) = \prod_{i=1}^n p(v_i|w)p(a_i|v_i, w)p(v_i'|v_i, a_i, w) \\ &\overset{\text{GM}}{=} \prod_{i=1}^n \frac{1}{|V|}\frac{1}{|A|} w_{v_i v_i'} = \frac{1}{|V|^n}\frac{1}{|A|^n} \prod_{v,a,v'} (w_{vv'})^{n_{vv'}^a} \\ &= \frac{1}{|V|^n}\frac{1}{|A|^n} \prod_{v,v' \in V} (w_{vv'})^{\sum_a n_{vv'}^a}, \end{aligned}$$

where the generative model proposed in Section 4.2.1 was used in step "GM". Based on this, we can now derive the following formula for the maximum likelihood estimate (MLE) of the graph weights:

$$w^* = \arg\max_w p(T|w) = \arg\max_w \frac{1}{|V|}\frac{1}{|A|^n} \prod_{v,v' \in V}^n (w_{vv'})^{\sum_a n_{vv'}^a} = \arg\max_w \prod_{v,v' \in V} (w_{vv'})^{\sum_a n_{vv'}^a}$$

The only constraint on the weights is that $\sum_{v'} w_{vv'} = 1 \; \forall v \in V$, which guarantees that w encodes proper probabilities. Thus, we can maximize the likelihood for every subset $w_v = \{w_{vv_0}, \ldots w_{vv_n}\}$ of w, which contains the weights for the edges starting in vertex v, separately:

$$w_v^* = \arg\max_{w_v} \prod_{v' \in V} (w_{vv'})^{\sum_a n_{vv'}^a}$$

By identifying this as the standard maximum likelihood formulation for the multinomial distribution, where the different v' are the possible outcomes of the multinomial

trial, we obtain as maximum likelihood estimate for the weights (by using Lagrange multipliers):

$$w_{vv'} = \sum_a n^a_{vv'} / \sum_{\tilde{v}} \sum_a n^a_{v\tilde{v}} = \frac{1}{n_v} \sum_a n^a_{vv'}, \tag{A.1}$$

where $n_v = \sum_a n^a_v$ is the number of visits of vertex v. This solution corresponds to the weights w_{on} proposed by Şimşek et al. (2005) where the additional normalization constant $\frac{1}{n_v}$ ensures that the edge weights are proper probabilities. However, note that the transitions are generated by an agent which typically does not explore uniform randomly, i.e., $\pi(v,a) = p(a|v) \neq 1/|V|$. Thus, we have a sampling bias in T. We can compensate for this sampling bias by importance sampling (Srinivasan, 2002), i.e., by assigning to each transition the weight $Q(a|v)/P(a|v)$. Here, $Q(a|v)$ is the target distribution and $P(a|v)$ is the proposal distribution under which the transitions have been generated. Thus: $Q(a|v) = 1/|A|$ and $P(a|v) = \pi(v,a) = n^a_v/n_v$, where we have estimated the agent's policy based on statistics of T. Accordingly, the importance weight of transition (v,a,v') is $n_v/(|A|n^a_v)$. By inserting this in the MLE equation, we obtain $w^*_v = \arg\max_{w_v} \prod_{v' \in V} (w_{vv'})^{\sum_a n^a_{vv'} n_v/(|A|n^a_v)} = \arg\max_{w_v} \prod_{v' \in V} (w_{vv'})^{n_v/|A| \sum_a n^a_{vv'}/n^a_v}$ and correspondingly

$$w_{vv'} = \sum_a \frac{n^a_{vv'}}{n^a_v} / \sum_a \sum_{\tilde{v}} \frac{n^a_{v\tilde{v}}}{n^a_v} = \sum_a \frac{n^a_{vv'}}{n^a_v} / \sum_a \frac{n^a_v}{n^a_v} = \frac{1}{|A|} \sum_a \frac{n^a_{vv'}}{n^a_v}. \tag{A.2}$$

These are exactly the "off-policy" weights w_{off} used in Chapter 4.

A.2 Derivation of FIGE's Update Equations

In this appendix, we give a derivation of FIGE's update equations based on the likelihood $L_T(G)$ of a graph G for a given set of transitions $T = \{(s_i, a_i, s'_i)\}_{i=1}^n$ under two simplifying assumptions. Please refer to Section 5.3.1 for a definition of the likelihood $L_T(G) = p(T|G)$. We aim at finding a state transition graph G^* with v_{num} nodes such that $G^* = \arg\max_G L_T(G)$. We derive the FIGE update equations as maximum likelihood solutions for $L_T(G)$ under two simplifying assumptions:

(A1) For $(s,a,s') \in T$, assume $p(v'|v,a) = 1$ if $v = \text{NN}_V(s) \wedge v' = \text{NN}_V(s')$ else 0.

Assumption A1 allows to effectively decouple the likelihood $p(T|G)$ from the graph's edges and their weights $w_{vv'}$ such that it depends solely on the graph node positions and can thus be written as $p(T|V)$. By using assumption A1, we obtain:

$$\begin{aligned}
\log p(T|G) &= \log \frac{1}{|A|^n |V|^n} \prod_{i=1}^n \left[\sum_{v \in V} p(s_i|v) \sum_{v' \in V} p(v'|v,a) p(s'|v',v,s_i) \right] \\
&\overset{A1}{=} \log \frac{1}{|A|^n |V|^n} \prod_{i=1}^n p(s_i|\text{NN}_V(s_i)) p(s'_i|\text{NN}_V(s'_i), \text{NN}_V(s_i), s_i) \hat{=} \log p(T|V)
\end{aligned}$$

$$\log p(T|V) = -n\log|A||V| + \sum_{i=1}^{n}\left[\log p(s_i|\mathrm{NN}_V(s_i)) + \log p(s_i'|\mathrm{NN}_V(s_i'),\mathrm{NN}_V(s_i),s_i)\right]$$
$$= -n\log|A||V| + n\log N_b - \frac{1}{b^2}D$$

with $D = \sum_{i=1}^{n}\left[\|s_i - \mathrm{NN}_V(s_i)\|_2^2 + \|(\mathrm{NN}_V(s_i') - \mathrm{NN}_V(s_i)) - (s_i' - s_i)\|_2^2\right]$. For given V, we create two partitions of T: $T^{\rightarrow}(v) = \{(s,s') \mid \exists(s,a,s') \in T : \mathrm{NN}_V(s) = v\}$ and $T^{\leftarrow}(v) = \{(s,s') \mid \exists(s,a,s') \in T : \mathrm{NN}_V(s') = v\}$. Furthermore, we create v_{num} sets S^V with $S^V(v) = \{s \mid \exists(s,a,s') \in T : \mathrm{NN}_V(s) = v\}$. For $|V| = v_{num} = const$, we can now maximize the log-likelihood $\log p(T|V)$ by minimizing D:

$$D = \sum_{i=1}^{n}\left[\|s_i - \mathrm{NN}_V(s_i)\|_2^2 + 2\frac{1}{2}\|(\mathrm{NN}_V(s_i') - \mathrm{NN}_V(s_i)) - (s_i' - s_i)\|_2^2\right]$$
$$= \sum_{v}\Bigg[\sum_{s\in S^V(v)}\|s - v\|_2^2 + \frac{1}{2}\sum_{s,s'\in T^{\rightarrow}(v)}\|\mathrm{NN}_V(s') - v - s' + s\|_2^2$$
$$+ \frac{1}{2}\sum_{s,s'\in T^{\leftarrow}(v)}\|v - \mathrm{NN}_V(s) - s' + s)\|_2^2\Bigg]$$

Each term of the outer sum corresponds to the contribution of node v's position to D; however, the terms cannot be minimized separately since they are coupled via $\mathrm{NN}_V(s)$ and $\mathrm{NN}_V(s')$. Minimizing them jointly is difficult because of the discontinuities of the nearest-neighbor terms. Thus, `FIGE` makes the following simplifying assumption:

$$\text{(A2) Assume } p(T|V) = \textstyle\prod_v p(T|v)$$

This assumption implies that the couplings between the terms of D are not taken into account and each v can be set greedily to the position where the respective term in the outer sum would become minimal when all other $\tilde{v} \in V$ would remain unchanged. Finally, the greedy `FIGE` update equation which moves a node from position v_{old} to position v_{new} is

$$v_{new} = \arg\min_{v}\Bigg[\sum_{s\in S^V(v_{old})}\|s - v\|_2^2 + \frac{1}{2}\sum_{s,s'\in T^{\rightarrow}(v_{old})}\|(\mathrm{NN}_V(s') - s' + s) - v\|_2^2$$
$$+ \frac{1}{2}\sum_{s,s'\in T^{\leftarrow}(v_{old})}\|v - (\mathrm{NN}_V(s) - s + s')\|_2^2\Bigg]$$

In this, the first sum is minimized by choosing $v_{new} = v_a = \mathrm{MEAN}_{s\in S^V(v_{old})}(s)$, the second sum by choosing $v_{new} = v_b = \mathrm{MEAN}_{s,s'\in T^{\rightarrow}(v_{old})}(\mathrm{NN}_V(s') - s' + s)$, and the third by $v_{new} = v_c = \mathrm{MEAN}_{s,s'\in T^{\leftarrow}(v_{old})}(\mathrm{NN}_V(s) - s + s')$. By using forces that pull v_{new} to v_a, v_b, and v_c with the respective weights, we obtain the `FIGE` update rule

$$v_{new} = v_{old} + \alpha\left[\frac{1}{2}(v_a - v_{old}) + \frac{1}{4}(v_b - v_{old}) + \frac{1}{4}(v_c - v_{old})\right]. \tag{A.3}$$

Since $A2$ is oversimplifying, one sweep of the `FIGE` update equations will typically not find the maximum likelihood solution. Thus, `FIGE` performs several update iterations to account for couplings between nodes.

List of Figures

1.1 Three-layer Robot Control Architecture 2
1.2 Examples of Acquired Skills 3
1.3 Schematic Illustration of the Thesis Structure 7

2.1 Sequential Decision Problem 12
2.2 Markov Decision Process 13

3.1 "Towers of Hanoi" with Five Pieces 35
3.2 State Transition Graph of "Towers of Hanoi" 36
3.3 Visit Count under Random Walks in "Towers of Hanoi" 38
3.4 Betweenness for the "Towers of Hanoi" 44
3.5 Illustration of Agglomerative Clustering in "Towers of Hanoi" 49

4.1 Data Flow Diagram of Graph-Based Skill Discovery 58
4.2 Illustration of OGAHC in "Towers of Hanoi" 67
4.3 Illustration of Graph Smoothing in "Towers of Hanoi" 68
4.4 Illustration of a Skill Prototype 69
4.5 Example of a Randomly Generated Euclidean Graph 71
4.6 Comparison of Graph Clustering Algorithms 72
4.7 Analysis of Granularity of Partitions 73
4.8 Effect of Graph Smoothing onto Bottleneck Detection Quality 73
4.9 Comparison of On- and Off-Policy Edge Weights 74
4.10 Examples of Random Graphs Governing the MDP's State Connectivity 76
4.11 Effect of Domain Stochasticity and Exploration on Graph Clustering . 76
4.12 Comparison of Graph Clustering Methods 77
4.13 Average Accordance of OGAHC and Off-line Clustering. 78
4.14 Performance of OGAHC in the Multi-Task Maze 79
4.15 Learning Curves in the Multi-Task Maze 80

5.1 Graph-Based Skill Discovery in Continuous Domains 87
5.2 Illustration of the Forces in FIGE 92
5.3 Illustration of the Mountain Car Domain 92
5.4 Illustration of Transition Graphs during FIGE Iterations 93
5.5 Illustration of Different Heuristics for Transition Graph Generation . . 94
5.6 Log-Likelihood of Transition Graphs 96

5.7 Illustration of 2D Multi-Valley Domain 98
5.8 Accordance Ratio of Graph Partitions 98
5.9 Evaluation of FIGE-Based Skill Discovery in 2D Multi-Valley 100
5.10 Skill Discovery in Multi-Task Domains for Different Number of Tasks 101
5.11 Representation Policy Iteration in Continuous Domains 103
5.12 Illustration of Representation Policy Iteration in Mountain Car 104
5.13 Evaluation of Representation Policy Iteration in Mountain Car 105
5.14 Evaluation of Representation Policy Iteration in Inverted Pendulum . . 106
5.15 Evaluation of Representation Policy Iteration in Octopus Arm 107

6.1 Agent Architectures for Developmental Period and for Task Learning . 117
6.2 Illustration of Skill Discovery in 1D Multi-Valley 122
6.3 Visualization of Generated Transition Graphs in 2D Multi-Valley . . . 124
6.4 Success Ratio of Learned Skills during Developmental Period 125
6.5 Ratio of Skill Learning and Exploration during Developmental Period 126
6.6 Performance of IMRL Agent in 2D Multi-Valley 127
6.7 Visualization of the Octopus Arm Task 128
6.8 Example Trajectory of the Octopus Arm Controlled by the IMRL Agent 129
6.9 Performance of IMRL Agent in the Octopus Arm Domain 130

List of Algorithms

2.1 Q-Learning 18
2.2 Normalized Spectral Clustering 29
2.3 PCCA$^+$ 30
2.4 Agglomerative Hierarchical Clustering 31
4.1 Constrained Agglomerative Clustering 63
4.2 On-line Graph-Based Agglomerative Hierarchical Clustering (OGAHC) 64
4.3 Graph Smoothing (SMOOTH) 65
5.1 Force-Based Iterative Graph Estimation (FIGE) 91
5.2 Representation Policy Iteration (RPI) 102
6.1 Incremental Force-Based Iterative Graph Estimation (IFIGE) 119

List of Symbols

The notation and the symbols used in this work follow mainly the convention proposed by Sutton and Barto (1998) and von Luxburg (2007).

Markov Decision Process

$\mathcal{M}$	Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P^a_{ss'}, R^a_{ss'}, S_0, S_T)$ 13
$\mathcal{S}$	set of all states of an MDP 13
$\mathcal{A}$	set of all actions of an MDP 13
s	state $s \in \mathcal{S}$ of an MDP 13
a	action $a \in \mathcal{A}$ of an MDP 13
s'	successor state $s' \in \mathcal{S}$ when executing action a in state s 13
$P^a_{ss'}$	$P^a_{ss'} = P(s'\|s,a)$; probability of transition from state s to s' under action a 13
$R^a_{ss'}$	$R^a_{ss'} = R(s,a,s')$; expected reward for transition from state s to s' under action a 13
$S_0(s)$	probability that an episode starts in state s 13
$S_T(s)$	probability that an episode terminates when visiting state s 13
$P(s,a)$	successor state of state s under action a in a deterministic MDP 14
γ	discount-rate of an MDP under the infinite-horizon discounted return model . . . 14
n_s	number of dimensions of a continuous state space, i.e., $S \subseteq \mathbb{R}^{n_s}$ 14
n_a	number of dimensions of a continuous action space, i.e., $A \subseteq \mathbb{R}^{n_a}$ 14
χ	parameter controlling stochasticity of MDP 75

Agent-Environment Interaction

t	discrete time step 11
T	final time step of an episode 12
s_t	state of the environment at time t 12
a_t	action taken by the agent at time t 12
r_t	reward obtained by the agent at time t 12

Policies and Value Functions

π	policy of an agent 12
$\pi(s)$	action taken under deterministic policy π 14
$\pi(s,a)$	$\pi(s,a) = \pi(a\|s)$; probability of action a in state s under stochastic policy π . . . 14
π^*	optimal policy (not necessarily unique) 15
R_t	return after time t; for the infinite-horizon, discounted model $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ 15
$V^\pi(s)$	state value function for policy π 15
$Q^\pi(s,a)$	action value function for policy π 15
$Q^*(s,a)$	optimal action value function 16

AGENT PARAMETERS

α	learning rate	18
ε	probability of performing a random action in an ε-greedy policy	19
ψ	threshold (in units of linkage criterion) controlling granularity of clustering	60
ρ	smoothing parameter controlling how liberal bottleneck detection is	65
v_{num}	number of nodes of transition graph generated by `FIGE`	86
K	number of iterations of `FIGE`	90
p_{num}	number of proto-value functions used in RPI	102
ζ	maximal node diameter threshold of `IFIGE`	119

FUNCTION APPROXIMATION

θ	learnable parameters of a function approximator	20
ϕ	feature vector used in a function approximator	20

OPTIONS FRAMEWORK

o	option $o =< I_o, \pi_o, \beta_o >$, a temporally extended activity	23
I_o	$I_o \subset S$, set of states in which option o can be initiated	23
β_o	$\beta_o : S \to [0,1]$, probability that option o terminates in state s	23
π_o	policy of option o	23
R_o	option-specific "pseudo" reward function	24
r_p	penalty used in option pseudo reward function R_o	68
Ψ_o	skill prototype $\Psi_o = (I_o, \beta_o, R_o)$	24

GRAPH THEORY

G	a graph $G = (V, E)$	26
V	a set of graph vertices	26
E	a set of edges $E \subset V \times V$	26
v_i	a graph vertex $v_i \in V$	26
e_{ij}	a graph edge $e_{ij} \in E$ connecting vertex v_i with vertex v_j	26
A	a subset of graph nodes $A \subset V$	26
$\overline{A}$	the complement of a set of graph nodes $A \subset V$, i.e., $\overline{A} = V \setminus A$	26
$p_G(v_a, v_b)$	a path connecting vertices $v_a, v_b \in V$ via a sequence of edges in G	26
$d_G(v_a, v_b)$	geodesic distance of vertices $v_a, v_b \in V$ in G, i.e., length of shortest path	26
G'	a subgraph of graph G: $G' \subset G$	27
w	function assigning weights to graph edges, $w : E \to \mathbb{R}$	27
w_{ij}	weight assigned to edge e_{ij}, i.e., $w_{ij} = w(e_{ij})$	27
W	weighted adjacency matrix of G, $W = (w_{ij})_{i,j=1,\ldots,n}$	27
d_i	degree of vertex v_i, $d_i = \sum_{j=1}^{n} w_{ij}$	27
D	degree matrix $D = \mathrm{diag}(d_1, \ldots, d_n)$	27
A_i	subset of graph nodes $A_i \subset V$, part of a graph partition P, also called "cluster"	27
$\mathcal{P}$	a graph partition $\mathcal{P} = \{A_1, \ldots, A_k\}$ with $A_i \neq \emptyset$, $A_i \cap A_j = \emptyset$, and $\bigcup_{i=1}^{k} A_i = V$	27
$\mathrm{CC}\{\mathcal{P}\}$	set of cluster pairs of $\mathcal{P}$ which are connected in G	70

GRAPH CUTS

$\mathrm{vol}(G)$	volume of a subgraph $G' = (V', E') \subset G$: $\mathrm{vol}(G') = \sum_{i \in V'} d_i$ 27
$\mathrm{cut}(\mathcal{P})$	cut of partition $\mathcal{P}$: $\mathrm{cut}(\mathcal{P}) = 0.5 \sum_{i=1}^{k} W(A_i, \overline{A_i})$ 27
$\mathrm{RatioCut}(\mathcal{P})$	ratio-cut of partition $\mathcal{P}$: $\mathrm{RatioCut}(\mathcal{P}) = \sum_{i=1}^{k} \mathrm{cut}(\{A_i, \overline{A_i}\})/\lvert A_i \rvert$ 28
$\mathrm{NCut}(\mathcal{P})$	normalized-cut of partition $\mathcal{P}$: $\mathrm{NCut}(\mathcal{P}) = \sum_{i=1}^{k} \mathrm{cut}(\{A_i, \overline{A_i}\})/\mathrm{vol}(A_i)$ 28

GRAPH LAPLACIANS

I_n	identity matrix of size $n \times n$ 29
$\mathcal{L}$	the unnormalized graph Laplacian matrix $\mathcal{L} = D - W$. 29
$\mathcal{L}_{sym}$	the symmetrized graph Laplacian matrix $\mathcal{L}_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ 29
$\mathcal{L}_{rw}$	the random-walk graph Laplacian matrix $\mathcal{L}_{rw} = D^{-1} L = I_n - D^{-1} W$. 29

TRANSITION GRAPH GENERATION

$\mathbb{G}$	the actual but unknown transition graph of a domain 58
T	set of n transitions sampled from environment: $T = \{(s_i, a_i, s_i')\}_{i=1}^{n}$ 58
$L_T(G)$	likelihood of graph G for given transitions T, i.e., $L_T(G) = p(T \mid G)$ 59
$n_{vv'}^{a}$	number of transitions in T from vertex v to v' under action a 58
n_{v}^{a}	number of samples for action a in vertex v in T 58
$n_{vv'}$	number of transitions from vertex v to v' under any action in T 91
n_v	number of visits of vertex v in T 91
$\mathrm{NN}_V(s)$	nearest neighbor v of state s among vertices V 90

GRAPH CLUSTERING

b_E	boundary of disjoint node sets $A, B \subset V$: $b_E(A,B) = E \cap (A \times B \cup B \times A)$ 60
b_V	boundary states of $A, B \subset V$: $b_V(A,B) = \{v \in V \mid \exists v' \in V : (v, v') \in b_E(A,B)\}$ ..60
c_m	connectivity mass of $A, B \subset V$: $c_m(A,B) = \sum_{e \in E \cap (A \times B)} w(e)$ 60
b^*	a domain's bottleneck edges 60
l	linkage of $A, B \subset V$: $l : 2^V \times 2^V \to \mathbb{R}$ 60
c	constraint for constrained agglomerative clustering: $c : 2^V \times 2^V \to \mathbb{B}$ 63
$\mathcal{D}$	dendrogram created during agglomerative hierarchical clustering 31

INTRINSIC MOTIVATION AND LIFELONG LEARNING

$\mathcal{T}_j$	j-th external task imposed onto the agent 116
$\mathcal{M}_j$	MDP $\mathcal{M}_j = (\mathcal{S}, \mathcal{A}, P_{ss'}^{a}, R_j)$ corresponding to $\mathcal{T}_j$ 116
r_e	external reward provided by environment 118
r_i	intrinsic reward generated by intrinsic motivation 117
π_e	policy aiming to maximize the external reward r_e 118
π_i	policy aiming to maximize the intrinsic reward r_i 117

PERFORMANCE METRICS

$R_{\mathrm{epi}}(t)$	accumulated reward obtained in the t-th episode 53
$R_{\mathrm{mwa}}^{k}(t)$	moving window average for $R_{\mathrm{epi}}(t)$ for window size $2k+1$ 53
R^{∞}	expected accumulated reward per episode for $t \to \infty$ 53
R_{acc}^{t}	reward accumulated during the first t episodes 54
R_{avg}^{t}	average reward per episode during the first t episodes 54
$\mathrm{acc}_{GT}(\mathcal{P})$	accordance of partition $\mathcal{P}$ with ground-truth partition $\mathcal{P}_{GT}$ 55

Bibliography

Albus, J. S. (1975). "A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller (CMAC)." In: *Journal of Dynamic Systems, Measurement, and Control* 97, pp. 220–233.

Amarel, S. (1968). "On Representations of Problems of Reasoning about Actions." In: *Machine Intelligence 3*. American Elsevier Publisher, pp. 131–171.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Anzai, Y. and Simon, H. A. (1979). "The Theory of Learning by Doing." In: *Psychological Review* 86.2, pp. 124–40.

Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., and Yoshida, C. (2009). "Cognitive Developmental Robotics: A Survey." In: *IEEE Transactions on Autonomous Mental Development* 1.1, pp. 12–34.

Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1996). "Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning." In: *Machine Learning* 23.2-3, pp. 279–303.

Asadi, M. and Huber, M. (2005). "Accelerating Action Dependent Hierarchical Reinforcement Learning Through Autonomous Subgoal Discovery." In: *In Proceedings of the ICML 2005 Workshop on Rich Representations for Reinforcement Learning*.

Ashby, W. R. (1956). *An introduction to cybernetics*. London: Chapman & Hall.

Bacon, P.-L. and Precup, D. (2013). "Using label propagation for learning temporally abstract actions in reinforcement learning." In: *Proceedings of the Workshop on Multiagent Interaction Networks (MAIN 2013), held in conjunction with AAMAS 2013*. Saint Paul, Minnesota, USA.

Baird, L. (1995). "Residual Algorithms: Reinforcement Learning with Function Approximation." In: *In Proceedings of the 12th International Conference on Machine Learning*. Morgan Kaufmann, pp. 30–37.

Baldassarre, G. (2011). "What are intrinsic motivations? A biological perspective." In: *IEEE International Conference on Development and Learning (ICDL)*. Vol. 2, pp. 1–8.

Barto, A. G. and Mahadevan, S. (2003). "Recent Advances in Hierarchical Reinforcement Learning." In: *Discrete Event Dynamic Systems* 13.4, pp. 341–379.

Barto, A. G., Konidaris, G., and Vigorito, C. (2013). "Behavioral Hierarchy: Exploration and Representation." In: *Computational and Robotic Models of the Hierarchical Organization of Behavior*. Berlin: Springer.

Barto, A. G., Singh, S., and Chentanez, N. (2004). "Intrinsically motivated learning of hierarchical collections of skills." In: *Proceedings of the 3rd International Conference of Developmental Learning*, 112–119.

Bartsch, S., Birnschein, T., Römmermann, M., Hilljegerdes, J., Kühn, D., and Kirchner, F. (2012). "Development of the six-legged walking and climbing robot SpaceClimber." In: *Journal of Field Robotics* 29.3, pp. 506–532.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bernstein, D. S. (1999). *Reusing Old Policies to Accelerate Learning on New MDPs*. Tech. rep. 99-26. Amherst: University of Massachusetts.

Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Athena Scientific.

Beyer, H.-G. and Schwefel, H.-P. (2002). "Evolution strategies - A comprehensive introduction." In: *Journal Natural Computing* 1.1, pp. 3–52.

Bieberstein, J. (2006). "Evaluierung der Dekomposition von Reinforcement Learning Problemen mittels der Subgoal-Utility-Erweiterung von Q-Learning." de. Diploma Thesis. University of Bremen.

Botvinick, M. M., Niv, Y., and Barto, A. G. (2009). "Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective." In: *Cognition* 113.3, pp. 262–280.

Boyan, J. A. (2002). "Least-squares temporal difference learning." In: *Machine Learning* 49, pp. 233–246.

Bradtke, S. J., Barto, A. G., and Kaelbling, P. (1996). "Linear least-squares algorithms for temporal difference learning." In: *Machine Learning* 22, pp. 33–57.

Bradtke, S. J. and Duff, M. O. (1994). "Reinforcement Learning Methods for Continuous-Time Markov Decision Problems." In: *Advances in Neural Information Processing Systems*. MIT Press, pp. 393–400.

Brooks, R. (1986). "A robust layered control system for a mobile robot." In: *IEEE Journal of Robotics and Automation* 2.1, pp. 14–23.

Chung, F. (1996). *Spectral Graph Theory*. Vol. 92. CBMS Regional Conference Series in Mathematics. American Mathematical Society.

Daniel, C., Neumann, G., and Peters, J. (2012). "Hierarchical Relative Entropy Policy Search." In: *Journal of Machine Learning Research - Proceedings Track*. Vol. 22. Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 273–281.

da Silva, B. C., Konidaris, G., and Barto, A. G. (2012). "Learning Parameterized Skills." In: *Proceedings of the 29th International Conference on Machine Learning*. Edinburgh, Scotland.

Dayan, P. and Hinton, G. E. (1993). "Feudal Reinforcement Learning." In: *Advances in Neural Information Processing Systems* 5, pp. 271–278.

Deisenroth, M. P., Neumann, G., and Peters, J. (2013). "A Survey on Policy Search for Robotics." In: *Foundations and Trends in Robotics* 2.1-2, pp. 1–142.

de Jong, T. and Ferguson-Hessler, M. G. (1996). "Types and qualities of knowledge." In: *Educational Psychologist* 31.2, pp. 105–113.

Deuflhard, P. and Weber, M. (2005). "Robust Perron cluster analysis in conformation dynamics." In: *Linear Algebra and its Applications* 398, pp. 161–184.

Dietterich, T. G. (2000a). "An Overview of MAXQ Hierarchical Reinforcement Learning." In: *Abstraction, Reformulation, and Approximation*. Lecture Notes in Computer Science 1864. Springer Berlin Heidelberg, pp. 26–44.

Dietterich, T. G. (2000b). "Hierarchical reinforcement learning with the MAXQ value function decomposition." In: *Journal of Artificial Intelligence Research* 13, pp. 227–303.

Dietterich, T. G. (2000c). "State Abstraction in MAXQ Hierarchical Reinforcement Learning." In: *Advances in Neural Information Processing Systems 12*. MIT Press, pp. 994–1000.

Digney, B. L. (1996). "Emergent Hierarchical Control Structures: Learning Reactive/ Hierarchical Relationships in Reinforcement Environments." In: *From Animals to Animats: The 4th Conference on Simulation of Adaptive Behavior.* Cambridge, MA: MIT Press, pp. 363–372.

Digney, B. L. (1998). "Learning hierarchical control structures for multiple tasks and changing environments." In: *Proceedings of the 5th Conference on the Simulation of Adaptive Behavior*. MIT Press, pp. 321–330.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). "Ant colony optimization." In: *IEEE Computational Intelligence Magazine* 1.4, pp. 28–39.

Faußer, S. and Schwenker, F. (2010). "Learning a Strategy with Neural Approximated Temporal-Difference Methods in English Draughts." In: *20th International Conference on Pattern Recognition (ICPR)*, pp. 2925–2928.

Fikes, R., Hart, P. E., and Nilsson, N. J. (1972). "Learning and Executing Generalized Robot Plans." In: *Artificial Intelligence* 3, pp. 251–288.

Ghafoorian, M., Taghizadeh, N., and Beigy, H. (2013). "Automatic Abstraction in Reinforcement Learning Using Ant System Algorithm." In: *2013 AAAI Spring Symposium Series*.

Grotzinger, J. P., Crisp, J., Vasavada, A. R., Anderson, R. C., Baker, C. J., Barry, R., Blake, D. F., Conrad, P., Edgett, K. S., Ferdowski, B., Gellert, R., Gilbert, J. B., Golombek, M., Gómez-Elvira, J., Hassler, D. M., Jandura, L., Litvak, M., Mahaffy, P., Maki, J., Meyer, M., Malin, M. C., Mitrofanov, I., Simmonds, J. J., Vaniman, D., Welch, R. V., and Wiens, R. C. (2012). "Mars Science Laboratory Mission and Science Investigation." In: *Space Science Reviews* 170.1-4, pp. 5–56.

Gullapalli, V. and Barto, A. (1992). "Shaping as a method for accelerating reinforcement learning." In: *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 554–559.

Hagen, L. and Kahng, A. B. (1992). "New spectral methods for ratio cut partitioning and clustering." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.9, pp. 1074–1085.

Harlow, H. F. (1950). "Learning and satiation of response in intrinsically motivated complex puzzle performance by monkeys." In: *Journal of Comparative and Physiological Psychology* 43.4, pp. 289–294.

Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning*. 2nd edition. New York, NY, USA: Springer.

Hawasly, M. and Ramamoorthy, S. (2013). "Lifelong Learning of Structure in the Space of Policies." In: *2013 AAAI Spring Symposium Series*. Stanford, CA.

Heidrich-Meisner, V., Lauer, M., Igel, C., and Riedmiller, M. (2007). "Reinforcement Learning in a Nutshell." In: *15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pp. 277–288.

Hengst, B. (2002). "Discovering Hierarchy in Reinforcement Learning with HEXQ." In: *Proceedings of the 19th International Conference on Machine Learning*. Morgan Kaufmann, pp. 243–250.

Hester, T. and Stone, P. (2012). "Intrinsically Motivated Model Learning for a Developing Curious Agent." In: *The Eleventh International Conference on Development and Learning (ICDL)*. San Diego, CA.

Howard, R. A. (1971). *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. Wiley.

Iba, G. A. (1989). "A Heuristic Approach to the Discovery of Macro-Operators." In: *Machine Learning* 3.4, pp. 285–317.

Iocchi, L., Ruiz-del-Solar, J., and Zant, T. v. d. (2012). "Domestic Service Robots in the Real World." In: *Journal of Intelligent & Robotic Systems* 66.1-2, pp. 183–186.

Jong, N. K., Hester, T., and Stone, P. (2008). "The utility of temporal abstraction in reinforcement learning." In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 299–306.

Jonsson, A. and Barto, A. (2006). "Causal Graph Based Decomposition of Factored MDPs." In: *Journal of Machine Learning Research* 7, pp. 2259–2301.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). "Reinforcement Learning: A Survey." In: *Journal of Artificial Intelligence Research* 4, pp. 237–285.

Kalyanakrishnan, S. and Stone, P. (2011). "Characterizing reinforcement learning methods through parameterized learning problems." In: *Machine Learning* 84.1-2, pp. 205–247.

Kazemitabar, S. J. and Beigy, H. (2009). "Using Strongly Connected Components as a Basis for Autonomous Skill Acquisition in Reinforcement Learning." In: *Proceedings of the 6th International Symposium on Advances in Neural Networks*. ISNN '09. Berlin, Heidelberg: Springer-Verlag, pp. 794–803.

Kier, W. M. and Smith, K. K. (1985). "Tongues, tentacles and trunks: The biomechanics of movement in muscular-hydrostats." In: *Zoological Journal of the Linnean Society* 83.4, pp. 307–324.

Kirchner, F. (1995). "Automatic Decomposition of Reinforcement Learning Tasks." In: *Proceedings of the AAAI 95 Fall Symposium Series on Active Learning*. Cambridge, MA, USA: AAAI Press, pp. 56–59.

Kirchner, F. (1996). "Hierarchical Reinforcement Leaming." In: *AAAI-96, Fall Symposium Series*. Cambridge, MA, USA: MIT.

Kirchner, F. (1998). "Q-Learning of complex behaviours on a six-legged walking machine." In: *Journal of Robotics and Autonomous Systems* 25, pp. 256–263.

Kirchner, F. (1999). "Hierarchical Q-Learning in Complex Robot Control Problems." Doktorarbeit. Bonn, Germany: University Bonn.

Kirchner, F. and Richter, C. (2000). "Q-Surfing: Exploring a World Model by Significance Values in Reinforcement Learning Tasks." In: *Proceedings of the European Conference on Artificial Intelligence*, pp. 311–315.

Klopf, A. H. (1972). *Brain Function and Adaptive Systems: A Heterostatic Theory*. Tech. rep. 72. Bedford, MA.

Kober, J., Bagnell, J. A., and Peters, J. (2012). "Reinforcement Learning in Robotics: A Survey." In: *International Journal of Robotics Research* 12, pp. 579–610.

Kober, J. and Peters, J. (2010). "Policy search for motor primitives in robotics." In: *Machine Learning* 84.1-2, pp. 171–203.

Köhler, T., Rauch, C., Schröer, M., Berghöfer, E., and Kirchner, F. (2012). "Concept of a Biologically Inspired Robust Behaviour Control System." In: *Proceedings of 5th International Conference on Intelligent Robotics and Applications*, pp. 486–495.

Kohl, N. and Stone, P. (2004). "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion." In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 3, pp. 2619–2624.

Konidaris, G. (2011). "Autonomous Robot Skill Acquisition." PhD thesis. Amherst: University of Massachusetts.

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

Konidaris, G. and Barto, A. G. (2009). "Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining." In: *Advances in Neural Information Processing Systems*. Vol. 22, pp. 1015–1023.

Konidaris, G., Kuindersma, S., Barto, A., and Grupen, R. (2010). "Constructing Skill Trees for Reinforcement Learning Agents from Demonstration Trajectories." In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 23, pp. 1162–1170.

Konidaris, G., Scheidwasser, I., and Barto, A. G. (2012). "Transfer in reinforcement learning via shared features." In: *Journal of Machine Learning Research* 13.1, pp. 1333–1371.

Korf, R. E. (1985). "Macro-Operators: A Weak Method for Learning." In: *Artificial Intelligence* 26, pp. 35–77.

Lagoudakis, M. G. and Parr, R. (2003). "Least-squares policy iteration." In: *Journal of Machine Learning Research* 4, pp. 1107–1149.

Lemburg, J., de Gea Fernández, J., Eich, M., Mronga, D., Kampmann, P., Vogt, A., Aggarwal, A., Shi, Y., and Kirchner, F. (2011). "AILA - design of an autonomous mobile dual-arm robot." In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, pp. 5147–5153.

Lewis, F. L. and Vrabie, D. (2009). "Reinforcement learning and adaptive dynamic programming for feedback control." In: *IEEE Circuits and Systems Magazine* 9.3, pp. 32–50.

Li, L., Walsh, T. J., and Littman, M. L. (2006). "Towards a Unified Theory of State Abstraction for MDPs." In: *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*. Fort Lauderdale, Florida, USA.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). "Exploration in Model-based Reinforcement Learning by Empirically Estimating Learning Progress." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 206–214.

Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). "Developmental robotics: a survey." In: *Connection Science* 15, pp. 151–190.

Maei, H. R., Szepesvári, C., Bhatnagar, S., and Sutton, R. S. (2010). "Toward Off-Policy Learning Control with Function Approximation." In: *Proceedings of the 27th International Conference on Machine Learning*, pp. 719–726.

Mahadevan, S. and Maggioni, M. (2007). "Proto-value functions: A laplacian framework for learning representation and control in markov decision processes." In: *Journal of Machine Learning Research* 8, pp. 2169–2231.

Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). "Dynamic abstraction in reinforcement learning via clustering." In: *Proceedings of the 21st International Conference on Machine Learning*. ACM, pp. 560–567.

Mathew, V., Kumar, P., and Ravindran, B. (2012). "Abstraction in Reinforcement Learning in Terms of Metastability." In: *Proceedings of the 10th European Workshop on Reinforcement Learning (EWRL)*. Edinburgh, Scotland.

McGovern, A. and Barto, A. G. (2001). "Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density." In: *In Proceedings of the 18th International Conference on Machine Learning*, pp. 361–368.

McGovern, A., Sutton, R. S., and Fagg, A. H. (1997). "Roles of Macro-Actions in Accelerating Reinforcement Learning." In: *Grace Hopper Celebration of Women in Computing*, pp. 13–18.

Mehta, N., Natarajan, S., Tadepalli, P., and Fern, A. (2008). "Transfer in variable-reward hierarchical reinforcement learning." In: *Machine Learning* 73.3, pp. 289–312.

Meila, M. and Shi, J. (2001). "A Random Walks View of Spectral Segmentation." In: *8th International Workshop on Artificial Intelligence and Statistics (AISTATS)*.

Menache, I., Mannor, S., and Shimkin, N. (2002). "Q-Cut - Dynamic Discovery of Sub-goals in Reinforcement Learning." In: *Proceedings of the 13th European Conference on Machine Learning*, pp. 295–306.

Metzen, J. H. (2012a). "Model-based Evolutionary Policy Search for Skill Learning in Continuous Domains." In: *10th European Workshop on Reinforcement Learning (EWRL)*. Edinburgh, Scotland.

Metzen, J. H. (2012b). "Online Skill Discovery using Graph-based Clustering." In: *Journal of Machine Learning Research* W&CP 24. Ed. by M. P. Deisenroth, C. Szepesvári, and J. Peters, pp. 77–88.

Metzen, J. H. (2013). "Learning Graph-based Representations for Continuous Reinforcement Learning Domains." In: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*. Ed. by H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezny. Springer Berlin Heidelberg, pp. 81–96.

Metzen, J. H., Edgington, M., Kassahun, Y., and Kirchner, F. (2008a). "Analysis of an evolutionary reinforcement learning method in a multiagent domain." In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Richland, SC, pp. 291–298.

Metzen, J. H., Edgington, M., Kassahun, Y., and Kirchner, F. (2008b). "Evolving Neural Networks for Online Reinforcement Learning." In: *Proceedings of the 10th Conference on Parallel Problem Solving from Nature (PPSN X)*, pp. 518–527.

Metzen, J. H. and Fabisch, A. (2013). "Learning Skill Templates for Parameterized Tasks." In: *11th European Workshop on Reinforcement Learning (EWRL)*. Dagstuhl, Germany.

Metzen, J. H., Fabisch, A., Senger, L., Gea Fernandez, J. de, and Kirchner, E. A. (2014). "Towards Learning of Generic Skills for Robotic Manipulation." In: *German Journal of Artificial Intelligence (Künstliche Intelligenz)* Special Issue "Transfer Learning". Accepted.

Metzen, J. H. and Kirchner, F. (2010). "Model-based direct policy search." In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Toronto, Canada, pp. 1589–1590.

Metzen, J. H. and Kirchner, F. (2013). "Incremental Learning of Skill Collections based on Intrinsic Motivation." In: *Frontiers in Neurorobotics* 7.11, pp. 1–12.

Minsky, M. (1961). "Steps toward Artificial Intelligence." In: *Proceedings of the IRE* 49.1, pp. 8–30.

Minsky, M. L. (1954). "Theory of Neural-analog Reinforcement Systems and Its Application to the Brain Model Problem." PhD thesis. Princeton University.

Moradi, P., Shiri, M. E., and Entezari, N. (2010). "Automatic Skill Acquisition in Reinforcement Learning Agents Using Connection Bridge Centrality." In: *Communication and Networking*. Communications in Computer and Information Science 120. Springer Berlin Heidelberg, pp. 51–62.

Mülling, K., Kober, J., Kroemer, O., and Peters, J. (2013). "Learning to Select and Generalize Striking Movements in Robot Table Tennis." In: *International Journal of Robotics Research* 3, pp. 263–279.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Cambridge, Massachusetts: The MIT Press.

Murphy, R. (2000). *Introduction to AI robotics*. Cambridge, Mass.: MIT Press.

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2004). "Autonomous Helicopter Flight via Reinforcement Learning." In: *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Niekum, S. and Barto, A. G. (2011). "Clustering via Dirichlet Process Mixture Models for Portable Skill Discovery." In: *Advances in Neural Information Processing Systems*. Vol. 24, pp. 1033–1041.

Niv, Y., Duff, M. O., and Dayan, P. (2005). "Dopamine, uncertainty and TD learning." In: *Behavioral and Brain Functions* 1.1, p. 6.

Okamura, A., Matarić, M., and Christensen, H. (2010). "Medical and Health-Care Robotics." In: *IEEE Robotics Automation Magazine* 17.3, pp. 26–37.

Oudeyer, P.-Y. and Kaplan, F. (2007). "What is Intrinsic Motivation? A Typology of Computational Approaches." In: *Frontiers in Neurorobotics* 1.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. (2007). "Intrinsic Motivation Systems for Autonomous Mental Development." In: *IEEE Transactions on Evolutionary Computation* 11.2, pp. 265–286.

Pang, S., Chen, C., and Wei, T. (2009). "A Realtime Clique Detection Algorithm: Time-Based Incremental Label Propagation." In: *3rd International Symposium on Intelligent Information Technology Application*, pp. 459–462.

Parr, R. and Russell, S. (1997). "Reinforcement Learning with Hierarchies of Machines." In: *Advances in Neural Information Processing Systems 10*. MIT Press, pp. 1043–1049.

Perkins, T. J. and Precup, D. (1999). *Using Options for Knowledge Transfer in Reinforcement Learning*. Tech. rep. 99-34. Amherst: University of Massachusetts.

Peters, J., Mülling, K., and Altun, Y. (2010). "Relative Entropy Policy Search." In: *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. Atlanta, Georgia, USA: AAAI Press.

Peters, J. and Schaal, S. (2007). "Reinforcement learning by reward-weighted regression for operational space control." In: *Proceedings of the 24th International Conference on Machine Learning*, pp. 745–750.

Pickett, M. and Barto, A. G. (2002). "PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning." In: *Proceedings of the 19th International Conference on Machine Learning*. Morgan Kaufmann, pp. 506–513.

Precup, D. (2000). "Temporal Abstraction in Reinforcement Learning." PhD thesis. Amherst, MA: University of Massachusetts.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc.

Raghavan, U., Albert, R., and Kumara, S. (2007). "Near linear time algorithm to detect community structures in large-scale networks." In: *Physical Review E* 76.3, p. 036106.

Randløv, J. and Alstrøm, P. (1998). "Learning to Drive a Bicycle using Reinforcement Learning and Shaping." In: *Proceedings of the 15th International Conference on Machine Learning*. Madison, Wisconsin, USA, pp. 463–471.

Riedmiller, M., Gabel, T., Hafner, R., and Lange, S. (2009). "Reinforcement learning for robot soccer." In: *Autonomous Robots* 27.1, pp. 55–73.

Ring, M. B. (1997). "CHILD: A First Step Towards Continual Learning." In: *Machine Learning* 28.1, pp. 77–104.

Rummery, G. A. and Niranjan, M. (1994). *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. 166. Cambridge University, Engineering Department.

Sacerdoti, E. D. (1974). "Planning in a hierarchy of abstraction spaces." In: *Artificial Intelligence* 5.2, pp. 115–135.

Schembri, M., Mirolli, M., and Baldassarre, G. (2007). "Evolution and learning in an intrinsically motivated reinforcement learning robot." In: *Proceedings of the 9th European Conference on Advances in Artificial Life*. ECAL'07. Springer-Verlag, 294–303.

Schmidhuber, J. (1991). "Curious Model-Building Control Systems." In: *Proceedings of the International Joint Conference on Neural Networks*. Singapore: IEEE, pp. 1458–1463.

Shi, J. and Malik, J. (2000). "Normalized Cuts and Image Segmentation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8, pp. 888–905.

Silver, D. L., Yang, Q., and Li, L. (2013). "Lifelong Machine Learning Systems: Beyond Learning Algorithms." In: *2013 AAAI Spring Symposium Series*.

Simon, H. A. (1996). *The sciences of the artificial*. Cambridge, Mass.: MIT Press.

Şimşek, Ö. and Barto, A. G. (2004). "Using relative novelty to identify useful temporal abstractions in reinforcement learning." In: *Proceedings of the 21st International Conference on Machine learning*. Banff, Alberta, Canada, pp. 751–758.

Şimşek, Ö. and Barto, A. G. (2009). "Skill characterization based on betweenness." In: *Advances in Neural Information Processing Systems (NIPS)*. Vol. 22, pp. 1497–1504.

Şimşek, Ö. and Barto, A. G. (2006). "An intrinsic reward mechanism for efficient exploration." In: *Proceedings of the 23rd International Conference on Machine learning*. ICML '06. ACM, 833–840.

Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). "Identifying useful subgoals in reinforcement learning by local graph partitioning." In: *Proceedings of the 22nd International Conference on Machine Learning*. Bonn, Germany: ACM, pp. 816–823.

Singh, S. P. (1992a). "Reinforcement Learning with a Hierarchy of Abstract Models." In: *In Proceedings of the 10th National Conference on Artificial Intelligence*. MIT/AAAI Press, pp. 202–207.

Singh, S. P. (1992b). "Transfer of Learning by Composing Solutions of Elemental Sequential Tasks." In: *Machine Learning*, pp. 323–339.

Singh, S. P., Barto, A. G., and Chentanez, N. (2004). "Intrinsically motivated reinforcement learning." In: *Proceedings of the 18th Annual Conference on Neural Information Processing Systems (NIPS)*. Vancouver, B.C., Canada.

Singh, S. P., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms." In: *Machine Learning* 38.3, pp. 287–308.

Singh, S. P., Lewis, R., Barto, A., and Sorg, J. (2010). "Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective." In: *IEEE Transactions on Autonomous Mental Development* 2.2, pp. 70–82.

Skinner, B. (1938). *The Behavior of Organisms: An Experimental Analysis*. The Century Psychology Series. Appleton-Century-Crofts.

Soni, V. and Singh, S. (2006). "Reinforcement learning of hierarchical skills on the Sony Aibo robot." In: *Proceedings of the 5th International Conference on Development and Learning*.

Speelman, C. and Kirsner, K. (2005). *Beyond the Learning Curve*. Oxford University Press.

Srinivasan, R. (2002). *Importance Sampling: Applications in Communications and Detection*. Springer.

Stolle, M. and Precup, D. (2002). "Learning Options in Reinforcement Learning." In: *Abstraction, Reformulation, and Approximation*. Lecture Notes in Computer Science 2371. Springer Berlin Heidelberg, pp. 212–223.

Stout, A. and Barto, A. G. (2010). "Competence Progress Intrinsic Motivation." In: *Proceedings of the 9th IEEE International Conference on Development and Learning*. Ann Arbor, Michigan, pp. 257–262.

Sutton, R. S. (1988). "Learning to Predict by the Methods of Temporal Differences." In: *Machine Learning* 3.1, pp. 9–44.

Sutton, R. S. and Barto, A. G. (1981). "Toward a modern theory of adaptive networks: Expectation and prediction." In: *Psychological Review* 88.2, pp. 135–170.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press.

Sutton, R. S., Koop, A., and Silver, D. (2007). "On the role of tracking in stationary environments." In: *Proceedings of the 24th International Conference on Machine Learning*. ACM, pp. 871–878.

Sutton, R. S., Singh, S., Precup, D., and Ravindran, B. (1999a). "Improved Switching among Temporally Abstract Actions." In: *Advances in Neural Information Processing Systems 11*. MIT Press, pp. 1066–1072.

Sutton, R. S., Precup, D., and Singh, S. (1999b). "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning." In: *Artificial Intelligence* 112, pp. 181–211.

Taylor, M. and Stone, P. (2009). "Transfer Learning for Reinforcement Learning Domains: A Survey." In: *Journal of Machine Learning Research* 10, pp. 1633–1685.

Tenenbaum, J. B., Silva, V. D., and Langford, J. C. (2000). "A Global Geometric Framework for Nonlinear Dimensionality Reduction." In: *Science* 290.5500, pp. 2319–2323.

Tesauro, G. (1995). "Temporal difference learning and TD-Gammon." In: *Communications of the ACM* 38.3, pp. 58–68.

Thorndike, E. L. (1911). *Animal Intelligence: Experimental Studies*. New York: The Macmillan company.

Thrun, S. (1995). "Learning To Play the Game of Chess." In: *Advances in Neural Information Processing Systems 7*. The MIT Press, pp. 1069–1076.

Thrun, S. (1996). "Is Learning The n-th Thing Any Easier Than Learning The First?" In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 640–646.

Thrun, S. and Mitchell, T. M. (1995). "Lifelong Robot Learning." In: *The Biology and Technology of Intelligent Autonomous Agents*. 144. Springer Berlin Heidelberg, pp. 165–196.

Thrun, S. and Schwartz, A. (1995). "Finding Structure in Reinforcement Learning." In: *Advances in Neural Information Processing Systems 7*. MIT Press, pp. 385–392.

van Eck, N. J. and van Wezel, M. (2008). "Application of reinforcement learning to the game of Othello." In: *Computers & Operations Research* 35.6, pp. 1999–2017.

Vigorito, C. and Barto, A. (2010). "Intrinsically Motivated Hierarchical Skill Learning in Structured Environments." In: *IEEE Transactions on Autonomous Mental Development* 2.2, pp. 132–143.

von Luxburg, U. (2007). "A tutorial on spectral clustering." In: *Statistics and Computing* 17.4, pp. 395–416.

von Luxburg, U. and Schölkopf, B. (2011). "Statistical Learning Theory: Models, Concepts, and Results." In: *Handbook of the History of Logic* 10, pp. 706–751.

Wagner, D. and Wagner, F. (1993). "Between Min Cut and Graph Bisection." In: *Mathematical Foundations of Computer Science*. Ed. by A. M. Borzyszkowski and S. Sokolowski. 711. Springer, pp. 744–750.

Watkins, C. (1989). "Learning from Delayed Rewards." PhD thesis. Cambridge University, England.

Watkins, C. and Dayan, P. (1992). "Q-Learning." In: *Machine Learning* 8.3-4, pp. 279–292.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., and Thelen, E. (2001). "Autonomous Mental Development by Robots and Animals." In: *Science* 291, pp. 599–600.

White, R. W. (1959). "Motivation reconsidered: the concept of competence." In: *Psychological review* 66, pp. 297–333.

Whiteson, S. (2012). "Evolutionary Computation for Reinforcement Learning." In: *Reinforcement Learning: State of the Art*. Berlin, Germany: Springer, pp. 325–358.

Yekutieli, Y., Sagiv-Zohar, R., Aharonov, R., Engel, Y., Hochner, B., and Flash, T. (2005). "A dynamic model of the Octopus arm. I. Biomechanics of the Octopus reaching movement." In: *Journal of Neurophysiology* 5, pp. 291–323.