AUTONOMOUS ABSTRACTION OF POLICIES BASED ON POLICY HOMOMORPHISM

by

SRIVIDHYA RAJENDRAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2009

DEDICATION

This thesis is dedicated to my parents, T.K. Rajendran and Bhagyalakshmi, my grandparents, T.M. Krishnaswami Iyer, Parvathi Iyer, M.V Subramaniam, and Visalakshi Iyer, and my husband Prashant.

ACKNOWLEDGEMENTS

I am very grateful to my thesis advisor Dr. Manfred Huber. I wouldn't have embarked upon and successfully completed this journey without his constant guidance, support and patience. He was always readily available to have discussions during my graduate studies at UTA. He has taught me how to be a better researcher and to express my ideas. I will always strive to emulate his qualities as a teacher and a researcher. Thank you Dr. Huber.

I would like to thank Dr. Zaruba, Dr. Kamangar, Dr. Gao and Dr. Peterson for their inputs as my dissertation committee members. I would like to extend my gratitude to all the CSE faculty members for their positive influences. I would also like to thank CSE staff members Pam McBride, Camille Costabile and Will Griffith for their help and support.

My stay in the graduate school at UTA was made pleasant and memorable by my friends who acted as my extended family members here in USA. I would like to thank my AI Lab members for their discussions and moral support. I would like to thank Dr. Chen, Dr. Corley, Dr. Rosenberger and Dr. Kim from IMSE department for treating me and Prashant as their own family members and helping us whenever needed. I would also like to thank the members of COSMOS Lab in IMSE department for the fun times we had and for making my memories at UTA cherishable.

My parent's support and love have played an instrumental role in this achievement. Their confidence in my abilities and hard work have always provided me with the necessary impetus to succeed. I would like to thank my sister Geetha, brother Krishnaswami and my

brother in law Sanket for their love, support, and guidance throughout this journey. I would like to thank my in laws for supporting me in achieving my dreams. Last but not the least; I would like to thank Prashant, my husband without whom I cannot imagine this endeavor. Thanks dear for your endless support, encouragement, and patience. You always believed in me and I could not have done this without you.

July 16, 2009

ABSTRACT

AUTONOMOUS ABSTRACTION OF POLICIES BASED

ON POLICY HOMOMORPHISM

Srividhya Rajendran, PhD.

The University of Texas at Arlington, 2009

Supervising Professor: Manfred Huber

A life long learning agent performing in a complex and dynamic environment needs the ability to learn increasingly complex tasks over time. These agents over their lifetime have to learn new tasks, adapt the policies of already learned tasks and extract and reuse the knowledge gained to learn new, more complex tasks. To do this, they need methods that allow them to autonomously extract knowledge from the already learned policy instances and reuse the knowledge gained to learn related tasks in novel environments.

This dissertation presents a novel approach that enables an agent to autonomously abstract reusable skills and concepts using policy instances of a similar *task type* and use the resulting abstractions to learn related tasks in novel situations. To achieve this, this work formalizes a novel idea of policy homomorphism that allows autonomous extraction of general policies for *task types*. Each extracted general policy is here an abstract policy that is

homomorphic to the set of specific policy instances of the corresponding *task type* that it is derived from and is made up of abstract states that identify situations in which the given policy is applicable and abstract actions that identify actions that need to be performed in those situations. Once extracted, the generalized policies are reused in new contexts to address related tasks by adding them as higher level actions that the agent can choose to perform.

To facilitate the autonomous abstraction of a policy of a given *task type* from a set of policies, the agent has to identify and categorize policies for various tasks into different *task types*. To achieve this the policy generalization approach presented here employs a utility-based criterion that enables the agent to autonomously categorize and generalize a set of situation-specific policies of different *task types* into a set of general policies containing one general policy for each identified *task type* using the policy homomorphism framework.

To demonstrate the working of this policy generalization method we show the abstraction of a general policy for a specific *task type* using two sets of policies of different *task types* in a grid world domain and further show how the abstracted general policies can be used to learn related tasks in novel grid world environments. Further, to demonstrate the working of the utility based criterion to identify *task types* and autonomously abstract general policies for the identified *task types* we show the abstraction of general polices using the utility criterion from a set of situation-specific policies of different *task types* in a grid world domain.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS…… iv

ABSTRACT…… vi

LIST OF ILLUSTRATIONS…….. xi

LIST OF TABLES…….. xiv

Chapter Page

1. INTRODUCTION ……..…… 1

1.1 Problem Description ……... 2

1.2 Approach Presented ……..…… 4

1.3 Overview of Dissertation ……. 5

2. RELATED WORK ……..…… 6

2.1 Temporal and State Abstractions …… 7

2.1.1 Temporal Abstractions in Reinforcement Learning……. 9

2.1.2 State Abstractions in Reinforcement Learning……. 12

2.2 Hierarchical Reinforcement Learning …… 13

3. TECHNICAL BACKGROUND AND NOTATION ……. 16

3.1 Reinforcement Learning ……..……. 16

3.1.1 Markov Decision Process ……..……. 17

3.1.2 Q-Learning ..……..……. 19

3.1.3 Exploration vs. Exploitation Strategies …… 21

3.2 Temporally Abstract Reinforcement Learning ..…… 23

3.2.1 Semi Markov Decision Process ……..… 25

3.2.2 Options 25

3.2.3 SMDP Learning 26

3.3 MDP Homomorphism 28

3.3.1 Bounded Parameter MDP 32

3.3.2 ε - Reduction Method 34

4. POLICY HOMOMORPHISM FRAMEWORK 38

4.1 Goal Based Policy 39

4.2 Policy Homomorphism 40

4.2.1 Partial Policy Homomorphism 44

4.2.2 Approximate Policy Homomorphism 45

4.2.3 Goal Based Policy Homomorphism 47

5. UTILTY OF A POLICY 48

5.1 The Utility of a Policy 49

5.2 Incremental Utility and the Utility of an Action/Policy Set 50

5.3 Utility-Based *Task Type* Identification and Policy Generalization 52

5.4 Controlling Action Set Size 53

6. LEARNING FRAMEWORK 55

6.1 Learning Component 56

6.2 Skills and Concepts Memory 57

6.3 Policy Abstraction Component 57

6.3.1 Parametric Mapping Functions 58

6.3.2 Policy Abstraction 59

7. EXPERIMENTS AND RESULTS 62

7.1 Experiments in Deterministic Grid World Domains 63

7.1.1 Learning Basic Policies 64

7.1.2 Extracting a Generalized Policy ……. 65

7.1.3 Reusing Generalized Policies …….. 69

7.2 Experiments in Non-Deterministic Grid World Domains ……..…. 72

7.2.1 Learning Basic Policies ……..……. 73

7.2.2 Extracting a Generalized Policy ……..…….. 74

7.2.3 Reusing Generalized Policies …….. 74

7.3 Autonomous Categorization and Generalization of Policies Based on *Task Types* ……..…. 78

7.3.1 Autonomous Categorization and Generalization of Policies Based on *Task Types* in a Deterministic Environment ……..……. 79

7.3.2 Autonomous Categorization and Generalization of Policies Based on *Task Types* in a Non-Deterministic Environment ……..……. 82

7.3.3 Learning "CLEAN ROOMS" Task in a Three-Dimensional Non-Deterministic Grid World ……. ……..…. 86

8. CONCLUSION AND FUTURE WORK ……..……. 90

REFERENCES……. 93

BIOGRAPHICAL INFORMATION……..…… 102

LIST OF ILLUSTRATIONS

Figure		Page
3.1	Reinforcement Learning Model ……	17
3.2	"Move Trash to Trash Can" Task using Simple Actions and Macro-Operators……	24
3.3	(a) A Grid World Problem. (b) A Reduced Model of the Grid World ……	28
3.4	(a) A Simple Grid World Environment with Multiple Similar Rooms. (b) The Option MDP Corresponding to a Get-Object-and-Leave-Room Option ……	31
3.5	State Transition Diagram for a BMDP ……	33
4.1	Goal Based Policy ……	39
4.2	7×7 Grid World with Two Doors A and B ……	40
4.3	(a) Learned Policy for "REACH Door A", (b) Learned Policy for "REACH Door B" ……	41
4.4	Generalized Policy for "REACH DOOR" ……	42
4.5	New Grid World with Areas where Generalized Policy is Applicable ……	42
4.6	Policy Mappings from Base Policies A and B to Generalized Policy ……	44
6.1	Learning Framework ……	56
7.1	17X17 Deterministic Grid Worlds used for Learning Situation-Specific Policies for Doorways ……	64
7.2	(a) One of the Trajectories Followed by the Agent under the Situation-Specific Policy Learned for the Doorway in Grid World 1 and (b) One of the Trajectories Followed by the Agent under the Situation-Specific Policy Learned for One of the Doorways in Grid World 2 ……	66
7.3	A Subset of the Abstracted States and Actions of the Extracted General Policy Using the Five Situation-Specific Policies to Reach Individual Doorways……	68

7.4 20×20 Deterministic Grid Worlds used to Evaluate Generalized Policy 70

7.5 Learning Curves with and without Using Generalized Policy to Reach (a) the Goal Door in Grid World 1 of Figure 7.4, (b) the Goal Door in Grid World 2 of Figure 7.4, and , (c) the Goal Door in Grid World 3 of Figure 7.4 71

7.6 17×17 Non-Deterministic Grid Worlds used for Learning Situation-Specific Policies for Doorways 73

7.7 20×20 Non-Deterministic Grid World used to Evaluate Generalized Policy 75

7.8 Learning Curves with and without Using Generalized Policy to Reach (a) the Goal Door in Grid World 1 of Figure 7.7, (b) the Goal Door in Grid World 2 of Figure 7.7, and (c) the Goal Door in Grid World 3 of Figure 7.7 76

7.9 20×20 Non-Deterministic Grid World used for "CLEAN ROOMS" Task..... 77

7.10 Learning Curves for "CLEAN ROOMS" Task with and without Using the Generalized "REACH DOOR" Policy 78

7.11 Grid Worlds Used for Learning Situation-Specific Policies For "Task1" and "Task2" 79

7.12 Parts of the State Space Around the Goal Doors Abstracted in the Generalized Policy GP1 (Regions Shown in Grey) 81

7.13 Parts of the State Space Around the Goal Doors Abstracted in the Generalized Policy GP2 (Regions Shown in Yellow)..... 81

7.14 Parts of the State Space Around the Goal Doors Abstracted in the Generalized Policy GP3 (Regions Shown in Pink) 82

7.15 Non-Deterministic Grid Worlds used for Learning Situation-Specific Policies for "Task 1" and "Task2" 83

7.16 Parts of the State Space Around the Goal Doors Abstracted in the Approximate Generalized Policy GP1 (Regions Shown in Green) 84

7.17 Parts of the State Space Around the Goal Doors Abstracted in the Approximate Generalized Policy GP2 (Regions Shown in Grey) 85

7.18 Parts of the State Space Around the Goal Doors Abstracted in the Approximate Generalized Policy GP3 (Regions Shown in Orange) 85

7.19 Grid Worlds used to Learn Situation-Specific Policies to Reach Doorways, Pick Objects, and Drop Objects 86

7.20 Three-Dimensional Non-Deterministic Grid World Domain 87

7.20 Learning Curves for "CLEAN ROOMS" Task in the Three-Dimensional Grid World Domain 88

LIST OF TABLES

Table		Page
3.1	Q- Learning Algorithm ……	20
6.1	Partial Policy Generalization Algorithm ……	60
6.2	Approximate Policy Generalization Algorithm ……	61
7.1	States in One of the Trajectories Followed Under the Learned Policy with its Corresponding Abstract States in General Policy and their Respective State Mapping Function. ……	68
7.2	Few of the Primitive Concepts Captured in the State Mapping Functions ……	69

CHAPTER 1

INTRODUCTION

One of the fundamental capabilities of human and intelligent beings is their ability to abstract and apply learned knowledge beyond the specific situation and environment in which the knowledge was acquired. In humans, this capability can be seen at various levels, ranging from basic situation transfer where, for example, most experience in grasping objects has been generalized to allow never before seen object to be picked up in the first attempt., to task-level transfer where, for example, skills learned playing tennis are transferred to playing squash, and up to abstract transfer where the essence of learned skills and concepts is transferred to abstract, non-physical domains through metaphorical extension.

While the behavioral forms of these abstraction capabilities are essential for the performance and potentially for the survival of humans and animals who have to operate in highly dynamic and changing environments, they are virtually absent in the current artificial intelligence learning agent which mostly restrict abstraction to simple state or temporal abstraction with only limited capabilities to transfer learned skills beyond a very limited range of situations and environments.

To start addressing this, this dissertation presents a novel framework and algorithms that allow reinforcement learning (RL) agents to abstract learned skills that transfer to new environments and can be applied in new contexts to facilitate the learning of substantially more complex tasks. The goal here is to provide life-long learning agents not only with the ability to

reason in terms of more abstract concepts, but also to allow the agent's operational and learning capabilities to scale to significantly larger, more dynamic and more complex domains and eventually to real world.

1.1 Problem Description

A reinforcement learning (RL) agent learns to successfully perform a task by learning a optimal policy for it. These agents mostly model their environment as markov decision processes (MDP) and use well established RL algorithms to solve these models. To make these RL agents perform successfully in the real world they not only need the ability to learn tasks but also need capabilities that allow them to extract knowledge from seen instances and reuse it to solve problems in unknown and unseen situations.

Traditional RL approaches encounter a number of limitations when learning to perform complex tasks in real world environments. The policies learned using these traditional approaches do not transfer, as a result the agent has to learn a separate policy for every task and environment instance. As the life of this RL agent increases the number of policies it would have learned also increases. Storing these becomes intractable. Further these learned policies are of limited use as the agent will almost never encounter two identical scenarios of the same task. Secondly, each state the agent is in is represented by a complete enumeration of all variables of the environment. As the number of variables increases, the size of the state space and as a consequence the time needed to learn a policy becomes intractable. Finally, traditional RL agents make decisions about the actions they need to perform to complete a task at every time step. As the complexity of the task increases reasoning about what action to take at each time step and thus operating in real-time becomes computationally intractable.

Biological systems like humans tackle similar problems. Humans are continuous learning beings and over a period of time they learn to perform increasingly complex tasks in a highly complex and dynamic environment. Developmental theories [Piaget 1952] [Lakoff 1987] [Mandler 1992] suggest they accomplish this by learning very early on in their development to abstract important information for a given task while ignoring the rest. They, as a part of this, learn to abstract skills and use them to learn to perform increasingly complex tasks. For example, while playing tennis we only decide the way to hit the ball by planning to play different strokes without the need to reason about each muscle movement individually which would make the task of playing tennis intractable. Similarly, while playing tennis we only pay attention to the environment variables that are important to playing tennis, such as the location of the ball, its speed, our own and the opponent's position etc. while ignoring the rest which again makes playing tennis possible. Also, we use the knowledge gained while playing tennis to learn games like badminton, this kind of knowledge reuse allows to learn novel tasks faster and also aids us in learning increasing complex tasks over time.

AI agents and robots need capabilities similar to those of biological systems. In particular, they need the ability to

1. Form useful skill and representational abstractions.
2. Identify similar tasks and situations.
3. Apply the knowledge gained in previous instances of a given task to novel situations of related tasks.

These capabilities would allow AI agents and robots to reason at a higher level, learn new tasks in a shorter amount of time, reuse knowledge gained, and derive more compressed state representations on which to learn.

This dissertation mainly concentrates on developing a novel method that allows a life-long learning RL agent to autonomously form useful skill and representational abstractions. Further we show how the skill and representational abstractions formed from a set of given *task types* can be used to learn related tasks in novel situation.

1.2 Approach Presented

The work presented in this dissertation presents a new approach for transfer learning in RL agents. It allows the agent to learn new tasks, autonomously abstract useful skills and concepts from the policy instances of a *task type* and reuse the extracted skills and concepts to learn related tasks in novel contexts. To achieve this we formalize a novel policy homomorphism framework. The learning component of the RL agent uses this policy homomorphism framework to autonomously generalize a set of previously learned, situation-specific policies for similar tasks into an abstract policy for the corresponding *task type*. A *task type* is defined here by a maximal set of policy instances for which a general homomorphic policy can be found. Each general policy abstracted by the learning framework is represented by two sets of functions where the first set of functions maps the individual states of each policy to unique states of the abstract policy and the second set of functions maps individual actions from each base policy to actions of the abstract policy. Once the agent has abstracted a general policy, it can use it to directly address similar tasks in novel situations or to learn new complex tasks. This is achieved by adding the generalized policy to the agent's action set, allowing the agent to choose it from states where this general policy is applicable. We demonstrate the working of this policy generalization method and the potential of subsequent reuse of the general policy on a set of deterministic and non-deterministic grid world domain examples.

We further extend our policy generalization approach by defining a criterion that enables the agent to autonomously identify and categorize a set of policies into sets of policies

of similar *task type* which are then used to abstract skills and concepts important to successfully achieve tasks of these *task types* using the policy homomorphism framework. To demonstrate the potential of this extended policy generalization method, we show the abstraction of a general polices for a specific *task types* using a set of policies of different *task types* in a grid world domain.

1.3 Overview of Dissertation

The remainder of this dissertation presents a new method for autonomous skill and representational abstraction and methods to reuse them to learn related tasks in novel contexts. The next chapter presents the related work. Chapter 3 explains the formalism and the technical aspects of the main related work used in this dissertation. Chapter 4, and Chapter 5 present the main technical contribution of this thesis. Chapter 4 presents the policy homomorphism framework which is used to autonomously abstract skills and representations. Chapter 5 presents a novel method to derive the utility of a policy or a set of policies based on the amount of decision reduction resulting from its use. This allows the agent to autonomously identify and categorize task and policy instances into *task types* and facilitates the construction of a set of admissible higher level actions from the set of learned skills. Chapter 6 presents the learning framework used in this dissertation. Chapter 7 presents experimental results of this work. Chapter 8 presents conclusions and proposes directions for future work.

CHAPTER 2

RELATED WORK

An RL agent that needs to perform successfully in a complex dynamic environment has to have methods that allow it to reuse knowledge gained from already learned tasks to learn tasks in novel environments. For this, the assumption is that the new task contains some elements or characteristics of the previously learned tasks. The problem with this assumption is that most often the details of the state space of the new problem and new environment are very different from the ones of the state space and environment of the older problem. As a result, the policies learned for the old tasks that are directly tied to the state space of the older problem are not simply and directly transferable to learn new similar or related tasks in a new environment with a different state space. Another problem in regards to the state space in real world systems is that the agent uses its percepts to determine the state of the environment. The amount of data produced by these percepts is huge and as a result processing and basing decision on this huge amount of data is intractable. Addressing this requires methods that would allow the agent to focus its attention on the parts of the data that are important for task completion while ignoring the rest. However, while most of the time just focusing attention on state variables that are important for task completion would be sufficient to complete the task successfully in this environment, it may not be sufficient or relevant to perform the same task in a new environment. This is because many times the relevant information for task completion may not rely just on the perceptual features that are observable by the agent but more importantly on functional features or on a functional signature of the state that allows the task to be successfully performed in the context of objects present in the environment (i.e. their affordances [Gibson 1977]). Lastly,

making decisions about actions that an agent needs to perform at each time step becomes impossible without any prior control knowledge. As a result of these issues, life-long learning agents need methods that would allow them to reuse the knowledge gained from prior control tasks and also methods that allow them to abstract decision making to a higher level.

Over the years there has been significant interest in developing techniques that would allow RL agents to tackle the curse of dimensionality, address reusability, reason contextually, and learn and manage internal representations of new knowledge and skills without hand-crafting new structures or re-leaning from scratch. This research can be divided loosely into temporal abstractions, spatial or state abstractions, and hierarchical reinforcement learning.

2.1 Temporal and State Abstractions

Both temporal and state abstractions have been widely studied since the early days of AI in order to address the curse of dimensionality.

Amarel's paper [Amarel 1968] discussing the missionaries and cannibals problem was one of the first papers that suggested the need for abstractions in problem solving. The paper presented a series of handcrafted abstractions for the missionaries and cannibals problem. These abstractions were used to demonstrate how the use of certain abstractions could significantly reduce the search space of the problem and thus make problems easily solvable. Much of the early research on abstractions came from joint psychology and AI research were computational systems were built to further the understanding of human problem solving [Newell et al 1963][Laird et. al 1986][Minton 1988][Anzai and Simon 1979]. Other early research on temporal abstractions in AI focused on planning systems. The STRIPS planner [Fikes et al. 1972] was one of the first planning systems to make use of temporal abstractions. Later systems like ABSTRIPS [Sacerdoti 1974] could automate some of the generation of planning hierarchies.

Other planning systems like ALPINE [Knoblock 1990][Knoblock 1991] automatically generated planning hierarchies by dropping literals from the goal descriptions in an ordered manner to create a more abstract space in which the problem was solved. Korf's [Korf 1985 a] planning domain automatically created open-loop macro operators that were designed to abstract over non serializable subgoals for use in a means-ends problem solver. Although these planning systems automatically generated abstractions, their main drawbacks were that they focused on creating abstractions in deterministic and completely observable environments. However, they introduced the concepts of temporal and state abstraction in order to make more complex problem tractable.

Temporal abstractions are methods that allow an agent to abstract the control knowledge gained from previous experiences of similar tasks to higher level actions also sometimes known as options, skills or behaviors [Sutton et al. 1999] [Thrun and Schwartz 1995] [Brooks 1986] [Huber and Grupen 1997]. These higher level actions take multiple time steps to complete. Each higher level action encapsulates multiple complex lower level actions which each take a single time step to complete. For example, if we consciously had to plan our muscle movements every time we wanted pick up a mug filled with coffee or tea to drink from it, the planning necessary for drinking the whole cup of coffee or tea would be nearly impossible. Instead, we create an abstraction that encapsulates the entire set of muscle movements that we use to pick up a mug; similarly for bringing the mug to our mouth and drinking from it. Temporal abstraction is extremely important for a life long learning RL agent because it allows the agent to abstract decision making to a higher level and thus allows the agent to ignore details about decision making at each time step. Temporal abstraction allows the agent to learn complex tasks over time as it shortens the effective depth that the agent must search to find a solution.

State abstractions are methods that allow a learning agent to perform abstraction that generalizes or aggregates over state variables [Parr 1998]. For example, if we have learned to "pick up mug" and also learned to "pick up jug" then we generalize across the policy to "pick up jug" and the policy to "pick up mug" to create a general policy that allows "pick up vessels that have handle". This general policy identifies situations in which both of these policies can be applied by only paying attention state attributes that are important for this *task type* while ignoring the rest of the state attributes. Further, it also allows the use of a general policy to pick up any vessels with handles even though we might not have learned an explicit policy for each one of them.

2.1.1 Temporal Abstraction in Reinforcement Learning

One of the advantages of using temporal abstractions in RL systems is that it allows the systems to learn more complex tasks. Most of the early work on scaling RL systems [Gullapalli 1992] [Singh 1991, 1992a,c] to solve complex problems involved training the systems on a series of related sub-problems and then using the solutions of the sub-problems as a starting point to learn solutions for harder problems. These systems showed that training the systems on sub-problems allowed the RL system to approximate solutions in cases where a solution would have been very difficult to find otherwise. One of the main limitations of these approaches is that they require the training sequence to be determined a priori by the programmer.

[Singh 1992 b][Singh 1994] present the H-DYNA algorithm that uses closed loop temporal abstractions to facilitate learning in the RL framework. Bradtke and Duff's [Bradtke and Duff 1995] work presents an SMDP learning algorithm, which learns action values for variable duration actions by backing up the discounted sum of reward received while the action was executing. McGovern et al.'s [McGovern et al. 1997] Macro-Q learning combines the SMDP backup for variable duration actions with the Q-learning backup for primitive actions to learn

action values for both primitive actions and options. Other works that use abstractions create hierarchies of states or actions. [Kaebling 1993][Dayan and Hinton 1993] both present methods that create a hierarchy of value functions based on attributes of the state space and are able to accelerate learning compared to flat systems. [Moore et al. 1999] extended this work to automatically generate hierarchies in goal directed systems. [Theocharous and Mahadevan 2002] demonstrate that hierarchical partially observable MDP's can enable a robot to successfully solve more difficult tasks that are not possible without hierarchy. These works mainly concentrate on restricting the search space of the RL agent to decrease the time needed to approximate an optimal solution.

Much of the recent work and theoretical foundation for the use of temporal abstraction in RL is provided by [Precup et al. 1998] [Sutton et al. 1998] [Sutton et al. 1999]. They show that use of temporal abstractions transforms an MDP into an SMDP and that convergence results still hold for the learning algorithms known to converge in the absence of abstraction. They use the options framework and SMDP learning [Sutton et al. 1998] [Sutton et al. 1999] [Precup 2000] to represent temporal abstractions and learn using them to solve problems. The options framework and SMDP learning are discussed in detail in Chapter 3. [Hauskrecht et al. 1998] present how the use of localized temporal abstractions can generate smaller and more abstract MDPs. They further showed that these new MDPs can be solved more quickly than the original MDP's and that the temporal abstractions help to facilitate knowledge transfer across tasks. Together this showed the usefulness of abstractions in a RL system.

[Dietterich 2000] presents the MAXQ algorithm that provides an alternative framework to temporal abstraction in RL. In this work the system uses a fixed hierarchy and each action at each level of the hierarchy has a separate value function. Using this framework this work provides convergence results for MAXQ learning.

None of these methods, however, provide methods for automatically creating new temporal abstractions. To do this, [Thrun and Schwartz 1995] present the SKILLS algorithm. The SKILLS algorithm can extract a pre-specified number of action sequences that are common across the task by examining the optimal policies within the set of related MDP's . These action sequences can be useful in other tasks that share this state space. [Bernstein 1999] presents a similar system that uses optimal policies for given tasks to generate a single new temporal abstraction that is called a "reuse option". This is achieved by examining the probabilities of taking each action in each state for a given optimal policy and the action distribution of each state in the reuse option is then formed by averaging the action probabilities for each of the optimal policies for that state.

Another method to automatically create new temporal abstractions in reinforcement learning framework is presented by Digney's Nested Q-Learning algorithm [Digney 1996] [Digney 1998]. This work creates new options online while the agent is learning using Q-learning. This work creates online options by examining the frequency of state visitations and the reward gradient at each state and assigning the states with high frequency of visits and reward gradient as the sub-goals for the new options. This work creates a hierarchy of actions for agents as newer actions can call existing actions as subroutines. [McGovern and Barto 2001] also create options by automatically discovering sub-goals. Their work uses diverse density to discover useful sub-goals by keeping track of successful and unsuccessful events. However in the case of complicated environments and rewards it becomes difficult to accumulate and classify sets of successful and unsuccessful trajectories needed to compute the density measures. In addition, these methods do not allow the agent to discover sub-goals that are not explicit part of the tasks used in the process of discovering them. Goel and Huber's [Goel and Huber 2003] work focuses on discovering subgoals by searching a learned policy model for certain structural properties.

Their method discovers sub-goals that are not even part of the successful trajectories of the policy. The agent can then learn policies for these sub-goals which in turn are added as options to use them for effective exploration as well as to accelerate learning in other tasks in which the same sub-goals are useful.

2.1.2 State Abstraction in Reinforcement Learning

The work on the state abstractions that this dissertation is most strongly related to, is the one of [Dean et al.1997], [Ravindran and Barto 2002], [Ravindran and Barto 2003] and [Jong and Stone 2005]. Dean, Givan and Leach introduce an abstraction method known as bounded parameter MDP (BMDP) that provides a mechanism to derive state space partitions of an, MDP that ensure approximately optimal policies to be learned. These partitions depend on the action space and the particular reward function of the task and can be used to learn a policy that is within an ε-dependent quality bound [Kim and Dean 2003]. [Ravindran and Barto 2002][Ravindran and Barto 2003] present a new framework for flexible skill transfer. They develop a formal framework to condense the state space and facilitate value function and policy transfer by exploiting state space redundancies and symmetries based on MDP homomorphisms. They further extend their method to SMDPs by defining SMDP homomorphisms, providing a framework to identify situations in which policies can be transferred. However, their work requires that abstracted SMDPs and initial base MDPs have the same properties under all possible policies, limiting its applicability to virtually identical environments. A detailed explanation of BMDP and MDP homomorphism is presented in Chapter 4. Jong and Stone [Jong and Stone 2005] present an approach that autonomously aggregates states based on policy irrelevance of state attributes to allow for a more flexible skill transfer. While this abstracts state representations in a task specific way, it is predicated on carefully engineered initial feature sets and exact policy transfer, limiting its applicability to relatively narrowly defined task domains and environment sets.

The most recent work that is closely related to this dissertation is the work of Wolf and Barto. Their work is an extension to the Ravindran and Barto's work [Ravindran and Barto 2003]. In their work Wolfe and Barto [Wolfe and Barto 2006] identify object (or environment) types based on MDP homomorphisms in the presence of an additional state property. Their method considers two objects to be of the same type if in the presence of these objects all possible policies are homomorphic and achieve the same result in terms of the chosen property. One limitation of this method is that it considers two object types similar only if all functionalities of the two objects are similar. Further, due to the requirement for a local SMDP homomorphism, it has to assume that both objects are in an identical environment. If the environment changes, two similar objects might no longer appear similar.

2.2 Hierarchical Reinforcement Learning

Over the years AI researchers have had an elevated interest in hierarchical reinforcement learning (HRL). HRL allows the agent to use methods that allow them to do hierarchical modeling and control of POMDPs. This is because hierarchical representation allows for better addressing of problems like dimensionality, uncertainty and reusability etc. The main end goal of HRL is to develop methods that allow the agents to build abstractions autonomously over their life span based on their experiences so that they can reuse these abstractions which reflect the gained knowledge to learn new complex tasks without the need for hand crafting new structures or relearning from scratch. HRL approaches are derived from traditional RL approaches and temporal and state abstraction methods to solve problems [Fikes et al. 1972] [Sacerdoti 1974] [Korf 1985 b]. [Parr and Russell 1997][Parr 1998] proposed a framework known as hierarchy of machines (HAM) for temporal abstractions. [Andre and Russell 2001] extend this to include programmable HAMs. HAMs exploit the theory of SMDPs, but the emphasis is on simplifying complex MDPs by restricting the class of realizable policies rather

than expanding the action choices. [Dietterich 2000] developed another approach to hierarchical RL called the MAXQ Value Function Decomposition. Like options and HAMs, this approach relies on the theory of SMDPs. Unlike options and HAMs, however, the MAXQ approach does not rely directly on reducing the entire problem to a single SMDP. Instead, a hierarchy of SMDPs is created whose solutions can be learned simultaneously [Barto and Mahadevan 2003].

Some of the recent work in hierarchical reinforcement learning includes [Bakker and Schmidhuber 2004] [Konidaris and Barto 2006] [Huber and Asadi 2007]. [Bakker and Schmidhuber 2004] developed a new method for hierarchical reinforcement learning. Their work learns to create both useful subgoals and the corresponding specialized subtask solvers. They learn high-level value functions that cover the state space at a coarse level and low-level value functions that cover only parts of the state space at a fine-grained level. The main limitations of the system include the large number of parameters, the lack of strict convergence guarantees and the dependence on identifying reasonable high-level observations. [Konidaris and Barto 2006] present an approach that achieves simple to complex generalization through shaping rewards based on learned value functions in agent space that are rich enough to solve the task in problem space. It acknowledges the problem of generalization and forgetting, in particular that sensor-based state space descriptors should be minimal but sufficient to describe the problem space, and that function generalization usually only works within single tasks. [Asadi and Huber 2007] present a learning architecture that transfers control knowledge in the form of behavioral skills and corresponding representation concepts from one task to subsequent learning tasks. The presented system uses the knowledge to construct a more compact state space representation for learning while assuring bounded optimality of the learned task policy by utilizing a representation hierarchy. While these techniques allow for faster learning times and more complex tasks to be addressed, the reuse of control knowledge is still largely limited to tasks in the same environment.

Even with these enhancements to traditional RL algorithms that allow for transfer learning, there are still a lot of issues that need addressing. The first one is that many of these approaches use a hand crafted feature space which requires expert knowledge for a specialized domain which is rarely available. Secondly, the abstract skills and representations learned by these methods fail to capture the contextual information that is important for transfer learning across different environments. Lastly, the macro actions or options learned that allow for the reuse of control knowledge are still largely limited to tasks in the same or identical environment.

CHAPTER 3

TECHNICAL BACKGROUND AND NOTATION

This chapter introduces the reinforcement learning framework [Barto et al. 1981] [Sutton 1988] [Kaelbling et al. 1996] [Sutton and Barto 1998] , and the options framework [Sutton et al. 1998] [Sutton et al. 1999] with particular attention paid to the aspects that we use in this dissertation. We also give a brief introduction to the MDP homomorphism framework [Ravindran and Barto 2002][Ravindran and Barto 2003] developed by Ravindran and Barto to extract a smaller state space from the original one by exploiting its symmetries and redundancies.

3.1 Reinforcement Learning

Reinforcement Learning (RL) [Barto et al. 1981] [Sutton 1988] [Kaelbling et al. 1996] [Sutton and Barto 1998] is a sub-area of machine learning. It is used by autonomous artificial intelligence (AI) agents to learn optimal policies for various tasks. In this learning framework the learner explores the environment by perceiving the state of the environment through its sensors and alters the state of the environment by performing subsequent actions using actuators. The environment in return provides the AI agent with reinforcement (which can be positive or negative). The AI agent uses this reinforcement to learn a policy that maximizes the expected cumulative reward over the tasks. Figure 3.1 shows the basic reinforcement learning model. In this model the agent at each time step perceives the state s_t of the environment and performs an action a_t. The environment responds by giving the agent reward r_t and reaches the succeeding state s_{t+1}. The agent uses this information to learn a policy $\pi : S \times A \to [0,1]$. A policy π defines the learning agent's way of behaving at a given time. In particular, the learned

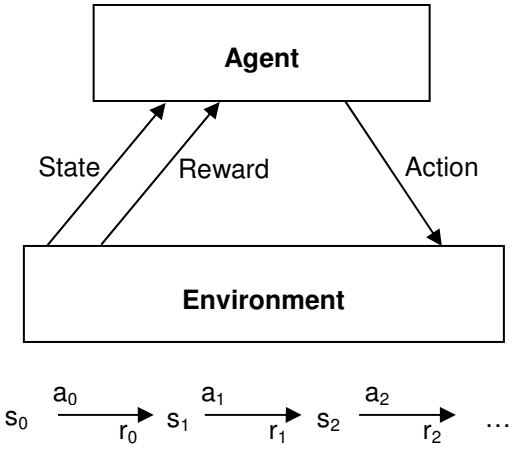


Figure 3.1 Reinforcement Learning Model [Mitchell 1997].

policy π is a mapping from the states of the environment to the probability of taking given actions when in those states.

In this dissertation we formulate all learning problems as Markov Decision Problems (MDPs). We use this formulation because in an MDP the state transitions and the reward function are only dependent on the current state and action, and not on any earlier states or actions.

3.1.1 Markov Decision Process

A Finite Markov Decision Process [Bellman 1957a] is represented as a tuple $< S, A, \psi, T, R >$ where $S = S_a \cup S_{na}$ is a finite set of states, $S_a \subseteq S$ and $S_{na} \subseteq S$ are the sets of absorbing and non absorbing states, respectively, with $S_a \cap S_{na} = \phi$, A is a finite set of actions, $\psi \subseteq S \times A$ is the set of admissible state-action pairs, indicating which actions can be chosen in a given state, $T : \psi \times S \to [0,1]$ is the transition probability function, and R is the expected reward function.

Figure 3.1 shows a problem that is formulated as an MDP. The discounted cumulative value function $V^{\pi}(s_t)$ of any arbitrary policy π learned by an RL agent for a deterministic MDP problem from an arbitrary initial state s_t is:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where r_t is the sequence of rewards generated by starting in state s_t and taking actions according to policy π, and $0 \le \gamma \le 1$ is a constant that determines the relative value of the delayed rewards versus the immediate rewards. The value $V^{\pi}(s)$ is known as the discounted cumulative reward achieved by a policy π from initial state s. The objective of the RL algorithm is to learn a policy π that maximizes $V^{\pi}(s)$ for all states s. This policy is called the optimal policy and is denoted by π^*:

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

and the value function of such an optimal policy is denoted by $V^*(s)$:

$$V^*(s) = \max_a \left(r + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right), \forall s \in S$$

where r is the expected reward that the agent receives when taking action a from state s, and $T(s, a, s')$ is the probability of transitioning from state s to s' when taking action a from state s. π^* is an optimal deterministic policy in case of an deterministic MDP. Using the optimal value function, the optimal policy function can be redefined as:

$$\pi^*(s) = \arg\max_a \left(r + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right), \forall s \in S$$

3.1.2 Q-Learning

Q learning [Watkins 1989] [Watkins and Dayan 1992] [Kaelbling et al. 1996] [Mitchell 1997] is a reinforcement learning algorithm that is used to learn optimal policies for tasks when the agent has no information about the model of the environment. As the agent has no information about the model of the environment, it does not have any information regarding r and $T(s, a, s')$. As a result it can not use the above equations to obtain an optimal policy or value function. Instead we need a more general function that enables the agent to predict the immediate reward and immediate successor state for each state-action transition. Thus a different evaluation function known as the Q function is used to learn an optimal policy. Q learning learns an action value representation $Q(s, a)$ instead of learning a state value function. $Q(s, a)$ is the expected discounted sum of future rewards when executing action a from state s and following an optimal policy thereafter.

$$Q(s, a) \equiv r + \gamma V^*(s')$$

where s' is the state reached by taking by action a from state s.

The relationship between Q value and V* is

$$V^*(s) = \max_{a'} Q(s, a')$$

so we can rewrite the Q function as

$$Q(s, a) \equiv r + \gamma \max_{a'} Q(s', a')$$

Since the definition of the Q function is recursive, it can be learned iteratively using the Q-learning algorithm. An agent using Q learning starts out with an initial Q value for each state-action pair stored. The Q values of these pairs are often initialized to small random values. The agent starts out by exploring the environment, observing the state of the environment and taking some random admissible action from that state. As a result the agent receives reinforcement

(positive or negative) from the environment. The agent uses this information to update the Q value of the corresponding state action pair using the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

where α is the learning rate. The Q values are updated until the Q values for the state-action pairs no longer change. Table 3.1 shows the Q- learning algorithm.

Table 3.1 Q- Learning Algorithm [Mitchell 1997]

For each state-action pair initialize $Q(s,a)$ to zero
Observe the current state s
Do forever

- Select an action a and execute it.
- Receive an immediate reward r
- Observe the new state s'
- Update the value of $Q(s,a)$ using the update rule:

  $$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$
- $s \leftarrow s'$

The Q-learning algorithm is guaranteed to converge if the system:

1. is a deterministic MDP.
2. the immediate reward values are bounded by some constant.
3. the agent visits every possible state-action pair infinitely often.

In the nondeterministic case where the reward function r and the transition function T have probabilistic outcomes, the above-mentioned Q-learning algorithm can fail to converge. Thus the traditional Q-learning algorithm is further extended to handle non-deterministic MDPs. In order to achieve this, the value V^{π} of a policy π is redefined as the expected discounted cumulative value of the rewards received by applying policy π.

$$V^{\pi}(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

Further we redefine π^* as the policy that maximizes $V^{\pi}(s)$ for all states s. As a result the Q value function is also rewritten as:

$$Q(s,a) = E\left[r + \gamma \sum_{s'} V^*(s')\right]$$

$$Q(s,a) = E[r] + + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

Now in order to use this Q value function iteratively we need to modify our Q value update rule as the old one can fail to converge in the non-deterministic case. This problem is overcome by modifying the training rule such that it takes a decaying weighted average of the current Q value and the revised estimate. The new training rule is:

$$Q_n(s,a) \leftarrow (1-\alpha_n) Q_{n-1}(s,a) + \alpha_n [r(s,a) + \gamma Q_{n-1}(s',a')]$$

where $\alpha_n = \frac{1}{1 + visits_n(s,a)}$, s and a are the state and action of the n[th] iteration, and $visits_n(s,a)$ is the total number of times this state-action pair has been visited up to and including the n[th] iteration.

3.1.3 Exploration vs. Exploitation Strategies

A reinforcement learning algorithm is different from other supervised learning algorithms in the sense that the learner has to explicitly explore the environment while learning a policy. One of the strategies for an agent in state s is to select an action a that maximizes $Q(s, a)$; this strategy is known as *exploitation*. But by using this strategy the agent risks overcommitting to actions that are found to have high Q-values during the early phase of training, failing to explore the other possible actions that might have the potential to yield even higher Q-values. The Q-learning convergence result requires that each state–action transition occurs infinitely often and since only the best action is chosen, this approach runs the risk of not

achieving the convergence of the learning algorithm for actions that are avoided. Another strategy for an agent in state s is to select a random action a. This strategy is known as *random exploration.* In this strategy, the agent does learn actions with good values but this turns out to be not very significant since the agent is following the approach of not putting to use what it has learned during exploration.

Therefore, the best way to train a reinforcement learner is a strategy that does both exploration and exploitation in moderation. This means a method that allows the agent to explore when it has no idea of the environment, and to exploit greedily when it has learned sufficiently about the environment. The method that this thesis uses for the aforementioned purpose is referred to as the *Boltzmann "soft-max" distribution*.

In a *Boltzmann "soft-max" distribution*, if there are *n* items and the "*fitness*" of each item i is $f(i)$, then the Boltzmann distribution defines the probability of selecting an item i, $p(i)$, as

$$p(i) = \frac{e^{\frac{f(i)}{\rho}}}{\sum_j e^{\frac{f(i)}{p}}}$$

where ρ is called temperature. By varying the parameter ρ we can vary the selection from picking a random item (ρ *is infinite*) to having higher probabilities for items with higher fitness (ρ *small finite*), and to strictly picking the item with best fitness (ρ *tends to 0*). This is accomplished by decaying the temperature exponentially using the equation $\rho_t = \rho_0 * e^{-\lambda t}$ where ρ_t is the temperature at time step t, ρ_0 is the temperature at time step $t = 0$, λ is the decay constant and t is the time step.

In our case where there are n actions from state s, the fitness of an action is given by $Q(s, a_i)$. The probability $p(a_i \mid s)$ of taking action a_i from s is given by:

$$p(a_i|s) = \frac{e^{\frac{Q(s,a_i)}{\rho}}}{\sum_{j=0}^{n} e^{\frac{Q(s,a_j)}{\rho}}}$$

There are many other types of exploration/exploitation techniques [Thrun 1992] used to solve Q-learning problems for example semi uniform distribution, error based exploration, selective attention etc. Boltzmann exploration here is used because of its ability to focus exploration on ambiguous situations and actions and because of its frequent use across RL research.

3.2 Temporally Abstract Reinforcement Learning

Though reinforcement learning allows AI agents to learn an optimal policy to complete a task based on delayed rewards received by the agent when it reaches the goal state of the task, it does not scale well to larger complex tasks. For example, in Q-learning where the Q values of state-action pairs are stored in a table, as the number of state variables increase the size of the state space increases exponentially, thus also increasing the number of entries of the table, making the learning of tasks increasingly time consuming and at some point totally infeasible as the complexity of tasks increases. Further, an agent has to make decisions about what action it needs to perform at each time step in order to complete a task successfully. As the complexity of the tasks increases, making decisions about actions at each time-step and operating in real-time becomes infeasible. As a result, we need new methods that allow learning, planning and representing knowledge at multiple-levels of temporal abstraction.

Temporally abstract reinforcement learning [Dietterich 2000] [Parr 1998] [Kim and Dean 2003] [Sutton et al. 1999] allows planning and learning at different levels of abstraction.

Temporal abstraction allows an agent to abstract the decision making point to a higher level [Sutton et al. 1999] [Thrun and Schwartz 1995] [Brooks 1986] [Huber and Grupen 1997]. Temporally abstract actions are like macro operators where a sequence of operations can be invoked by a name as if it were a primitive action. Figure 3.2 shows an example task of moving the trash to a trash can using macro-operators and simple actions. In the temporal abstraction approach to reinforcement learning these macro-operators resemble closed loop partial policies. These closed loop partial policies are defined over a subset of the states and have a well defined termination condition. These temporally abstracted actions are also commonly called

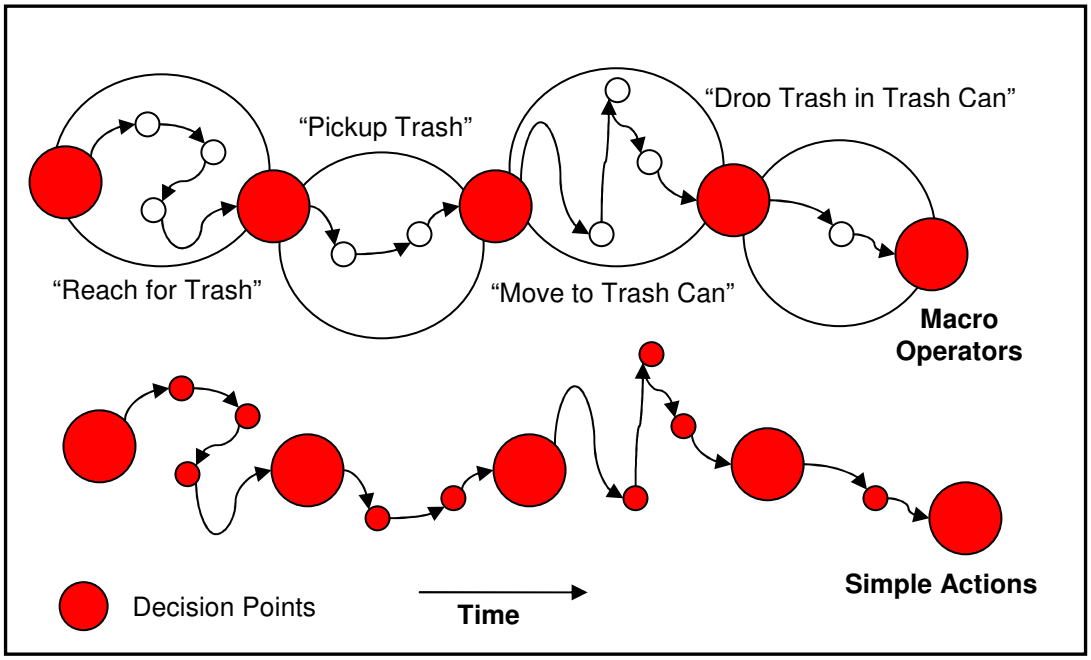


Figure 3.2 "Move Trash to Trash Can" Task using Simple Actions and Macro-Operators.

options, skills or behaviors [Sutton, Precup, and Singh 1999] [Thrun and Schwartz 1995] [Brooks 1986] [Huber and Grupen 1997].

In order for an agent to learn at multiple levels of abstraction, the formulation of a problem in the MDP framework is not sufficient. As a result the MDP framework is extended to the semi markov decision process (SMDP) framework.

3.2.1 Semi Markov Decision Process

A discrete time SMDP is a generalization of an MDP in which actions take multiple time steps to complete. An SMDP is defined as a tuple $\langle S, A, \psi, T, R \rangle$ where S is the set of states, A is the set of actions, $\psi \subseteq S \times A$ is the admissible set of state-action pairs, $T : \psi \times S \times N \to [0,1]$ is the transition probability function with $T(s, a, s', n)$ being the probability of transition from state s to s' under action a in n time steps, and $R : S \times A \times N \to \Re$ is the expected discounted reward function, with $R(s, a, n)$ being the expected reward for performing action a in state s and completing it in n time steps.

An agent using temporally abstract reinforcement learning to learn a policy for a task formulates the problem in the SMDP framework where the transition time in a sate corresponds to the duration of the selected activity. If τ is the transition time in state s upon execution of a higher level action a then a takes τ steps to complete when initiated in state s.The random distribution of the variable τ depends on the policies and termination condition of all of the lower level actions that comprise a.

3.2.2 Options

An option [Sutton et al. 1998] [Sutton et al. 1999] on a finite core MDP is a tuple $\langle I, \pi, \beta \rangle$ where $I \subseteq S$ is a set of initiation states from where this policy can be initiated, π is a stationary stochastic policy $\pi : S \times \bigcup_{s \in S} A_s \to [0,1]$ and $\beta : S \to [0,1]$ is a termination condition. The option $\langle I, \pi, \beta \rangle$ is available in state s if and only if $s \in I$. If an option is executed, then the base actions are selected according to policy π until the option terminates according to the termination condition β. For example, if an agent is in state s and is following an option o then the probability of the next action a is $\pi(s, a)$ and the environment transitions to the state s',

where the option terminates with a probability of $\beta(s')$ or else continues where the next action a' is determined based on the probability $\pi(s', a')$ and so on, until the option terminates. Upon termination of an option the agent can select another option to execute. $\{s : \beta(s) < 1\} \subseteq I$ is the set of states from where this option can be initiated. Thus an option's policy needs to be defined only over its initiation set I . Any primitive action of the core MDP can also be considered as an option but with a termination probability of 1 for the successor state of the initiation state.

3.2.3 SMDP Learning

An agent that formulates its problem in the SMDP framework has to use SMDP learning to learn an optimal policy to complete tasks. Let us consider a policy μ over options that selects option o in state s with the probability $\mu(s, o)$ and option $o's$ policy selects other options until it terminates. That is the policy of the option selects other options and so on, until each option is expanded down to primitive actions. Then the probability of a primitive action at any time step depends on the current state in the core MDP plus the policies of all the options currently involved in the hierarchical specification. Each policy μ over options determines a conventional policy over the core MDP. This conventional policy over the core MDP is called a flat policy denoted by $flat(\mu)$ [Parr 1998] [Dietterich 2000] [Precup 2000]. Flat policies corresponding to policies over options are generally not markov even if all the options are markov. The value function for the option policy defined in terms of the value functions of semi-markov flat policies is given as:

$$V^{\mu}(s) = E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{\tau-1} r_{t+\tau} + ... \mid \varepsilon(\pi, s, t)\}$$

where $\varepsilon(\pi, s, t)$ is the event of π being initiated at time t in state s. Given the definition of the value function for flat policies $V^{flat(\mu)}(s)$, the value of s for a policy μ over options is defined as $V^{\mu}(s)$ and the option value function for μ is:

$$Q^{\mu}(s, o) = E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{\tau-1} r_{t+\tau} + ... \mid \varepsilon(o\mu, s, t)\}$$

where $o\mu$ is the semi-markov policy that fo llows o until it terminates after τ time-steps and then continues according to μ.

Adding any set of semi-markov options to the finite core MDP yields a well defined discrete time SMDP whose actions are options and whose rewards are the returns delivered over the course of an option's execution. Since the policy of each option is semi-markov, the distributions defining the next state, transition times, and reward depend only on the option executed and the state in which the option was initialized. Sutton et al. [Sutton, Precup, and Singh 1999] extended the conventional single-step actions model to a multi-time model of an option that generalizes the single step model consisting of $R(s,a)$ and $T(s,a,s')$, s, $s' \in S$ of a conventional action a. Then for an option o the discounted reward $R(s,o)$ for any $s \in S$ is:

$$R(s,o) = E\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{\tau-1} r_{t+\tau} \mid \varepsilon(o,s,t)\}$$

where $t+\tau$ is the random time at which the option terminates, and $\varepsilon(o,s,t)$ is the event of o being initiated at time t in state s. Similarly we can define the discounted transition probability to s' given that option o is being executed and was initiated in state s:

$$F(s' \mid s,o) = \sum_{\tau=1}^{\infty} P(s',\tau)\gamma^{\tau}$$

for all $s \in S$ where $P(s',\tau)$ is the probability that option o terminates in state s' after τ time steps when initiated in state s.Using $R(s,o)$ and $F(s' \mid s,o)$ we can write $V_O^*(s)$, the optimal value function over an option set o, based on the generalized form of the Bellman optimality equation [Bellman 1957b]:

$$V_O^*(s) = \max_{o \in O_s} \left[R(s,o) + \sum_{s'} F(s' \mid s,o) V_O^*(s') \right]$$

Further, using $V_O^*(s)$ we can define the optimal Q-value function for the state-option pair as:

$$Q_O^*(s,o) = R(s,o) + \sum_{s'} F(s' \mid s,o) \max_{o' \in O_{s'}} Q_O^*(s',o')$$

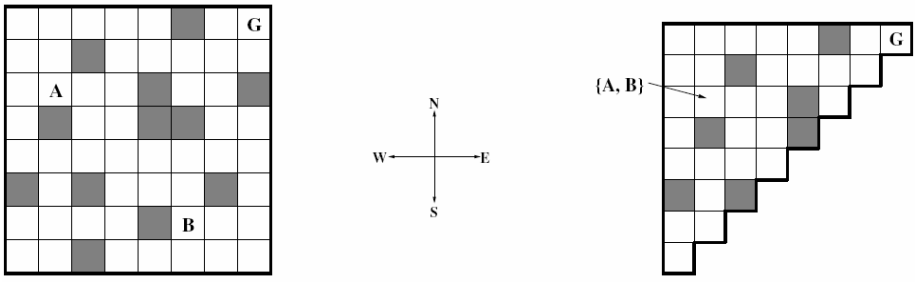
for all $s \in S$ and $o \in O_s$. This is used to derive the iterative Q-learning update:

$$Q_{k+1}(s,o) = (1-\alpha_k)Q_k(s,o) + \alpha_k\left[r + \gamma^{\tau} \max_{o' \in O_{s'}} Q_O(s',o')\right]$$

This update is applied upon termination of o at s' after executing for τ time steps, and r is the discounted reward accumulated during o's execution.

3.3 MDP Homomorphism

MDP homomorphism [Ravindran and Barto 2002][Ravindran and Barto 2003] is a formal framework developed by Ravindran and Barto to extract a smaller state space from the original one by exploiting its symmetries and redundancies. Figure 3.3 shows a grid world example where the agent has to learn to reach the goal state labeled as G. This grid world on close observation reveals that it is symmetrical about the NE-SW diagonal. As a result, taking



(a) (b)

Figure 3.3 (a) A Grid World Problem. (b) A Reduced Model of the Grid World. [Ravindran and Barto 2002]

the action "East" from the grid location labeled "A" is equivalent to taking the action "North" in grid location "B". Thus, state-action pair ("A", "East") is equivalent to state-action pair ("B","North"). This notion of equivalence can be exploited to build a smaller MDP model of this grid world which in turn can be used to learn a policy to reach the goal G. To exploit the redundancies and symmetries of a given MDP and to build a smaller MDP, Ravindran and Barto

[Ravindran and Barto 2002] [Ravindran and Barto 2003] introduce a new framework of MDP homomorphism that allows MDP minimization based on the notion of homomorphism.

Definition [Ravindran and Barto 2002]**:** An MDP homomorphism f from an MDP $\mathrm{M}=\langle \mathrm{S},\mathrm{A},\psi,\mathrm{T},\mathrm{R}\rangle$ to an MDP $\mathrm{M}'=\langle \mathrm{S}',\mathrm{A}',\psi',\mathrm{T}',\mathrm{R}'\rangle$ is a surjection from ψ to ψ', defined by a tuple of surjections $\langle h: S\to S', g_s: A_s \to A'_{h(s)} \mid s\in S\rangle$ such that:

1. for each state pair (s_i, s_j) : $T'(h(s_i), g_s(a), h(s_j)) = T(s_i, a, [s_j]_{B_f \mid S}), \forall s_i, s_j \in S, a \in A_s$
2. for each state-action pair: $R'(h(s), g_s(a)) = R(s,a), \forall s \in S, a \in A_s$

M' is said to be homomorphic image of M under f. Here B_f is partition of the block B caused by function f to which *s* belongs, the projection of B_f onto S is the partition $B_f \mid S$ such that for any $s_i, s_j \in S, [s_i]_{B_f \mid S} = [s_j]_{B_f \mid S}$ if and only if every block of B_f containing a pair in which $s_i (s_j)$ is a component also contains a pair in which $s_j (s_i)$ is a component. Condition (1) states that the state-action pairs that have the same image under f have the same block transition behavior in M, i.e. they have the same probability of transitioning to any given block of states with the same image under h. Condition 2 states that state-action pairs that have the same image under f have the same expected reward.

Definition [Ravindran and Barto 2002]**:** State action pairs (s_1, a_1) and $(s_2, a_2) \in \psi$ are equivalent if there exists a homomorphism f of M such that $f(s_1, a_1) = f(s_2, a_2)$. States s_1 and $s_2 \in S$ are equivalent if:

1. for every action $a_1 \in A_{s_1}$, there is an action $a_2 \in A_{s_2}$ such that (s_1, a_1) and (s_2, a_2) are equivalent.

2. for every action $a_2 \in A_{s_2}$,there is an action $a_1 \in A_{s_1}$, such that (s_1, a_1) and (s_2, a_2) are equivalent.

Thus, the surjection h maps equivalent states of M onto the same image state in M', while g_s is a state dependent mapping of actions in M onto image actions in M'. For example Figure 3.3 shows a homomorphism f from the grid world of Figure 3.3(a) to that of Figure 3.3(b). $h(A) = h(B)$ is the state marked as $\{A, B\}$ in Figure 3.3 (b). Also, $g_A(E) = g_B(N) = E$, $g_A(W) = g_B(S) = W$, and so on. A policy in M' induces a policy in M .

Though, MDP homomorphism allows for extracting a minimal image of a given MDP, in most cases both conditions of MDP homomorphism do not hold for the entire space ψ of an MDP, but is only satisfied by a part of the MDP. As a result, the entire MDP M cannot be abstracted to a smaller homomorphic MDP M'. This need leads to the extension, of the MDP homomorphism definition so that it permits the creation of a partial homomorphic image of a MDP. For example in the grid world environment shown in Figure 3.4(a) the agent's goal is to collect all the objects in various rooms by reaching the same location as that of the object. In this grid world scenario the entire grid world is irreducible. But all of the rooms in the worlds are equivalent to one another and transformations such as reflections and rotations map them onto each other. As a result, a partial homomorphic image of this environment can be created as shown in Figure 3.4(b) with homomorphic conditions holding only for the states in the rooms and not in the corridors.

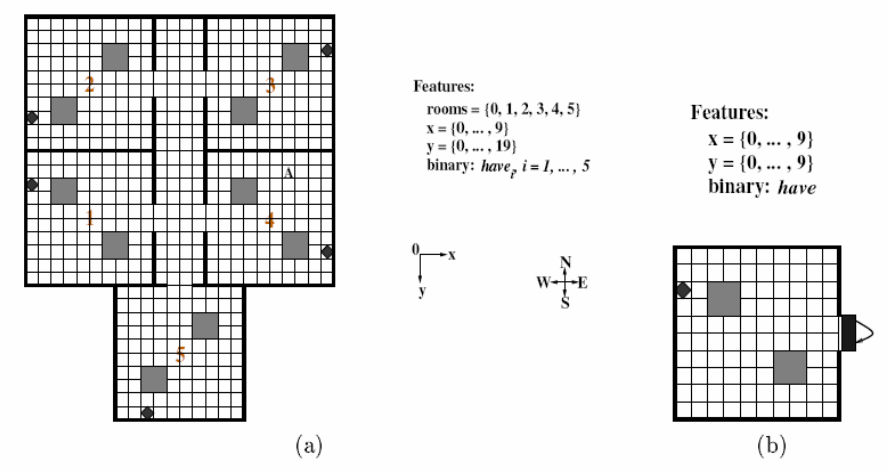


Figure 3.4 (a) A Simple Grid World Environment with Multiple Similar Rooms. (b) The Option MDP Corresponding to a Get-Object-and-Leave-Room Option. [Ravindran and Barto 2002]

Definition [Ravindran and Barto 2002]**:** A partial MDP homomorphism from $\mathrm{M}=\langle \mathrm{S},\mathrm{A},\psi,\mathrm{T},\mathrm{R}\rangle$ to $\mathrm{M}'=\langle \mathrm{S}'\cup\{\tau\},\mathrm{A}'\cup\{\alpha\},\psi'\cup\{(\tau,\alpha)\},\mathrm{T}',\mathrm{R}'\rangle$ is a surjection from ψ to $\psi'\cup\{(\tau,\alpha)\}$, defined by a tuple of surjections $f=\left\langle h:S\to S'\cup\{\tau\}, g_s:A_s\to A'_{h(s)} \mid s\in S\right\rangle$ such that:

1. state pair (s_i,s_j) : $T'(h(s_i),g_s(a),h(s_j))=T(s_i,a,[s_j]_{B_f\mid S}),\forall s_i\in h^{-1}(S'),s_j\in S,a\in A_s$
2. for each state-action pair: $R'(h(s),g_s(a))=R(s,a),\forall s\in h^{-1}(S'),a\in A_s$
3. $T'(\tau,a,\tau)=1.0$

where M' is the partial homomorphic image of M under f. The state τ is an absorbing state in M' with one action α that transitions to τ with probability 1. The homomorphism conditions hold only for states that do not map to τ. All the actions in states that map to τ, map to α. Lifting policies defined in M' yield policy fragments in M, with action probabilities specified only for the elements in the support of f.

This MDP minimization algorithm is extended to find partial homomorphic images by suitably restricting the search for homomorphisms to a subset of ψ. Further, an options framework can be used to define an options policy as a solution to the option MDP. The option MDP is defined as $M_O = \langle S' \cup \{\tau\}, A' \cup \{\alpha\}, \psi', T', R_O \rangle$ where $S' \subseteq S$, are the states in which the option policy needs to be defined, τ is an absorbing state representing the states in $S - S'$, $A' = A$, $\psi' = \{(s,a) \mid (s,a) \in \psi, s \in S'\} \cup \{(\tau, \alpha)\}$, $T'(s,a,s') = T(s,a,s')$, if $(s,a) \in \psi', s' \in S'$, $T'(\tau, \alpha, \tau) = 1$, and $T'(s,\alpha,\tau) = \sum_{s' \notin S'} T(s,a,s')$ for all (s,a) in ψ' and R_O is a reward function chosen depending on the sub task O. In the grid world shown in Figure 3.4(a) an option that accomplishes the task of collecting an object and leaving room 1 can be defined as a solution to the MDP in Figure 3.4(b). If a policy that is defined in the option MDP is lifted, it yields different policy fragments depending on the room in which the option is invoked. For example, a policy in the option MDP that picks E in all states would yield a policy fragment that picks W in rooms 3 and 4 , picks N in room 5 and picks E in room 1 and 2. In this example, various rooms in the grid world of Figure 3.4(a) exactly map onto the option MDP given in Figure 3.4(b). However, in real world scenarios exact equivalences as that of Figure 3.4 (a) and (b) are very rare, thus to increase the usefulness of such sub goal options, partial homomorphisms are extended to incorporate inexact settings. The loss of asymptotic performance of an agent in this new scenario is bounded by modeling the option homomorphism as a map from an MDP to a Bounded Parameter MDP (BMDP).

3.3.1 Bounded Parameter MDP (BMDP)

The BMDP method was introduced by [Dean et al 1997] as a mechanism to derive state space partitions of an MDP that ensure approximately optimal policies to be learned. These partitions depend on the action space and the particular reward function of the task. [Kim and Dean 2003] introduced an algorithm to derive a set of such partitions and used it to learn a policy for the task indicated by the reward function. The resulting policy is ensured to be within

an independent quality bound. The reduction technique is based on the framework of Bounded Parameter MDP (BMDP) [Kim and Dean 2003]. A BMDP is a four tuple $M = < \hat{S}, \hat{A}, \hat{\psi}, \hat{T}, \hat{R} >$ where $\hat{S}$, $\hat{A}$ and $\hat{\psi}$ are defined as for MDPs, and $\hat{T}$ and $\hat{R}$ are analogous to T and R in MDPs but assign closed intervals rather than single values to each state-action pair. That is, for any action a and states $s, s' \in S$, the values of $\hat{R}(s,a)$ and $\hat{T}(s,a,s')$ are both closed intervals $[l,u]$ where l,u are both real numbers with $l \leq u$ and in the case of $\hat{T}$ we require $0 \leq l \leq u \leq 1$. To ensure that $\hat{T}$ is well-defined we require that for any action a and state s, the sum of the lower bounds of $\hat{T}(s,a,s')$ over all states s' must be less than or equal to 1 while the upper bounds must sum to a value greater than or equal to 1. Figure 3.5 illustrates the state-transition diagram for a simple BMDP with three states and one action.

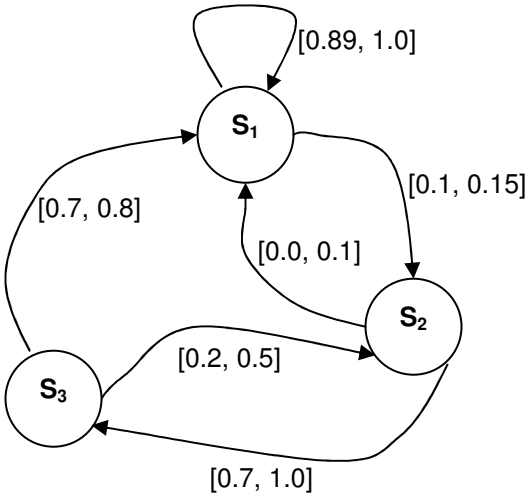


Figure 3.5 State Transition Diagram for a BMDP.

An interval value function $\hat{V}$ is a map from states to closed intervals. A BMDP $\hat{M} = \langle \hat{S}, \hat{A}, \hat{\psi}, \hat{T}, \hat{R} \rangle$ induces an exact MDP $M = \langle S, A, \psi, T, R \rangle$ where $S = \hat{S}$, $A = \hat{A}$, and $\psi = \psi'$ and

for any action a and states $s, s' \in S$, $R(s,a)$ and $T(s,a,s')$ are in the range of $\hat{R}(s,a)$ and $\hat{T}(s,a,s')$ respectively. In a BMDP $\hat{M}$, the interval value $\hat{V}$ for state s is defined by the interval:

$$\hat{V}_\pi(s) = \left[\min_{\hat{T},\hat{R}} V_\pi(s), \max_{\hat{T},\hat{R}} V_\pi(s) \right]$$

3.3.2 ε-Reduction Method

[Dean et al 1997] introduced a family of algorithms that take a MDP and a real value $0 \le \varepsilon \le 1$ as an input and compute a Bounded Parameter MDP where each closed interval has a scope less than ε. The states in this MDP correspond to blocks of a partition of the state space in which the states in the same block have approximately the same proprieties in terms of transitions and rewards. Let $P = \{B_1, \cdots, B_n\}$ be a partition of the state space [Dean et al 1997].

Definition [Dean et al. 1997]**:** A partition $P = \{B_1, \cdots, B_n\}$ of the state space of a MDP M has the property of ε-approximate stochastic bisimulation homogeneity with respect to M for $0 \le \varepsilon \le 1$ if and only if for each $B_i, B_j \in P$, for each $a \in A$ and for each $s, s' \in B_i$:

$$\left| R(s,a) - R(s',a) \right| \le \varepsilon$$

and

$$\left| \sum_{s'' \in B_j} T(s,a,s'') - \sum_{s'' \in B_j} T(s',a,s'') \right| \le \varepsilon$$

Definition [Dean et al. 1997]**:** A partition P' is a refinement of a partition P if and only if each block of P' is a subset of some block of P. In this case we say that P is coarser than P'.

Definition [Dean et al. 1997]**:** The immediate reward partition is the partition in which two states $s, s' \in S$, are in the same block if they have the same rewards.

Definition [Dean et al. 1997]**:** The block B_i of a partition P is ε-stable with respect to block B_j if and only if for all actions $a \in A$ and all states $s, s' \in B_i$:

$$\left| \sum_{s'' \in B_j} T(s, a, s'') - \sum_{s'' \in B_j} T(s', a, s'') \right| \leq \varepsilon$$

The ε-model reduction algorithm first uses the immediate reward partition as an initial partition and checks the ε-stability for each block of this partition until there are no unstable blocks left. For example, when block B_i happens to be unstable with respect to block B_j, block B_i will be replaced by a set of sub-blocks $B_{i_1}, \cdots, B_{i_k}$ such that each B_{i_m} is a maximal sub-block of B_i that is ε-stable with respect to B_j.

Theorem [Dean et al. 1997]**:** For ε > 0, the partition P found by the ε-reduction model algorithm from the MDP M is coarser than, and thus no larger than M. Once the ε-stable blocks of the partition have been constructed, the transition and reward function between blocks can be defined. The transition of each block by definition is the interval with the bounds of maximum and minimum probabilities of all possible transitions from all states of a block to the states of another block.

$$\hat{T}(B_j, a, B_i) = \left[\min_{s \in B_j} \sum_{s' \in B_i} T(s, a, s'), \max_{s \in B_j} \sum_{s' \in B_i} T(s, a, s') \right]$$

and

$$\hat{R}(B_j, a) = \left[\min_{s \in B_j} R(s, a), \max_{s \in B_j} R(s, a) \right]$$

Using the BMDP definition an approximate homomorphism can be more formally defined.

Definition [Ravindran and Barto 2002]**:** An approximate MDP homomorphism from $\mathrm{M} = \langle \mathrm{S}, \mathrm{A}, \psi, \mathrm{T}, \mathrm{R} \rangle$ to a BMDP $\mathrm{M}' = \langle \mathrm{S}', \mathrm{A}', \psi', \mathrm{T}'_{\updownarrow}, \mathrm{R}'_{\updownarrow} \rangle$ is a surjection from ψ to ψ', defined by a tuple of surjections $f = \langle h : S \to S', g_s : A_s \to A'_{h(s)} \mid s \in S \rangle$ such that $\forall s, s' \in s$ and $a \in A_s$:

1. $R'_{\updownarrow}(h(s), g_s(a)) = \left[\min_{t \in [s]_{B_h \mid S}} R(t,a), \max_{t \in [s]_{B_h \mid S}} R(t,a) \right]$

2. $T'_{\updownarrow}(h(s_i), g_s(a), h(s_j)) = \left[\min_{t \in [s]_{B_h \mid S}} T(t,a,[s']_{B_h \mid S}), \max_{t \in [s]_{B_h \mid S}} T(t,a,[s']_{B_h \mid S}) \right]$.

The MDP homomorphism framework provides a method to identify situations in which policies can be transferred. However, the underlying condition requires that the abstracted MDP and the core MDP have the same properties and identical transition structure under all possible policies, limiting its applicability to perfectly symmetric or identical environments. In addition to complete symmetry, most minimization algorithms for MDPs and other formalisms require that we specify the complete system model. For example, algorithms that exploit symmetries [Emerson and Sistla 1996] require the symmetry group be specified beforehand. Further, they require the designer to provide considerable domain knowledge to the agent which might not be available or difficult to obtain in many cases.

An RL agent that continuously learns to perform increasingly tasks in complex and dynamic environment has to have methods that abstract knowledge from its past experiences and reuse this knowledge to learn related tasks in novel environment. This dissertation introduces a new policy homomorphism, that less restrictive framework compared to that of an MDP homomorphism framework. A policy homomorphism framework requires only that policies are homomorphic instead of the whole MDPs. This enables the agent with policy homomorphism to construct a general policy using the already learning situation-specific policies of a *task type*. Further, the requirement of designer to provide considerable domain

knowledge to the agent is eliminated by learning the general policy in the form of functions. Each general policy is made up of a set of function that identifies the situations in which a given policy is applicable and a set of functions that provides information about what actions requires to be taken from each of these situations to complete the *task type* successfully. These functions allow the agent to automatically extract information (in form implicit and explicit state variables) that is important for the *task type* completion without the designer having to hand design them. The extracted general policy then can be further used to learn related tasks in novel environments by reusing them as higher level actions.

CHAPTER 4

POLICY HOMOMORPHISM FRAMEWORK

This chapter introduces the main technical contribution of this dissertation. In this chapter we introduce and formalize the policy homomorphism framework. An agent that has to continuously learn and perform in a complex and dynamic environment has to have the ability to extract knowledge from past instances of learning and performing tasks and reuse them to perform similar tasks and also learn related tasks in novel environments. The policy homomorphism framework provides the agent with this capability by enabling the abstraction of control and representational information from a set of situation-specific policies for similar task instances. The abstracted information or knowledge is saved in the form of a general policy for that *task type*. A *task type* is defined by a maximal set of policy instances for which a general homomorphic policy can be found. Further, to reuse the knowledge captured in a general policy to learn related tasks in novel environments, the agent needs the ability to predict the states in which a given general policy would be applicable. This is achieved by learning the general policy in the form of two sets of functions. The first set of functions maps the individual states of each policy to unique states of the abstract policy and the second set of functions maps individual actions from each base policy to specific actions of the abstract policy. Once the agent has learned these general policies they are stored as new skills and are further used by the agent as higher level actions in situations where they are applicable.

To facilitate learning of policies, to abstract knowledge from these policies in the form of general policies and further to reuse them to learn policies for new related tasks, we formalize all

our problems as a SMDP. To ensure that general policies for transfer represent the core objectives of the originally learned task instances the policy homomorphism framework restricts the types of policies used for abstraction to goal based policies. This is imposed to attain the goal of knowledge abstraction which is to allow reuse of knowledge gained to learn new related tasks which necessitates the need to capture the objective of policies. One way to achieve this by defining the objectives of the policies to be the goal states of these policies. While this need for goal based objectives might appear as a limitation, it should be noted here that most tasks, when represented in an appropriate state space, can be represented using goal based policies.

4.1 Goal Based Policy

A goal based policy is defined as a policy that terminates once it reaches any of the goal states of the policy. Figure 4.1 shows an example of a goal based policy. More formally a goal based policy can be defined as:

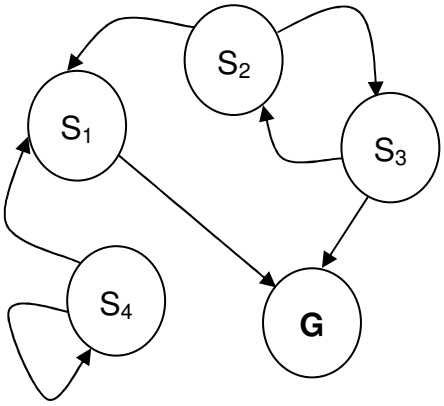


Figure 4.1 Goal Based Policy.

Definition 1: A Goal Based Policy is a tuple $\langle \pi, S_I, S_T, S_g \rangle$ where $\pi : S_\pi \times A \rightarrow [0,1]$ is a mapping from states in S_π to probabilities of selecting actions in A. $S_\pi = S_I \cup S_T$ is the state set on which policy π is defined and $S_I \subseteq S_{na}$, S_T ,and $S_g \subseteq S_T$ are the sets of initiation states, termination states, and goal states for policy π, respectively, with $(S_a \cap S_\pi) \subseteq S_T$. □

The agent that uses the policy homomorphism framework initially starts out by learning situation-specific policies for task instances. As the agent learns more and more of these policies they are used by the policy homomorphism framework to extract general policies with one general policy corresponding to each *task type.* The goal here is to construct general policy which, when applied in any of the task instances from which it was derived, would result in the successful performance if the task. Derivation of such a policy would then not only reduce the number of policies that have to be remembered in order to complete the set of past tasks, but also promises the potential to address new instances of the *task type*.

4.2 Policy Homomorphism

Figure 4.2 shows a grid world with two doors A and B. The task of the agent in this environment is to learn a policy to reach the door that is specified as a goal. To do this an agent

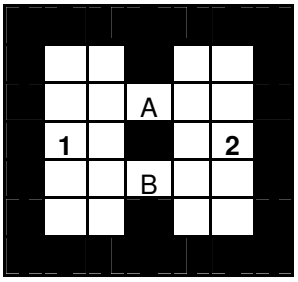
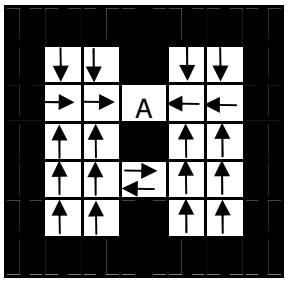


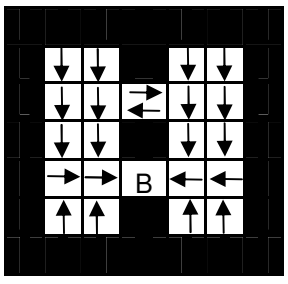
Figure 4.2 7×7 Grid World with Two Doors A and B.

in this environment has to learn a separate policy for both door A and door B. Figure 4.3(a) and (b) show the “REACH DOOR A” and “REACH DOOR B” polices learned by the agent to reach door A and to reach door B in the given environment respectively. The policy learned for “REACH DOOR A” cannot be used to reach door B and vice versa. This is because the policies learned are directly tied to the raw state features. But upon close observation we can notice that the policies learned for reaching these two doors are homomorphic to each other and can be generalized to one general policy which can solve both tasks. Figure 4.4 shows the generalized

policy extracted from the two separate policies to reach doors A and B. The generalized policy can then be used to reach both doors A and B from the states where the generalized policy is applicable. In addition, due to the generalization involved in unifying the two task instances (door A and door B), the resulting policy has the potential to represent the most relevant aspects for reaching a door and might therefore also represent a valid policy for "REACH DOOR" for other doors not used as goals in the underlying policy instances.This type of generalization is very important for a life long learning agent as it allows them to extract the knowledge gained from its past experiences of learning various tasks and use it to learn or solve related tasks in novel situations. Also, it allows an agent to reduce the number of decisions it has to make to solve a task by abstracting the decision to a higher level of abstraction. Further, generalizing extracts functional signatures for *task types* and also primitive concepts that are important to learn and perform complex tasks in a highly dynamic environment successfully.



(a)



(b)

Figure 4.3 (a) Learned Policy for "REACH DOOR A" and (b) Learned Policy for "REACH DOOR B".

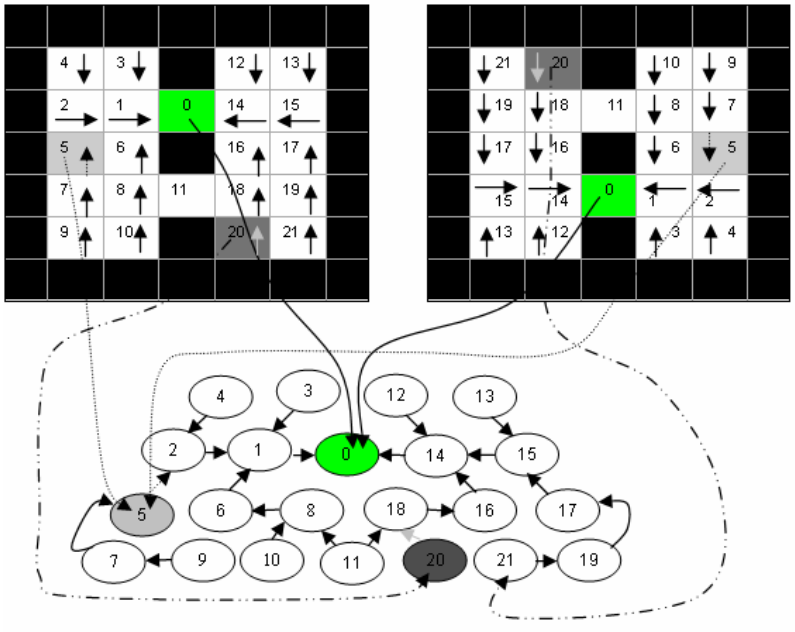


Figure 4.4 Generalized Policy for “REACH DOOR”

Figure 4.5 shows a new grid world where the agent’s task is to grab the gold. When, learning this task in this grid world (shown in Figure 4.5) the agent can use the generalized “REACH DOOR” policy to accelerate the learning of a policy to grab the gold. The grey areas in Figure 4.5 show the regions in which the learned generalized policy is applicable when applied in the context of one of the doorways as the goal of the general policy.

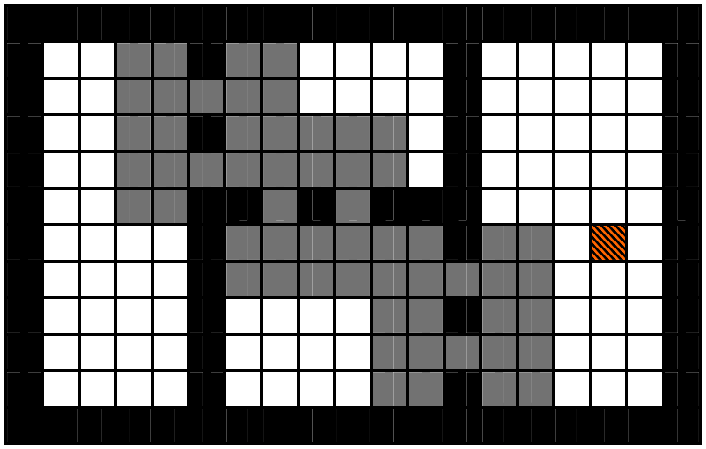


Figure 4.5 New Grid World with Areas where Generalized Policy is Applicable.

In order for a general policy to exist which performs in the same way as the underlying, previously learned policy instances when associated with the corresponding goal in the same environment, all policy instances have to be mappable onto the generalized policy, thus requiring that the policy instances be homomorphic to the generalized policy. To identify and extract a general policy that is homomorphic to a policy / set of policies, we here formally define the concept of a policy homomorphism.

Definition 2: A Policy Homomorphism $f : \pi \to \pi'$ is a surjection from a base policy π to an abstract policy π' where f is defined by a tuple of surjections $(h : S_\pi \to S'_\pi, g_s : A_\pi \to A'_\pi)$ and function $\overline{g_s} : A_\pi \times S_\pi \to [0,1]$ over the states and action pairs of policy π. Function h maps the states of the base policy π to states in abstract policy π' where $S_\pi \subseteq S$, function g_s maps the actions of base policy π to actions of abstract policy π' where $A_\pi = \{a \in A \mid \exists s \in S_\pi : \pi(s,a) > 0\}$, and the following properties hold:

1. For each state-action pair $(s,a) : \pi'(h(s), g_s(a)) = \overline{g_s}(s,a)$
2. For each state pair $(s_i, s_j) : \sum_{b \in A'_\pi} \pi(h(s_i), b) T'(h(s_i), b, h(s_j)) = \sum_{a \in A_\pi} \pi(s_i, a) T(s_i, a, s_j)$

where T and T' are the transition probabilities in the base and in the abstract policy, respectively.□

This policy homomorphism definition requires that every state s of base policy π be mapped onto the abstract state $h(s)$ in the abstract policy π'. However, allows multiple states in the base policy to be mapped onto the same state in the abstract policy. Further, it requires that the probability of transitions into a state s' from state s by taking an action determined by base policy π is equal to the probability of transitions into $h(s')$ from state $h(s)$ by taking an action in the abstract policy π'. That is, there has to be a consistent abstract policy corresponding to the

base policy where the action probabilities in the abstract policy are defined by the function $\overline{g_s}$. The latter allows to address situations where multiple action choice in the base policies lead to identical outcome.

4.2.1 Partial Policy Homomorphism

A complete policy homomorphism (as defined in definition 2) requires that every state in a given policy be mapped onto a particular state in the abstract policy. As a result, it does not allow abstraction of policies that might be partially homomorphic and thus only generalize over a subset of the state space. In order to make the policy homormorphism framework more general and applicable, we extend our definition of policy homomorphism to cover a partial policy homomorphism. Figure 4.6 shows an example of a generalized policy derived from a set of two partially homomorphic policies as well as the corresponding state and action mappings. For this example we see that the grid worlds used to learn policy A and policy B are not completely reducible and thus only a subset of the state space from each of the policies A and B can be mapped to a generalized policy.

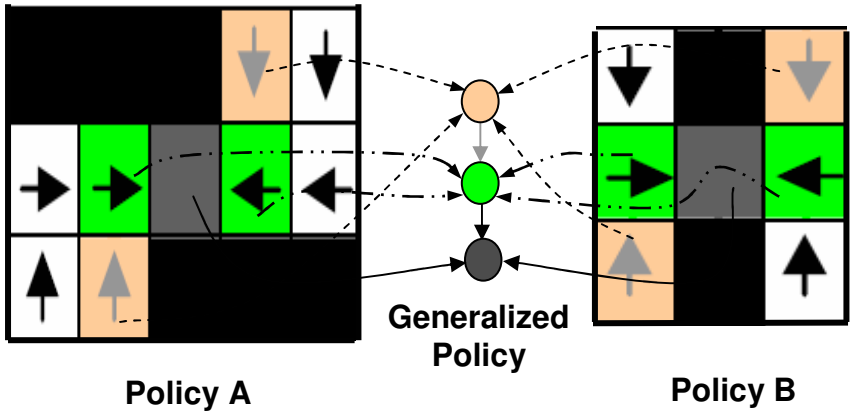


Figure 4.6 Policy Mappings from Base Policies A and B to Generalized Policy.

Definition 3: A Partial Policy Homomorphism $f : \pi_p \rightarrow \pi_p^{'}$ is a surjection from a partial base policy π_p to an abstract policy $\pi_p^{'}$ where f is defined by a tuple of surjections, $(h : S_{\pi_p} \rightarrow S_{\pi_p}^{'}, g_s : A_{\pi_p} \rightarrow A^{'}{}_{\pi_p})$ and functions $\overline{g_s} : A_{\pi_p} \times S \rightarrow [0,1]$ over the state and action sets of the partial policy π_p, $S_{\pi_p} \subseteq S_{\pi}$ and $A_{\pi_p} = \{a \in A \mid \exists s \in S_{\pi_p} : h(s) \notin S_T^{'} \wedge \pi(s,a) > 0\}$ such that the following properties hold:

1. For all state action pairs $(s,a) : \pi_p^{'}(h(s), g_s(a)) = \overline{g_s}(s,a)$
2. for each state pair (s_i, s_j) with $s_i \in S_{\pi_p}$, $s_j \in S_{\pi_p}, h(s_i) \notin S_T^{'}$:

$$\sum_{b \in A_{\pi_p}^{'}} \pi'(h(s_i), b) T'(h(s_i), b, h(s_j)) = \sum_{a \in A_{\pi_p}} \pi(s_i, a) T(s_i, a, s_j)$$

where T and T' are the transition probabilities for the base policy and the abstract partial policy, respectively. □

The partial policy homomorphism definition basically allows for the abstract policies to cover only parts of the base policies. However, it does not allow for any arbitrary sub-graphs but requires that a state in the base policy that is mapped to a state in the generalized policy either has to be mapped onto a terminal state of the generalized policy or that all of its successor states under the policy also have to mapped onto states in the generalized policy. This ensures that if an action in a given state is mapped on to the part of the partial abstract policy, then all possible outcomes of this action have to be reflected in this partial policy.

4.2.2 Approximate Policy Homomorphism

The problem with an absolute or partial policy homomorphism is that it requires that the probability of transitions into a state s' from state s by taking an action determined by base policy π to be equal to the probability of transitions into $h(s')$ from state $h(s)$ by taking an

action in the abstract policy π'. However, having two or more policies with equal transition probabilities is very unlikely in the real world. As a result an exact policy homomorphism is of limited use. In order to make the policy homomorphism framework applicable in real world tasks we introduce an approximate policy homomorphism.

Definition 4: An Approximate Partial Policy Homomorphism $f : \pi_p \to \pi'_p$ is a surjection from a partial base policy π_p to an abstract policy π'_p where f is defined by a tuple of surjections $(h : S_{\pi_p} \to S'_{\pi_p}, g_s : A_{\pi_p} \to A'_{\pi_p})$ and functions $\overline{g_s} = A_{\pi_p} \times S_{\pi_p} \to [0,1]$ over the state and action sets of the partial policy π_p, $S_{\pi_p} \subseteq S_\pi$ and $A_{\pi_p} = \{a \in A \mid \exists s \in S_{\pi_p} : h(s) \notin S'_T \wedge \pi(s,a) > 0\}$, such that the following properties hold:

1. For all state action pairs $(s,a) : \pi'_p(h(s), g_s(a)) = \overline{g_s}(s,a)$
2. for each state pair (s_i, s_j) with $s_i \in S_{\pi_p}, s_j \in S_{\pi_p}, h(s_i) \notin S'_T$:

$$\left| \sum_{b \in A'_{\pi_p}} \pi'(h(s_i), b) T'(h(s_i), b, h(s_j)) - \sum_{a \in A_{\pi_p}} \pi(s_i, a) T(s_i, a, s_j) \right| \leq \varepsilon$$

where T and T′ are the transition probabilities for the base policy and the abstract partial policy, respectively. □

The main difference between the partial and approximate policy homomorphism is that in the approximate policy homomorphism transition probabilities between the base and abstract policies can vary slightly as long as they don't differ by more than ε. Using the results of the BMDP [Givan et al. 2000] performance bounds, and assuming that the reward is concentrated at the goal (and is the same for different task instances), this ensures that the generalized policy

will perform (in terms of value) within a predetermined ε-dependent bound of the original, optimal base policy.

4.2.3 Goal Based Policy Homomorphism

Both absolute and approximate partial policy homomorphisms can be applied to derive abstract policies. However, to ensure that the general policy captures the objective of the underlying policies, and thus ensures that transferred generalized policies have a high likelihood to capture the important aspects of the underlying policy instances the approach presented here limits the application of policy homomorphisms to goal based policies. Capturing the objective is important as it characterizes the *task type.* Thus, the learning framework requires that any policy homomorphism used for transfer be also a goal based policy homomorphism.

Definition 5: A Goal Based Policy Homomorphism $f : \pi \to \pi'$ is a policy homomorphism that fulfills the following additional properties:

1. All states that are goal states in the base policy are present as goal states, S'_{π_g}, in the abstract policy π' and $S'_{\pi_g} = \bigcup_{s \in S_{\pi_g}} h(s)$.
2. $S'_{\pi_g} \bigcap (\bigcup_{s \in (S_{\pi_p} - S_{\pi_g})} h(s)) = \phi$.
3. All non-goal states in policy π' are either terminal states or have a non-zero probability to lead to a goal state under policy π'. □

A goal based homomorphism requires that all states in the generalized policy are either terminal states or can lead to a goal state under policy π'. As a result, goal-based homomorphic policies can not contain components that cannot lead to the goal, except for "boundary" states of the partial policy which are terminal and indicate failure of the policy to reach its objective.

CHAPTER 5

UTILITY OF A POLICY

A life long learning RL agent has to continuously learn to perform various tasks in a complex and dynamic environment. Although the policy homomorphism framework provides these agents with the ability to extract reusable skills and concepts from a set of situation-specific policies for similar *task types*, only having this ability is not enough for an agent to successfully learn, adapt, and perform in a complex and dynamic environment. For this, besides having the abovementioned ability, they also need to identify if a given set of policies learned needs to be abstracted and when and whether the abstracted general policy will be useful if saved. Second, they have to autonomously identify and extract reusable skills and concepts for different *task types*. This is a very important capability as the assumption that all experiences of similar *task types* occur contiguously is not very practical. Third, they need the ability to incorporate new experiences gained from learning policies for similar *task types* into an already existing generalized policy for that *task type*, i.e. the capability of adapting the existing skills and concepts associated with a given *task type* to reflect the new knowledge without having to re-build the generalized policy of the given *task type* from scratch. Finally, they have to know which of the generalized policies abstracted over time are still useful and should be included in the action set to learn new tasks. This is important because over time there might be some generalized policies that become redundant because they are replaced by newer generalized policies for a similar *task type* or by a generalized policy that encompasses the generalized policy of this *task type*. This is achieved by incorporating a new policy evaluation criterion in the current policy homomorphism framework. The criterion allows the agent to

evaluate existing and newly learned or abstracted policies or generalized policies in terms of their usefulness.

5.1 The Utility of a Policy

The criterion used here to determine whether a policy or policy generalization is useful is termed as the utility of a policy. The utility $U(\pi)$ of a policy π is defined as the expected proportion of reduction in decision complexity resulting from the use of that policy.

$$U(\pi) = \frac{E[-\Delta_{Dec}]}{E[Dec]}$$

where Dec is the number of decision points in a task and Δ_{Dec}^{π} is the difference between the number of decision points when using only previously existing action and the one when including policy π.

To capture the usefulness of a policy or a generalized policy its utility quantifies the following:

1. The precision with which a given policy or generalized policy represents situation-specific tasks policy instances.
2. The percentage of each situation-specific tasks policy a given policy or generalized policy covers.
3. The number of situation-specific tasks policies covered by a given policy or generalized policy.

Using the set of underlying tasks, the utility of a policy/generalized policy π can be written as:

$$U(\pi) = \sum_{T} p(T) \frac{E[Dec \mid T]}{E[Dec]} \frac{E[-\Delta_{Dec}^{\pi} \mid T]}{E[Dec \mid T]} - Penalty$$

where $p(T)$ is the probability of a task T, representing its importance, $E[Dec \mid T]$ is the expected length of trajectories given this task, $E[Dec]$ is the expected length of trajectories, $E[-\Delta_{Dec}^{\pi} \mid T]$ is the expected number of decision points reduced by using the policy π in task T, and $E[Dec \mid T]$ is the expected number of total decisions give the task. The penalty term in the utility of a policy captures the information about the part of the task that is not captured in the policy.

$$Penalty = \sum_{T} p(T) \frac{\left(1 - \sum_{h \in T} \frac{p(h \mid \pi)}{p(h \mid T)}\right)}{E[Dec \mid T]}$$

where $p(T)$ is the probability of a task, $E[Dec \mid T]$ is the expected number of total decisions given the task, h is a trajectory, $p(h \mid T)$ is the probability of a trajectory given the task, and $p(h \mid \pi)$ is the probability of a trajectory given the policy.

5.2 Incremental Utility and the Utility of an Action/Policy Set

Though the utility of a policy evaluates the usefulness of a policy or generalized policy, it fails to capture the utility of a set of policies when these policies share regions of the state space that are common between them. In particular, it can not correctly capture the incremental utility of adding a new policy to a set of already existing policies. This is largely because it does not capture the redundancy in the action set caused by two policies generating the same actions in overlapping parts of the state space. Moreover, it does not account for the added decision complexity (and thus the associated reduction in utility).

Thus, to evaluate the utility of a policy that shares regions of state space with other policies we define the incremental utility of a policy or more precisely the utility of an action or policy set.

For a set of policies $\pi_1, \pi_2, \cdots \pi_n$ the utility of the set of policies $U(\pi_1, \pi_2, \cdots \pi_n)$ can be written as:

$$U(\pi_1, \pi_2, \cdots \pi_n) = \sum_T \left(p(T) \frac{\sum_s p(s \mid T) \left(\sum_{i=1}^{n} \left(p(\pi_i \mid s) p(Dec \mid s, \pi_i) E[-\Delta_{Dec}^{\pi_i} \mid T] \right) - DC \right)}{E[Dec]} \right) - Penalty$$

$$DC = p(s \mid \pi_i) \sum_{k=2}^{n} \left((-1)^k \sum_{|D|=k, D=\{\pi_{l1}, \cdots \pi_{lk}\}, D \subseteq \{\pi_1, \cdots \pi_k\}} \prod_{\pi \in D} p(Dec \mid s, \pi) \right)$$

where $p(T)$ is the probability of a task T, $E[Dec \mid T]$ is the expected length of trajectories given a task, $E[Dec]$ is the expected length of trajectories, $E[-\Delta_{Dec}^{\pi} \mid T]$ is the expected number of subsequent decision points reduced by using policy π from states given the task T, and $E[Dec \mid T]$ is the expected number of total decisions given the task. The term DC represnts the added decision complexity and compensates for the additional decision choices in the region of the state space that is shared between the policies $\pi_1, \pi_2, \cdots \pi_n$, assuming that adding decision choices increases the time to learn linearly in terms of the number of extra decisions. In particular, it represents the repeated number of additional decision option caused by the availability of the policies. $P(Dec \mid s, \pi)$ represents the likelihood that policy π would be available as a choice in state s during execution of task T and $P(s \mid T)$ is the likelihood that state s would be encountered in the task T. The penalty term in this utility again in captures the information about the part of the task that is not captured in the policies.

$$Penalty = \sum_T \left(p(T) \sum_{i=1}^{n} \left(p(\pi_i \mid h) \frac{\left(1 - \sum_{h \in T} \frac{p(h \mid \pi_i)}{p(h \mid T)} \right)}{E[Dec \mid T]} \right) \right)$$

where $p(T)$ is the probability of a task, $E[Dec \mid T]$ is the expected number of total decisions given the task, h is a trajectory, $p(h \mid T)$ is the probability of a trajectory given the task, $p(h \mid \pi_i)$ is the probability of a trajectory given the policy π_i, and $p(\pi_i \mid h)$ is the probability that policy π_i would be chosen in task T.

The analytical calculation of these utility terms is very difficult. Therefore, we calculate these utilities by sampling trajectories from the situation-specific policies learned by the RL agent. The main assumption we make during these calculations is that the decision complexity increases linearly with the increase in the number of decision point and the action set. From the utility of action sets, the incremental utility of a policy π_i when added to action set D, $\pi_i \notin D$, can be calculated as $U(D \cup \{\pi_i\}) - U(D)$.

The utility of a policy or generalized policy can then be used by the policy homomorphism framework in two separate contexts:

1. Context of abstraction.
2. Context of deciding an action set.

5.3 Utility-Based Task Type Identification and Policy Generalization

The utility of a policy in the context of abstraction is calculated only with respect to the task instances that are candidate for abstraction. This allows the policy homomorphism framework to evaluate if a policy or generalized policy is worth saving. For example: if we have a generalized policy that covers only a small portion of each of the underlying situation-specific policies, then the utility of the set containing this generalized policy and the situation-specific policies from which it is derived would be negative and as a result it would not be useful to save this generalized policy as there would be no net gain hence the added decision complexity caused by adding the extra policy exceeds the number of decisions and task instances to which

it can be applied. Further, if there is a choice between different sets of generalized policies abstracted from same set of situation-specific policies, it needs to be decided which of these sets identify the best *task types* to be saved. For example, given hat from a set of situation-specific policies $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$ one can get two sets of generalized policies $S_1 = \{\pi_1'\}, S_2 = \{\pi_2', \pi_3'\}$, where set S_1 contains generalized policy that covers parts of all five situation-specific policies of one *task type*, while set S_2 contains a set of generalized policies that divide the five situation-specific policies into two separate *task types* but cover a large part of each of the underlying policies. The utility of a policy/set of policies in this scenario allows the comparison of the utility of policies in set S_1 to with the utility of the policies in set S_2, thus allowing the set with the best utility to be saved which also derives the best possible *task types* that this set of situation-specific policies belong to.

5.4 Controlling Action Set Size

The utility of a policy in the context of deciding an action is calculated with respect to the entire history of task instances that an agent has encountered during its lifetime. This allows the policy homomorphism framework of an agent to decide which of the policies saved is the best choice to be included in the action set as higher level actions during the learning of new task instances given the entire history of the task instances encountered by this agent. Further, it gives the agent the ability to adjust the exploration rates of the actions (lower level or higher level) in the action set based on the incremental utility of the actions by considering them as policies. The utility of actions gives a measure of how useful these actions are in the context of the entire history of task instances which in turn gives a measure to predict how good they are in the current scenario. As a result, exploration probabilities for the actions with higher incremental utility can be adjusted so that they are explored more often and thus, if the task instance being learned is represented by already learned policies/generalized policies, then it would learn it faster but with a caveat that if the task instance being learned has never been seen or is not

captured in the generalized policies derived for each *task type* then the learning with this exploration rate might take the agent longer to learn compared to the scenario where there exploration rates are uniform.

CHAPTER 6

LEARNING FRAMEWORK

This dissertation introduces a novel approach for autonomous skill and representational abstraction. The generalized skills with their corresponding representational abstractions form the generalized policies for instances of a common *task type* and characterize abstract state attributes that allow to identify situations in which the generalized policies are applicable. Using the concept of a policy homomorphism, this framework generalizes a set of previously learned, situation-specific policies for similar tasks into an abstract policy for the corresponding *task type*, where a *task type* is defined by the maximal set of policy instances for which a general homomorphic policy can be found. A general policy is formed here by learning a set of functions, h, that map individual states of each policy to unique states of the abstract policy and a set of functions, g_s, that map individual actions from each base policy to specific actions of the abstract policy. Once the agent has abstracted a general policy, it can use it to directly address similar tasks in novel situations or to learn new complex tasks. This is achieved by adding the generalized policy to the agent's action set, allowing the agent to choose it from states in which it is deemed applicable based on a generalization of the learned state mapping functions, h. The working of this policy generalization method and the potential of the reuse of generalized policies is demonstrated on a set of grid world domain examples in Chapter 7. The learning of basic and generalized polices along with its reuse to learn new tasks mainly happens within the learning framework of the RL agent.

Figure 6.1 shows the learning framework of a life-long learning RL agent using the policy homomorphism approach. The learning framework shown in Figure 6.1 is made up of 4 main parts:

1. Learning Component
2. Skills Memory
3. Concepts Memory
4. Policy Abstraction Component

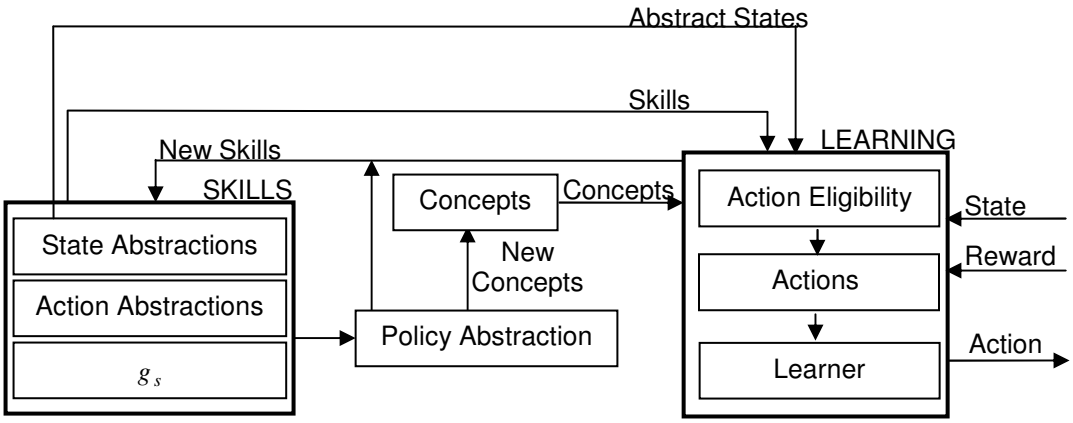


Figure 6.1 Learning Framework

6.1 Learning Component

The learning component of the RL agent's learning framework learns policies to perform tasks successfully. At each time t the agent perceives the state s_t of the environment and chooses to perform action a_t from the set of admissible actions. As a result the environment reaches state s_{t+1} and the agent receives a reward r_t. The agent uses this information to learn a policy π that maximizes the expected reward. Policies that choose actions only from the set of primitive actions are called situation-specific policies. Once the agent learns a situation-specific policy the agent stores this policy in skill memory which is in turn used by the policy abstraction component to abstract a general policy for a given *task type*. The abstracted

generalized policy is added to the skill repository and also to the set of actions. These abstracted general policies are known as the higher level actions which allow the agent to reuse the knowledge gained from previous experiences of a given *task type* to learn policies for related tasks in novel situations. Each higher level action is only admissible in specific situations, identified by the abstracted state representations of the general policy corresponding to the given higher level action. All higher level actions may take multiple time steps to complete. To learn a policy for a new task while reusing the knowledge of the past experiences available, the agent at time step t perceives the state s_t of the environment and calculates the set of admissible higher and lower level actions from the current state, and chooses an action. If the chosen action is a higher level action o_t then the agent continues to choose actions based on the policy of the general policy corresponding to the higher level action o_t until the policy terminates or the agent reaches a state s_{t+k} from where o_t is no longer available. As a result, the environment gives the agent reward r which is used by the agent to learn a policy for the new task such that it maximizes the expected reward. The RL agent in this dissertation uses Q-learning to learn policies for tasks.

6.2 Skills and Concepts Memory

The skills and concepts components of the learning framework represent repositories of learned skills and concepts, respectively, which are used by the Learning component to learn policies for new tasks while interacting with the environment.

6.3 Policy Abstraction Component

The policy abstraction component of the learning architecture performs the job of policy generalization. It uses a set of learned policies for similar tasks to autonomously abstract a general policy for a *task type*. To do this, the agent uses the concept of policy homomorphism and goal based policies for the autonomous identification of similar tasks and the construction of

the general policies. In order to ensure that these general policies can be reused in subsequent learning tasks it is essential here that they are extracted in a form that can be transferred to new situations and novel environments. While in new situations where the entire state space of the new environment is known it would be possible to re-evaluate whether the local state space is homomorphic, this is very computationally complex and would limit the applicability of the policy homomorphism framework. To overcome this, it is necessary that the general policy, and thus the mapping functions h, g_s, and $\overline{g_s}$ are derived explicitly in a form that allows their application for a new situation and new environment. To achieve this, the learning framework explicitly learns representational concepts and parametric mapping functions.

6.3.1 Parametric Mapping Functions

To facilitate the application of the generalized policy in novel situations and in the context of new tasks of the same task type without the need for complete knowledge of the state space , the policy abstraction component learns and represents the policy homomorphism as an explicit parametric mapping function, $f_G(s, G)$, from the set of underlying base policies. The surjection are here parameterized in term of the state and the target goal, allowing them ti be applied flexibly to all available for the given task type. For example, in the case of a “REACH DOOR” task in an environment with multiple doorways, the parametric functions yield different state and action mapping, $h_G(s)$ and $g_{s_g}(a)$, for each potential goal G.

In the course of learning these parametric mapping functions, the policy abstraction component also identifies representational concepts that are important in the description of the mapping functions and thus represent concepts that are fundamental in characterizing the state in the context of the general policy. These could therefore subsequently be used as state abstractions to yield more compact state representations. For example, in the context of the “REACH DOOR” task, concept like relative distance to the door and of relative orientation are

essential constructing the parametric state mapping and should therefore be discovered by the policy abstraction component for storage in concept memory.

To learn these concepts and the mapping function $f_G(s,G) = (h_G, g_{s_G})$, two parametric mappings, $h'(G,s) = h_G(s)$, and $g'_s(G,s,a) = g_{s_g}(a)$ are learned from positive and negative training instances extracted from the base policies. The resulting mapping functions, when applied to a particular goal instance (extracted from the current state information) then allows to map the current state in a new environment onto a state in the generalized policy without the need for complete knowledge of the system model.

6.3.2 Policy Abstraction

The abstraction process starts at the goal state and at each step extracts a general mapping function $f_G(s,G) = (h_G, g_{s_G})$ such that $h'(G,s) = h_G(s), g'_s(G,s,a) = g_{s_g}(a)$. The sets of functions h_G map the individual states of the specific policy instances to unique states of the abstract policy and the set of functions g_{s_G} map the actions from specific policy instances to abstract actions in the general policy, respectively. Upon the extraction of the mapping function the policy extraction algorithm calculated a set of predecessor states that lead to the state set $h_G^{-1}(s)$ in the base policies when an action $a \in A(s)$ is taken. This process continues until the policy abstraction algorithm can no longer extend the current generalized policy. The general mapping functions of a generalized policy allow states in new policy instances to be mapped to states in the general policy without the need to map the entire partial policy and thus without the need for a complete state space model for the new environment. The functions are abstracted using a decision tree classifier to which at each step we provide the positive and negative instances of for the set of states or action for which a general function is be formulated. The abstraction process itself follows a greedy method to abstract a general policy. This method is employed to make the whole process of policy abstraction tractable. Table 6.1 and 6.2 detail the

algorithm used to abstract a partial generalized policy and a approximate partial generalized policy.

Table 6.1 Partial Policy Generalization Algorithm

1. Initialization t=0,

$$h' = \{\}, g'_s = \{\}, P = \{\pi_{p_1}, \pi_{p_2}, \pi_{p_3} \cdots, \pi_{p_n}\} \quad , \quad S'^{(0)}_{\pi_p} = s'_g; \forall i : S^{(0)}_{\pi_{p_i}} = s_{g_i}, h'(s_{g_i}, s_{g_i}) = s'_g$$

2. Increment *t*

3. $\forall \pi_{p_i} \in P$ find $s_i^{(t)} \in (S_{\pi_{p_i}} / S^{(t-1)}_{\pi_{p_i}})$ such that:

   - $s_i^{(t)}$ is a predecessor of a state in $S^{(t-1)}_{\pi_{p_i}}$

     $$\exists s \in S^{(t-1)}_{\pi_i}, a : (T(s_i^t, a, s) > 0.0) \wedge (\pi_i(s_i^t, a) > 0.0)$$

   - There is an abstract state $s'^{(t)}$ such that

     - $h'(s_{g_i}, s_i^{(t)}) = h_i(s_i^{(t)}) = s'^{(t)}$
     - $g'_s(s_{g_i}, s_i^{(t)}, b) = g_{s_i}(b)$
     - (h_i, g_{s_i}) is a partial policy homomorphism for the partial policy π_{p_i} with state set

       $S^{(t-1)}_{\pi_{p_i}} \cup s_i^{(t)}$

   - $S^{(t)}_{\pi_{p_i}} = S^{(t-1)}_{\pi_{p_i}} \cup s_i^{(t)}, S'^{(t)}_{\pi_p} = S'^{(t-1)}_{\pi_p} \cup s'^{(t)}$

   If step 3 is successful, then: *Goto 2.* Else: *Stop*

Table 6.2 Approximate Policy Generalization Algorithm

1. Initialization t=0,

$$h' = \{\}, g'_s = \{\}, P = \{\pi_{p_1}, \pi_{p_2}, \pi_{p_3} \cdots, \pi_{p_n}\}, \; S'^{(0)}_{\pi_p} = s'_g; \forall i : S^{(0)}_{\pi_{p_i}} = s_{g_i}, h'(s_{g_i}, s_{g_i}) = s'_g$$

2. Increment t

3. $\forall i : \pi_{p_i} \in P$ find $s_i^{(t)} \in (S_{\pi_{p_i}} - S^{(t-1)}_{\pi_{p_i}})$ such that:

   - $s_i^{(t)}$ is a predecessor of a state in $S^{(t-1)}_{\pi_{pi}}$ $\exists s \in S^{(t-1)}_{\pi_{pi}}, a :$
     $(T(s_i^t, a, s) > 0.0) \wedge (\pi_i(s_i^t, a) > 0.0)$

     $\left| T(s_i^t, a, s) - T'(h(s_i^t), g_s(a), h(s)) \right| \leq \varepsilon$

     - There is an abstract state $s'^{(t)}$ such that
     - $h'(s_{g_i}, s_i^{(t)}) = h^{(t)}(s_i^{(t)}) = s'^{(t)}$
     - $g'_s(s_{g_i}, s_i^{(t)}, b) = g_{s_i}(b)$
     - (h_i, g_{s_i}) is an approximate partial policy homomorphism for the partial policy π_{p_i} with state set $S^{(t-1)}_{\pi_{pi}} \cup s_i^{(t)}$

   - $S^{(t)}_{\pi_{pi}} = S^{(t-1)}_{\pi_{pi}} \cup s_i^{(t)}, S'^{(t)}_{\pi_p} = S'^{(t-1)}_{\pi_p} \cup s'^{(t)}$.

4. If step 3 is successful, then: Goto 2. Else: Stop

CHAPTER 7

EXPERIMENTS AND RESULTS

This chapter demonstrates the working of the policy homomorphism framework using examples in a grid world domain. The first set of experiments is performed on a deterministic grid world domain to demonstrate the working of the policy homomorphism framework in the context of extracting a partial policy from a set of policies of similar *task type* and its reuse to show how the extracted general policy can be used to accelerate learning of related tasks in a novel scenario. The second set of experiments demonstrates the extraction of an approximate partial policy from a set of situation- specific policies of a similar *task type* using the policy homomorphism framework and its reuse to shows how the extracted approximate partial policy can be use to learn related tasks in novel environments. Each of these first two sets of experiments is further divided into 3 phases. Phase one learns a set of basic policies to perform task instances. The second phase extracts a general policy from these situation-specific policies of similar *task type*. In the third and final phase of the experiments the newly abstracted general policy is added as a new higher level action to the RL agent's action repository. This is aimed at allowing the reuse of the knowledge gained in the form of a generalized policy. The added higher level action along with the initial action set is used to learn policies for similar tasks in a novel situation and the learning performance is compared to the situation where only the initial actions are available.

In the third set of experiments we working of policy homomorphism framework to demonstrate how the learning framework can be used to autonomously abstract and categorize

different *task types* and form general policies using a set of situation-specific policies for each of these *task types.* Further, we show how the set of generalized policies abstracted can then be reused to learn related tasks in novel situations.

7.1 Experiments in Deterministic Grid World Domains

In this set of experiments we show how the agent can abstract a general policy from a set of policies of similar *task type* and reuse them to learn related tasks in novel situations. The RL agent in this set of experiments begins by learning policies for reaching various doors in a set of grid world domains. These “REACH DOOR X” policies are then used to abstract a general “REACH DOOR” policy. The general “REACH DOOR” policy is further used by the agent to learn similar tasks in novel situations.

For these set of experiments the grid world domain of the agent is deterministic and consists of multiple doors. Each location in the grid world is of one of the four types Wall, Empty, Obstacle, and Doorway. The agent in this grid world domain is initialized randomly at one of the empty locations in the grid world. The agent’s action space consists of “FORWARD”, “TURN-LEFT” and ”TURN-RIGHT” actions. Each of these actions in the agent’s action space takes only one time step to complete. The state space of the agent consists of the X,Y location of the agent, orientation of the agent and the X,Y locations of the four doors nearest to the agent. At each point in time the agent observes the state of the environment and chooses an action to perform. For each action the agent performs in this grid world domain it incurs a cost of -0.25. In addition the agent receives a reward of +100 from the environment whenever it reaches a door that is identified as a goal state.

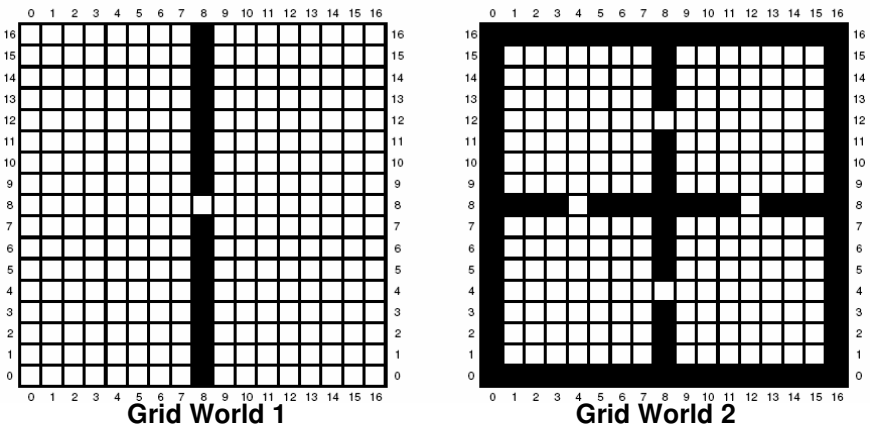


Figure 7.1 17×17 Deterministic Grid Worlds used for Learning Situation-Specific Policies for Doorways.

7.1.1 Learning Basic Policies

In this phase of the experiment the RL agent learns a set of basic policies for reaching specific doorways in a grid world domain. Figure 7.1 shows the set of grid worlds used by the agent to learn a set of situation-specific "REACH DOOR X" policies. Grid World 1 shown in Figure 7.1 has two rooms connected by a doorway and Grid World 2 has four rooms, each connected to its adjacent room by a doorway. The agent starts at a random location within the specified grid worlds and uses Q-learning to learn a policy to reach a door that is identified as the goal state. At each point in time the agent perceives the state s_t of the environment and chooses to perform an action a_t from the set of admissible primitive actions. As a result the state of the environment transitions from state s_t to s_{t+1} and the agent receives a reward r_t. The agent uses this information to learn a policy. During the learning, the agent uses the Boltzmann "soft-max" distribution to explore the grid world, reducing the temperature variable until the exploration drops to approximately 10%. This level is maintained to allow the system to learn the globally optimal policy. As a result of learning in both these grid world environments,

the agent learns 5 basic policies to reach each of the individual doorways. Figure 7.2 shows one of the trajectories followed by the agent under the policies learned to reach a door for each of the grid worlds shown in Figure 7.1.

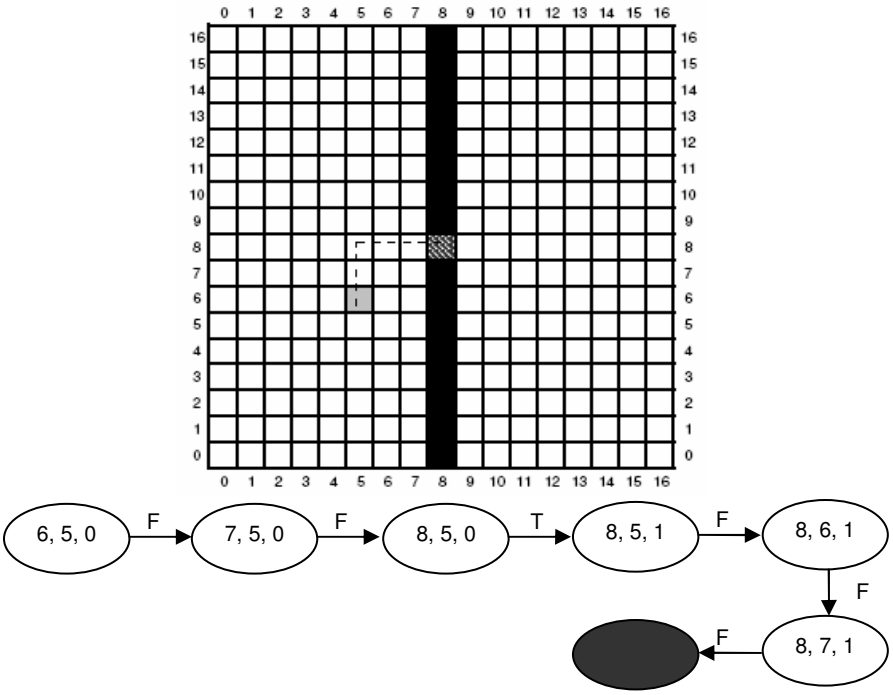
7.1.2 Extracting a Generalized Policy

The policies learned by the agent to reach doorways in the previous phase can only be used in future scenarios if the environment is exactly the same as the ones for which the policy is learned. Even a slight change in the environment makes the learned policies useless. This is because the policies learned directly tied to the raw state features which change every time the environment used to learn policy changes. In particular, each state is characterized in terms of the absolute position of the agent and take doorways and therefore any move of the doorway will prevent the agent from reaching the door using the learned policy. Thus, in order to reuse the knowledge gained from learning specific reach door policies to learn related tasks in novel situations, an RL agent needs to abstract knowledge gained in the form of reusable skills and concepts.

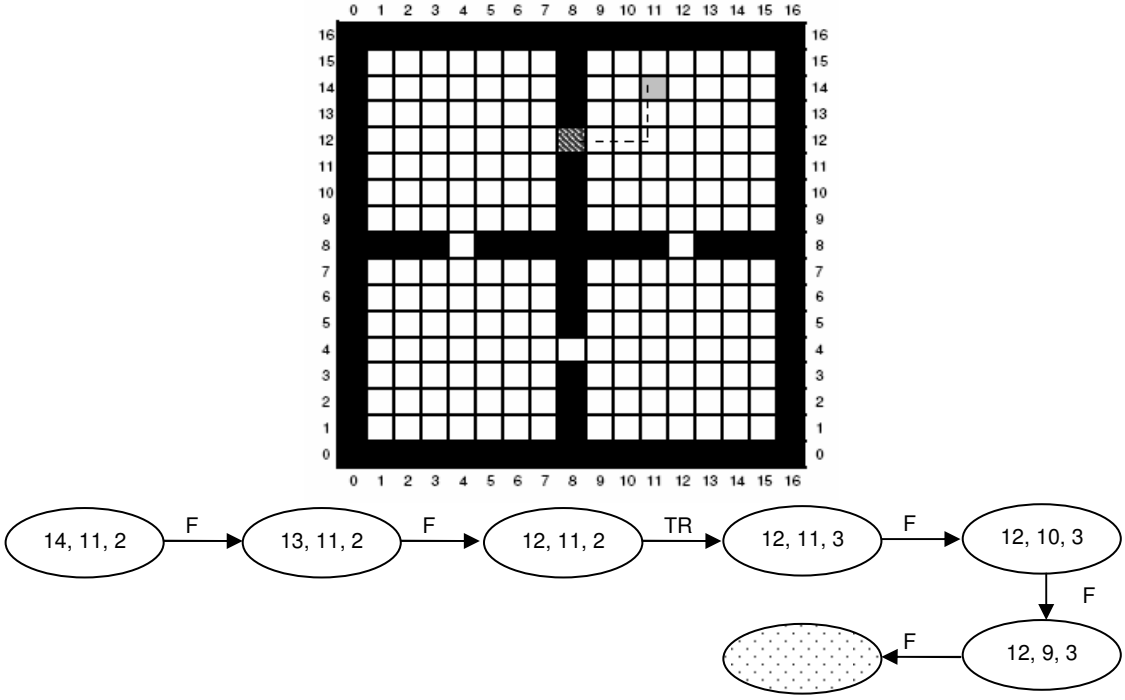
In this phase of the experiment the agent uses the situation-specific “REACH DOOR X” policies to extract a general policy using the policy abstraction algorithm. The policy abstraction algorithm uses the partial goal based policy homomorphism framework to extract a general policy.

7.1.2.1 Enhancing the Attributes of a States in a Policy

To allow the policy abstraction algorithm to capture a general state mapping function without the need for a complex function approximator, each situation-specific policy that is a candidate for the abstraction process goes through a preparation process before they are used to abstract a general policy. In this preparation phase, the states within the base policy are



(a)



(b)

Figure 7.2 (a) One of the Trajectories Followed by the Agent under the Situation-Specific Policy Learned for the Doorway in Grid World 1 and (b) One of the Trajectories Followed by the Agent under the Situation-Specific Policy Learned for One of the Doorways in Grid World 2.

enhanced. By enhancing, we mean that there an addition to the set of attributes that represent that state. The additional attributes that are added to these states are formed by applying various arithmetic, logical, relational operators on an attribute or a set of attributes of the original state representation. For example if a state is represented by the agent's X, Y locations, the agent's orientation and the goal's X, Y locations then, the additional attributes of this state can be formed by attributes like "$X_A - X_G$" , "$Y_A - Y_G$", "Is Goal on Agent's Left" is formed by the result of operation "if $X_A<X_G$" etc. We do this preprocessing because most of the time by adding these attributes which are formed through manipulation of the primary attributes may expose functional signatures that are important to capture the *task type* at an abstract level and that are not captured in the primary state attributes. As a result of preprocessing about 100 states attributes are added to the state beside original state attributes.

Once each policy has been augmented with the extra attributes the policy abstraction algorithm starts to extract a general policy from the set of base policies. Starting from the goal state, it builds a set of functions that classify the states of each policy in terms of the corresponding abstract state in the general policy by adding one state at a time. Similarly, it also builds action mapping functions to map action a_i that results in a transition from state s_t to s_{t+1} in the base policy to a_i' that result in a corresponding transition from $h(s_t)$ to $h(s_{t+1})$ in the abstract policy. The algorithm continues until it reaches a point where the addition of more states does no longer increase the expected utility of the general policy. The functions are abstracted using a decision tree classifier to which at each step we provide the positive and negative instances of the set of states and actions for which a general function is be formulated. Figure 7.3 shows a subset of the abstracted states and actions of the extracted generalized policy using the five basic "REACH DOOR" policies. Table 7.1 shows the states for one of the trajectories followed by the learned policy in Grid World1 (shown in Figure 7.2(a)) and Grid World 2 (shown in Figure 7.2(b)) with their corresponding abstract states in the general

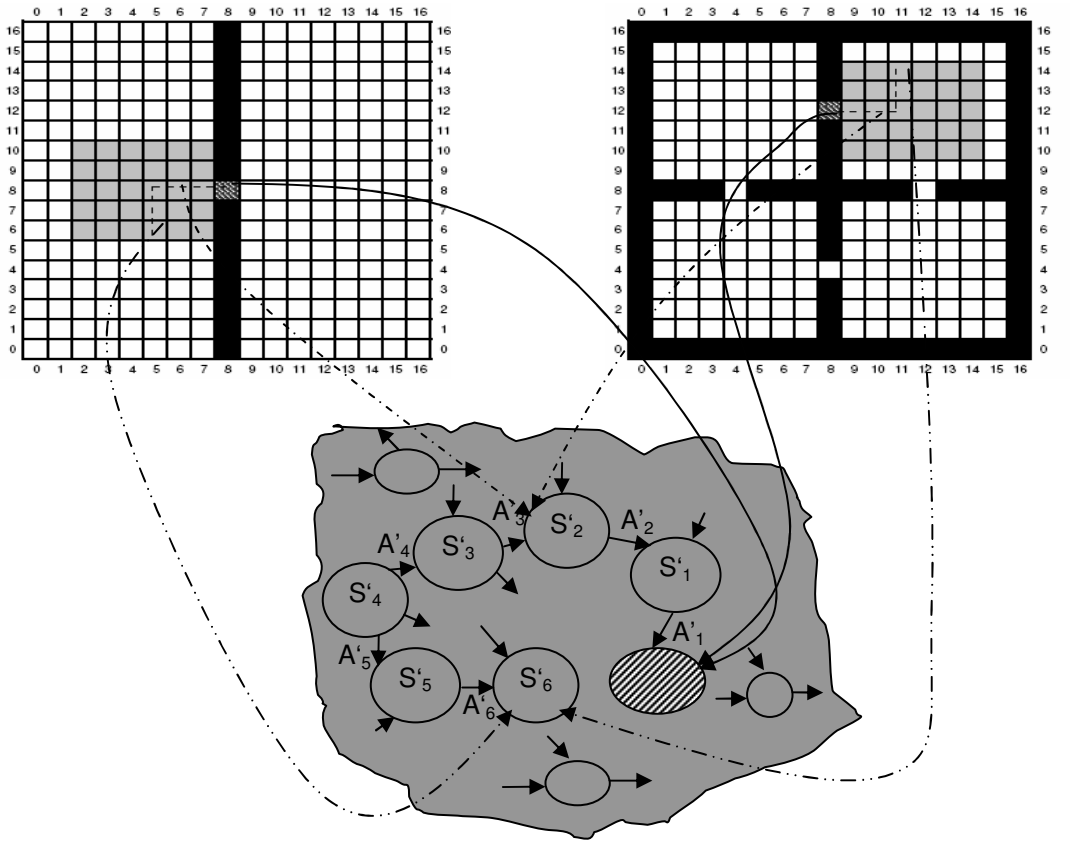


Figure 7.3 A Subset of the Abstracted States and Actions of the Extracted General Policy Using the Five Situation-Specific Policies to Reach Individual Doorways

Table 7.1 States in One of the Trajectories Followed Under the Learned Policy with its Corresponding Abstract States in General Policy and their Respective State Mapping Function.

Grid World 1	Grid World 2	Abstract State	Abstract State Function
6, 5, 0	14, 11, 2	S_1'	$DTF > 1 \wedge DTF \leq 2 \wedge ADFG > 4 \wedge ADFG \leq 5 \wedge ALHG = False$
7, 5, 0	13, 11, 2	S_2'	$DTR > 2 \wedge AFHG = True \wedge ADFG \leq 4$
8, 5, 0	12, 11, 2	S_3'	$DTB \leq 0 \wedge DTR > 2 \wedge DTR \leq 3 \wedge AFHG = False$
8, 5, 1	12, 11, 3	S_4'	$DTF > 2 \wedge DTF \leq 3 \wedge DTR \leq 0 \wedge ALHG = False$
8, 6, 1	12, 10, 3	S_5'	$DTF > 1 \wedge DTF \leq 2 \wedge DTR \leq 0 \wedge ALHG = False$
8, 7, 1	12, 9, 3	S_6'	$DTF \leq 1 \wedge DTR \leq 0 \wedge AFHG = True \wedge ALHG = False$
Goal (8, 8)	Goal (12, 8)	S_7'	$DTB \leq 0 \wedge DTR \leq 0 \wedge AFHG = False \wedge ALHG = False$

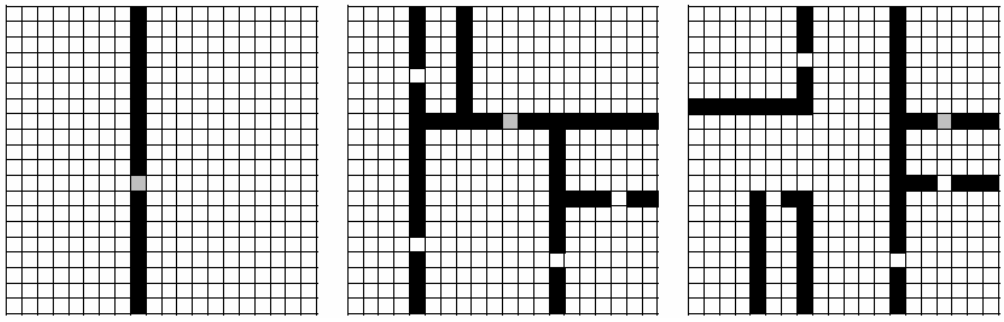
Table 7.2 Few of the Primitive Concepts Captured in the State Mapping Functions.

Literal	Concept Captured
DTF	Distance to Travel in Front
DTR	Distance to Travel in Right
DTB	Distance to Travel in Back
ADFG	Agents Distance from Goal
ALHG	Agent Left Has Goal
AFHG	Agent Front Has Goal

policy and the respective state mapping functions. The literals that make up the individual state mapping function represent primitive concepts. These concepts make up the perceptual signatures that are important for the agent to pay attention to in order to complete the "REACH DOOR X" task. For example, the state (6,5,0) in Grid World 1 and state (14,11,2) in Grid World 2 map onto the abstract state S_1' in the general policy under the state mapping function $DTF > 1 \wedge DTF \leq 2 \wedge ADFG > 4 \wedge ADFG \leq 5 \wedge ALHG = False$ that maps both of these individual states uniquely to the abstract state S_1' in the general policy. The concepts that make these two states homomorphic is that for both of these states the agent has to travel 2 steps in the forward direction towards the goal, the agent's total distance from the goal is greater than 4 steps and less than or equal to 5 steps and the agent's "left has goal" is false. Table 7.2 shows all the important primitive concepts captured in the state mapping function show in Table 7.1.

7.1.3 Reusing Generalized Policies

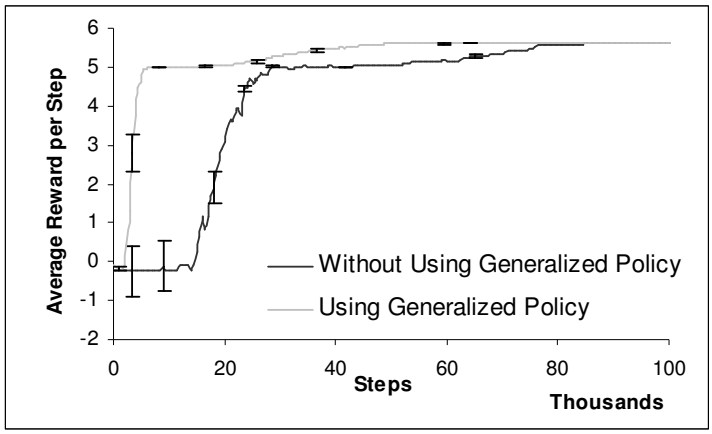
Once the agent has learned a generalized policy it can now use this to learn related new tasks in novel situation. To allow for this to happen, the extracted generalized policy is added as a higher level action to the list of pre existing lower level actions the RL agent can perform. The new grid world domains have a similar setup except that the action space of the agent now has an extra high level action i.e. the generalized "REACH DOOR" policy. The initiation set for the new action consists of all the states that are indicated as homomorphic to states in the general policy by the learned mapping functions.



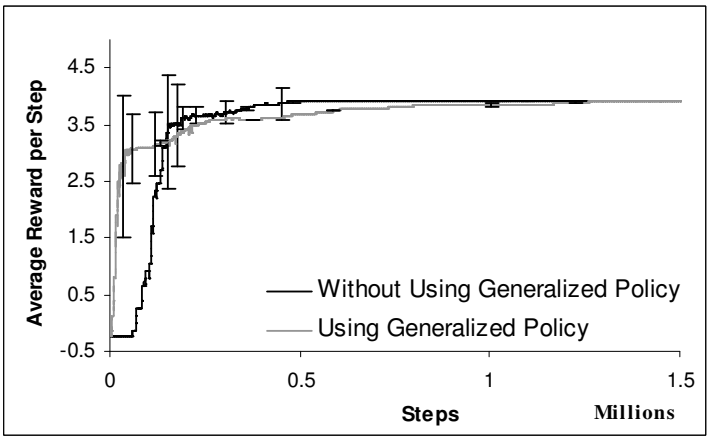
Grid World 1 **Grid World 2** **Grid World 3**

Figure 7.4 20×20 Deterministic Grid Worlds used to Evaluate Generalized Policy

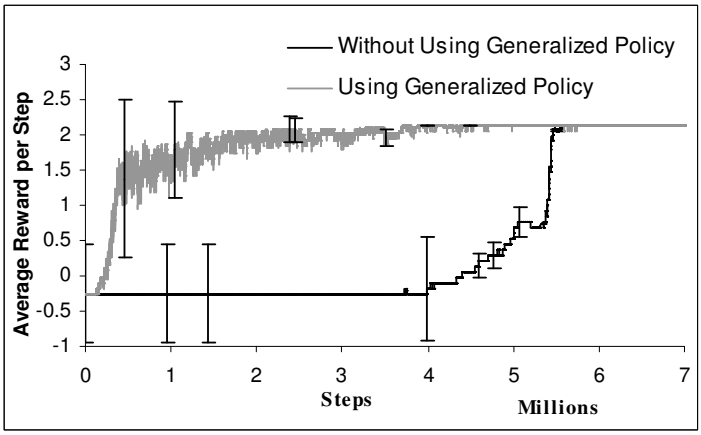
Figure 7.4 shows the set of grid worlds used to evaluate the reuse of the learned generalized policy. The goal doorway is indicated in grey in each of the grid worlds. To learn a policy for this new task the agent starts randomly at a location within the specified grid world and at each time step t perceives the state s_t of the environment and takes an option o_t. Once the option o_t is initialized the further actions are decided by the policy of the option until the option terminates at time $t+k$. As a result the agent receives a reward r_t and the environment transitions to a state s_{t+k}. The agent uses this information along with SMDP learning to learn an optimal policy to reach the new doorway in the novel grid world. Figure 7.5 shows the learning curves for the policies learned by the RL agent for Grid World 1, Grid World 2 and Grid World 3 shown in Figure 7.4. Each graph shows learning curves for an agent learning using only the initial action set (in black) and for an agent learning with the augmented action set including the generalized "REACH DOOR" policy. Each curve is an average of 30 runs and the performance is presented in terms of the average reward per trial that the agent would receive under the policy learned at that point. The vertical confidence intervals indicate one standard deviation in each direction. Each step on the learning curve represents one lower level action. The



(a)



(b)



(c)

Figure 7.5 Learning Curves with and without Using Generalized Policy to Reach (a) the Goal Door in Grid World 1 of Figure 7.4, (b) the Goal Door in Grid World 2 of Figure 7.4, and , (c) the Goal Door in Grid World 3 of Figure 7.4

learning curves show that there is a significant improvement in the time it takes for the agent to learn a policy to reach the goal state when using the generalized policy. These experiments successfully demonstrate the applicablity and usefulness of policy generalization using partial policy homomorphism.

7.2 Experiments in Nondeterministic Grid World Domains

The working of the approximate policy homomorphism approach is demonstrated using examples in a non-deterministic grid world domain. The experiments are divided into three phases. In the first phase we again learn a set of “REACH DOOR X” policies to reach specific doorways using a set of nondeterministic grid worlds with rooms connected by doorways. The agent then uses these specific “REACH DOOR X” policies to abstract a general policy using the approximate partial policy homomorphism framework in the second phase. In the final phase of the experiments we demonstrate how the learned generalized policy can be reused to learn similar tasks in novel environments and situations. To present extraction and reuse of reuse of generalized policies we present results using two tasks in novel grid world scenarios. In the first scenario we use the abstracted generalized policy to learn “REACH DOOR X” in 3 novel grid world environments and in the second we use the generalized policy to learn a “CLEAN ROOMS” task in a novel grid world environment.

For this set of experiments the grid world domain of the agent is non-deterministic and consists of multiple doors. Each location in the grid world is of one of the four types Wall, Empty, Obstacle, and Doorway. The agent in this grid world domain is initialized randomly at one of the empty locations in the grid world. The agent’s action space consists of “FORWARD”, “TURN-LEFT” and ”TURN-RIGHT” actions. Each of these actions in the agent’s action space takes only one time step to complete. When the agent picks an action like FORWARD it successfully reaches the grid location in front of it 80% of the time while it results in the agent

changing its orientation being changed 20 % of time. Similarly, if the agent chooses to execute TURN-LEFT or TURN-RIGHT then the agent successfully turns in the specified direction 80 % of the time, 10% of the time it reaches the grid location in front of it and the remaining 10 % of the time it results in the agent turning in the opposite direction to the specified action. The state space of the agent consists of the X,Y location of the agent, the orientation of the agent and the X,Y locations of the four doors nearest to the agent. At each point in time the agent observes the state of the environment and chooses an action to perform. For each action the agent performs in this grid world domain, it incurs a cost of -0.25. In addition the agent receives a reward of +100 from the environment whenever it reaches a door that is identified as a goal state.

7.2.1 Learning Basic Policies

Figure 7.6 shows the set of grid worlds used by the RL agent to learn policies to reach specific doorways. In this phase of the experiments the agent learns a set of situation-specific "REACH DOOR X" policies to reach specific doorways. To learn a policy to the specified

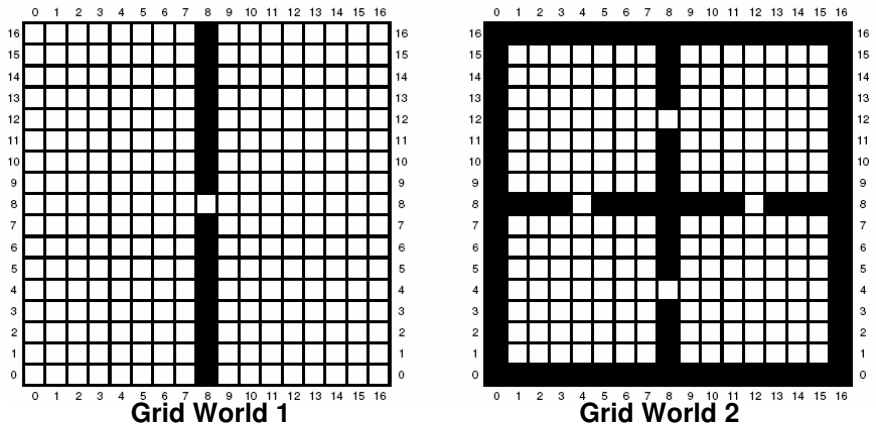


Figure 7.6 17×17 Non –Deterministic Grid Worlds used for Learning Situation-Specific Policies for Doorways.

doorway the agent starts at a random empty location within the grid world and explores the world. During each step of learning the agent uses Q-learning to update the Q value for the

state action pair visited. As a result of learning in both of the worlds with each of the doorways as goal doorways, the agent again learns a total of 5 basic policies to reach each of the individual doorways within these grid worlds.

7.2.2 Extracting a Generalized Policy

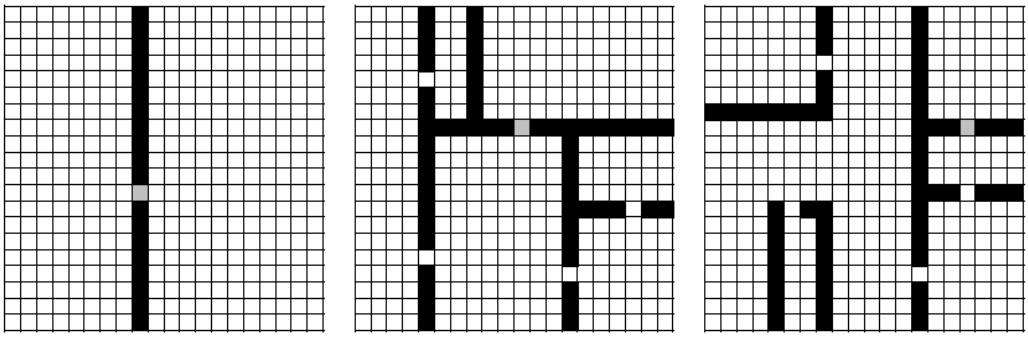
In this phase of the experiments the agent uses the learned policies to extract a general policy. To do this the agent first enhances the states within the learned policies by adding new state attributes. The new state attributes that are added, are formed by applying various operators e.g. logical, relational and arithmetic operators to an attribute or a set of attributes of the original state representation. This step is aimed at capturing the information that cannot be acquired by just using the original state attributes but may be important to capture the information relevant for successfully completion of a *task type*. The policy abstraction algorithm uses this state representation to extract a general mapping function without the need for a complex function approximator. To achieve this, the policy abstraction algorithm starts at the goal state of each policy and builds a state mapping function that maps each state s_t of a policy instance to a unique abstract state s'_t of the abstract policy. Also, at each step the abstraction algorithm builds an action mapping function that maps the action a_t that results in state s_t to transition to s_{t+1} in a basic policy to an abstract action a'_t that transitions the abstract policy from the corresponding state s'_t to s'_{t+1}. The algorithm uses a greedy method to build the functions. This is aimed at keeping the method tractable and continues until it reaches a point where the addition of more states no longer increases the expected utility of the general policy.

7.2.3 Reusing Generalized Policies

In this phase we demonstrate the reuse of the abstracted general policy to learn two new tasks. The first being a task to “REACH DOOR” task in a novel environment and the second is a “CLEAN ROOMS” task.

7.2.3.1 Learning "REACH DOOR" Task Using the Abstracted Generalized Policy

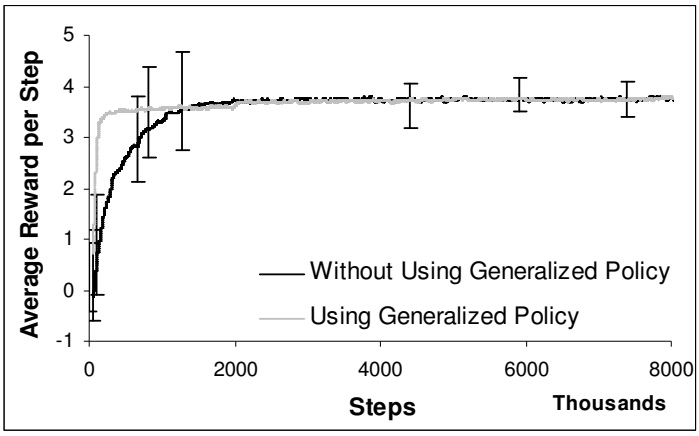
Figure 7.7 shows the new grid worlds used to learn "Reach Door" policies to new doors in the current environment using the generalized policy. The grid worlds used in the current experiment have the same properties as the one used to learn the situation-specific policies. The only difference is that the agent's action set has the additional higher level action besides the primitive actions that were present during the learning of the situation-specific policies to reach doorways.



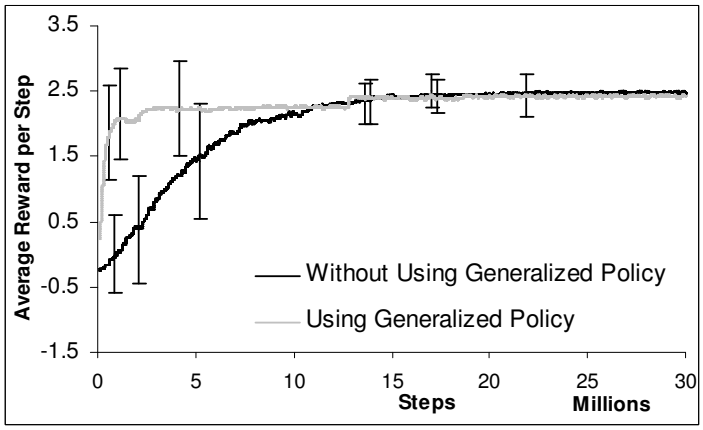
Grid World 1 **Grid World 2** **Grid World 3**

Figure 7.7 20×20 Non-Deterministic Grid Worlds used to Evaluate Generalized Policy

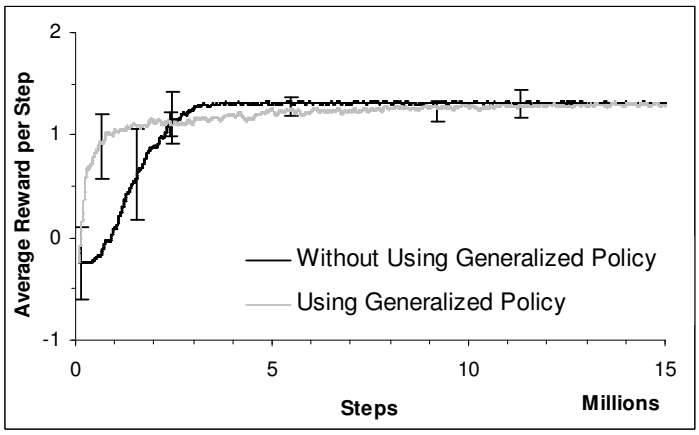
The RL agent can pick any of the lower level actions to execute at each time step and the availability of a higher level action in a state depends on whether the generalized policy is applicable in the given situation, which is determined by the abstract state of the generalized policy. The agent starts from a random location within each grid world and uses SMDP learning to learn a policy to reach the identified goal door.



(a)



(b)



(c)

Figure 7.8 Learning Curves with and without Using Generalized Policy to Reach (a) the Goal Door in Grid World 1 of Figure 7.7, (b) the Goal Door in Grid World 2 of Figure 7.7, and (c) the Goal Door in Grid World 3 of Figure 7.7

Figure 7.8 shows the learning curves for the policy to a goal door in the novel grid worlds with and without using the generalized "REACH DOOR" policy. Each curve is an average of 30 runs and the performance is presented in terms of the average reward per trial that the agent would receive under the policy learned at that point. The vertical confidence intervals indicate one standard deviation in each direction. Each step on the learning curve represents one lower level.The learning curves show that there is a significant improvement in the time it takes for the agent to learn a policy to reach the goal state when using the generalized policy.

7.2.3.2 Learning "CLEAN ROOMS" Task Using the Abstracted Generalized Policy

Figure 7.9 shows the grid world environment used for learning the cleaning task. In this task the agent has to learn a policy to pick up the "Blue" and "Red" objects and drop them in the "Grey" colored trash can. To learn this task the agent uses the extracted general "REACH DOOR" policy as a higher level action. Also, it has one additional primitive action PICK-DROP along with the other primitive actions. Action PICK-DROP always succeeds with a probability of 1. The other properties of the grid world are the same as to the ones that were used during the process of learning situation-specific "REACH DOOR" policies. The agent starts from a random location within the grid world and uses SMDP learning to learn a policy to clean the rooms by picking up and dropping objects in the trash can.

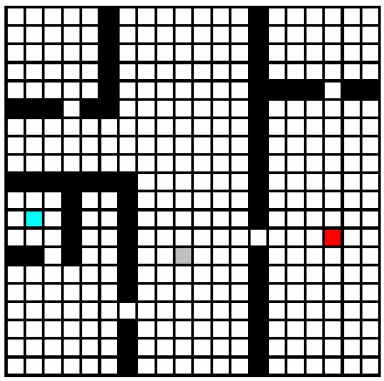


Figure 7.9 20×20 Non –Deterministic Grid World Used for the "CLEAN ROOMS" Task

Figure 7.10 shows the learning curves for the cleaning task with and without the generalized policy. Each curve is an average of 30 runs and the performance is presented in terms of the average reward per trial that the agent would receive under the policy learned at that point. Each step on the learning curve represents one lower level action.The vertical confidence intervals indicate one standard deviation in each direction. The learning curves show that there is

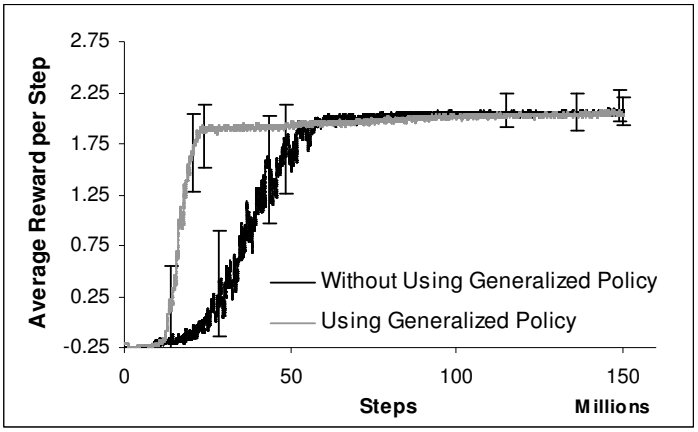


Figure 7.10 Learning Curves for "CLEAN ROOMS" Task with and without using the Generalized "REACH DOOR" Policy.

a significant improvement in the time it takes for the agent to learn a policy to reach the goal state when using the generalized policy.

7.3 Autonomous Categorization and Generalization of Policies Based on Task Types

Although the policy homomorphism framework provides a life long learning RL agent with the ability to extract reusable skills and concepts from a set of situation-specific policies for similar *task types*, only having this ability is not enough for an agent to successfully learn, adapt, and perform in a complex and dynamic environment. Besides this they need to identify if a given set of policies learned needs to be abstracted and when and whether the abstracted general policy or policies will be useful if saved. In addition, these have to autonomously identify which of the situation-specific policies belong to a common *task type* and how many *task type* they

represent and then extract reusable skills and concepts for these different *task types*. Finally, they have to know which of the previously generalized policies abstracted over time are still useful and should be included in the action set to learn new tasks. This is achieved in our policy homomorphism framework by incorporating a new utility-based policy evaluation criterion. The criterion allows the agent to evaluate existing and newly identified generalized policies in terms of their usefulness.

7.3.1 Autonomous Categorization and Generalization of Policies Based on *Task Types* in a Deterministic Environment

To illustrate the working of the policy homomorphism framework for autonomous categorization and extraction of general policies for different *task types* in a deterministic environment the agent here learns situation-specific policies for 2 different tasks characterized by different reward functions in the same environment. In the first task ,"Task1", the agent has to learn situation-specific policies to reach the goal doorway (identified as 'G') for the Grid Worlds shown in Figure 7.11. In the second task ,"Task2", the agent has to not only learn to reach goal doorway for the Grid Worlds shown in Figure 7.11 but it also has to learn this while keeping a certain distance away from other door within the same room. This task is characterized by an additional negative reward that the agent receives when it comes close to the wrong door.

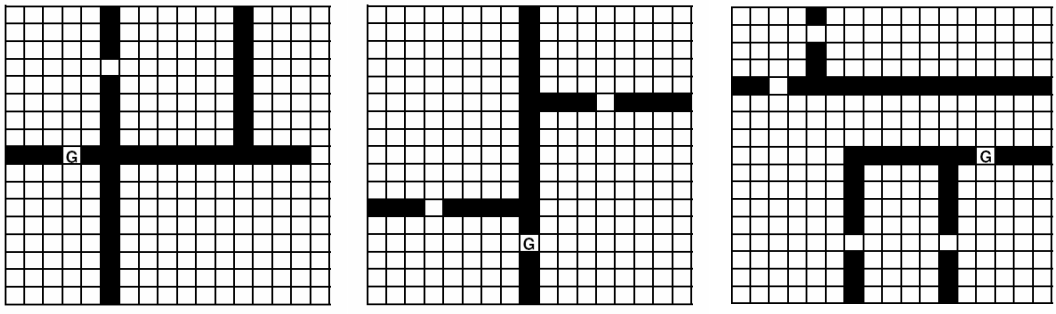


Figure 7.11 Grid Worlds Used for Learning Situation-Specific Policies for "Task1" and "Task2"

For both of these tasks the grid world domain of the agent is deterministic and consists of multiple doors. Each location in the grid world is of one of the four types Wall, Empty, Obstacle, and Doorway. The agent in this grid world domain is initialized randomly at one of the empty locations in the grid world. The agent's action space consists of "FORWARD", "TURN-LEFT", and "TURN-RIGHT" actions. Each of these actions in the agent's action space takes only one time step to complete. The state space of the agent consists of the X,Y location of the agent, orientation and the X,Y locations of the four doors nearest to the agent. At each point in time the agent observes the state of the environment and chooses an action to perform. For each action the agent performs in this grid world domain, it incurs a cost of -0.25. In addition the agent receives a reward of +100 from the environment whenever it reaches a door that is identified as a goal state. In "Task 2" the agent receives additional negative reward of -5 if it at distance of four or less from a door that is in the same room as that of the goal door. The agent starts at a random location and learns policies for "Task 1" and "Task 2" for each of the grid worlds shown in Figure 7.11. As a result of learning the agent acquires 6 specific policies.

These situations specific policies are now used by the agent's policy abstraction component to abstract general policies, with one general policy for each of the identified *task types*. To do this, similar to the previous experiments each state is enhanced by adding additional attributes that are formed by manipulating the original state attributes. Once all the states in each of the policies go through this preprocessing they are used by the policy abstraction process to abstract a general policy. The abstraction process starts at the goal state and extracts a state mapping and an action mapping function that maps each individual state and action of the situation-specific policies to the abstract state and abstract action of the general policy. When the six situation-specific policies learned are abstracted using the policy homomorphism frame work, initially they can be abstracted to one single general policy GP1. However, at some point along the incremental, greedy building process no further extension of

the general policies is possible that would encompass all the situation-specific policies, representing the maximal general policy GP1 under the assumption that all policy instances are of the same *task types.* At this point, the utility based abstraction mechanism splits the policy set into multiple maximal sets such that it can further expand on a general policy in each of the sets separately. In this case this leads to a split into two sets and the homomorphism framework continues to extract two generalized policies GP2 and GP3 for these these sets. This process continues until the resulting sets contain only one policy instance which in this example occurs directly after GP2 and GP3. Figure 7.12, Figure 7.13 and Figure 7.14 shows the regions of the

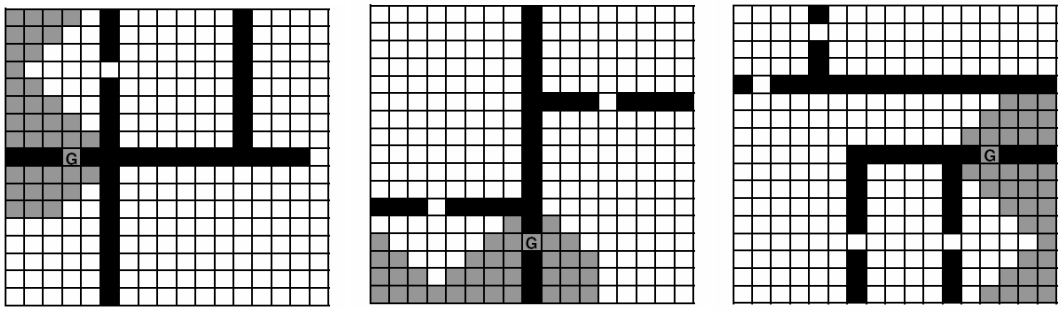


Figure 7.12 Parts of the State Space Around the Goal Doors Abstracted in the Generalized Policy GP1 (Regions Shown in Grey).

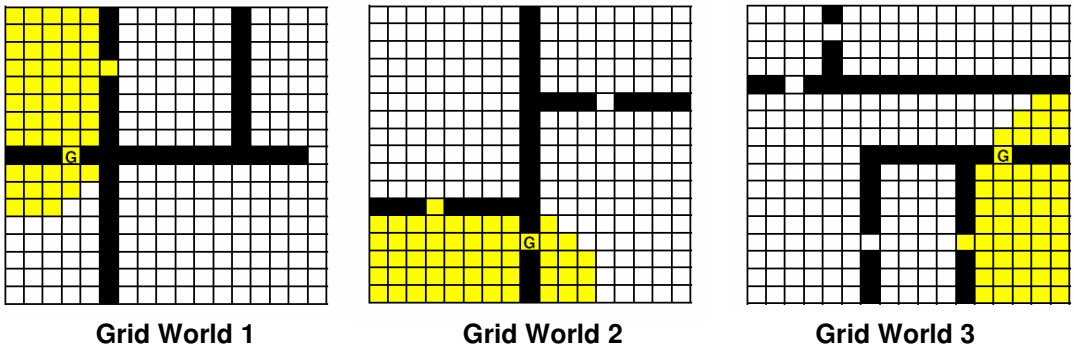


Figure 7.13 Parts of the State Space Around the Goal Doors Abstracted in the Generalized Policy GP2 (Regions Shown in Yellow).

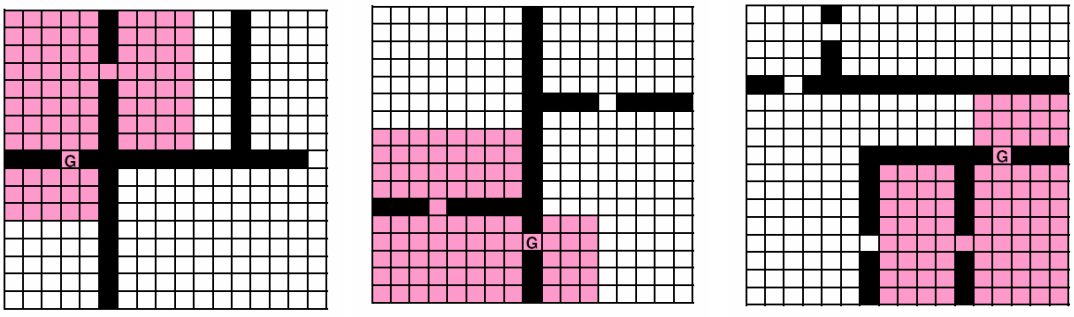


Figure 7.14 Parts of the State Space Around the Goal Doors Abstracted in the Generalized Policy GP3 (Regions Shown in Pink).

state space around goal doors within each of the grid worlds 1, 2 and 3 that are abstracted in generalized policy GP1, GP2 and GP3. During the process of abstracting the generalized policies, the policy homomorphism framework also calculates the utility of the sets of policies to make a decision about the optimal abstracted set of generalized policies and with them about the number and identity of *task types* represented in the task instances (and their corresponding specific policies). The utility is calculated here empirically using the trajectories generated from the learned situation-specific policies for these tasks instances. The calculation in this scenario for set one that contain only GP 1 yields a utility of 10.67 while it leads to a utility of 14.77 for the second set of generalized policies which contain GP2 and GP3, thus, saving the second set of generalized policies in the skills memory. This higher utility for the second set here implies that the agent divides the task instances into two *task types* which correctly represent the two different task scenarios embedded in the originally learned policies.

7.3.2 Autonomous Categorization and Generalization of Policies Based on *Task Types* in a Non-Deterministic Environment

To demonstrate of the autonomous categorization and generalization of approximate policies based on *task types* a similar experiment the one shown in Section 7.3.1 is performed here where the agent learn situation-specific policies for 2 different tasks in a non-deterministic

environment. In the first task, “Task1”, the agent has to learn situation-specific policies to reach of the goal doorway (represented by ‘G’) in Grid Worlds shown in Figure 7.15. In the second task, “Task2”, the agent has to not only learn to reach the goal doorway (represented by ‘G’) but also has to maintain a certain minimum distance from others doors within the same rooms in the grid worlds shown in Figure 7.15.

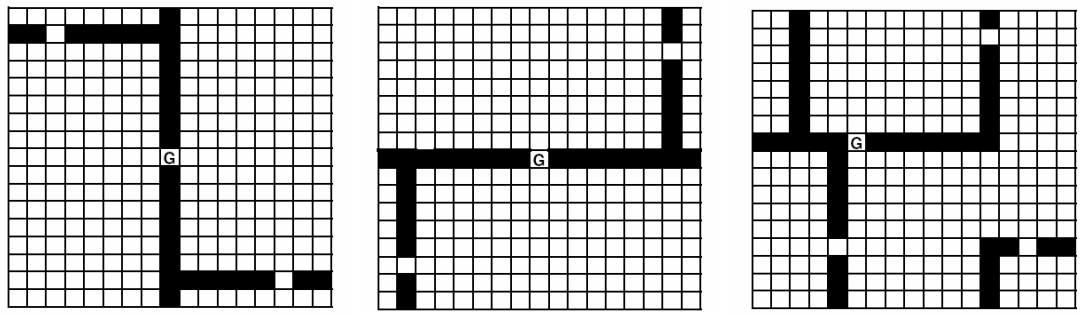


Figure 7.15 Non-Deterministic Grid Worlds used for Learning Situation-Specific Policies for “Task1” and “Task2”.

For both of these tasks the grid world domain of the agent is non-deterministic. Each location in the grid world is of one of the four types Wall, Empty, Obstacle, and Doorway. The agent in this grid world domain is initialized randomly at one of the empty locations in the grid world. The agent’s action space consists of “FORWARD”, “TURN-LEFT”, and ”TURN-RIGHT” actions. When the agent picks an action like FORWARD then the agent successfully reaches the grid location in front of it 80% of the time and results in the agent’s orientation being changed 20 % of the time. The state space of the agent consists of the X,Y location of the agent, its orientation and the X,Y locations of the four doors nearest to the agent. At each point in time the agent observes the state of the environment and chooses an action to perform. For each action the agent performs in this grid world domain, it incurs a cost of -0.25. In addition the agent receives a reward of +100 from the environment whenever it reaches a door that is identified as a goal state. The agent starts at a random location and learns a policy to reach the

goal doorway for each of the grid worlds shown in Figure 7.15. As a result of learning the agent acquires 6 specific policies.

Similar to the deterministic scenario, the state space of policy is enhanced by adding additional state attributes that are formed by the manipulation of the original state attributes. The abstraction process starts at the goal state and extracts a state mapping and action mapping function that maps each individual state and action of the situation-specific policies to the abstract state and abstract action of the general policy. During the initial stages of abstraction all the policies are abstracted into a single generalized policy GP1. However, as the process continues, the abstraction process can no longer generalize them into one general policy GP1 and divides the set of situation-specific policies is again split, leading to a set of two generalized policies GP2 and GP3. Figure 7.16, Figure 7.17 and Figure 7.18 show the regions of the state space around goal doors within each of the grid worlds 1, 2 and 3 that are abstracted in generalized policies GP1, GP2 and GP3. During the process of abstracting the generalized policies, the policy homomorphism

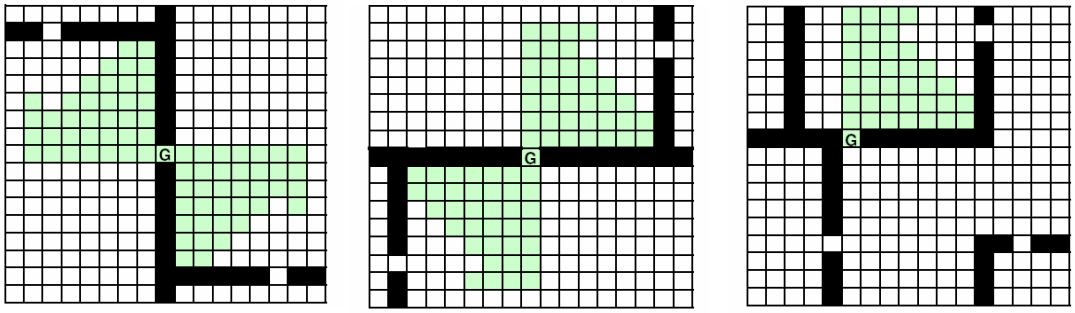


Figure 7.16 Parts of the State Space Around the Goal Doors Abstracted in the Approximate Generalized Policy GP1 (Regions Shown in Green).

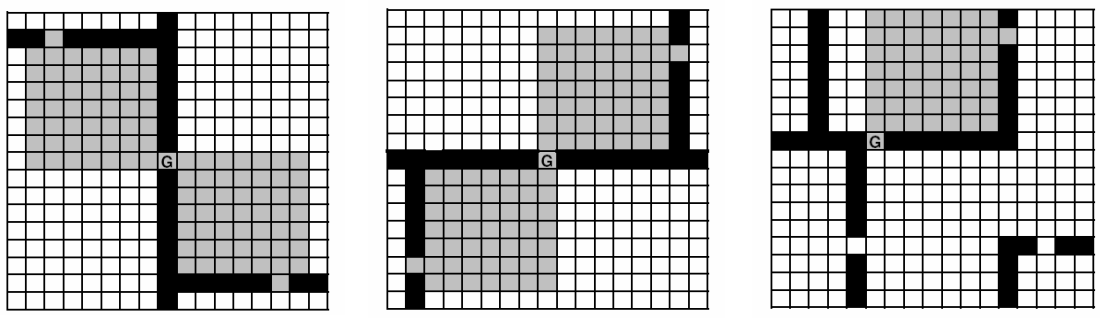


Figure 7.17 Parts of the State Space Around the Goal Doors Abstracted in the Approximate Generalized Policy GP2 (Regions Shown in Grey).

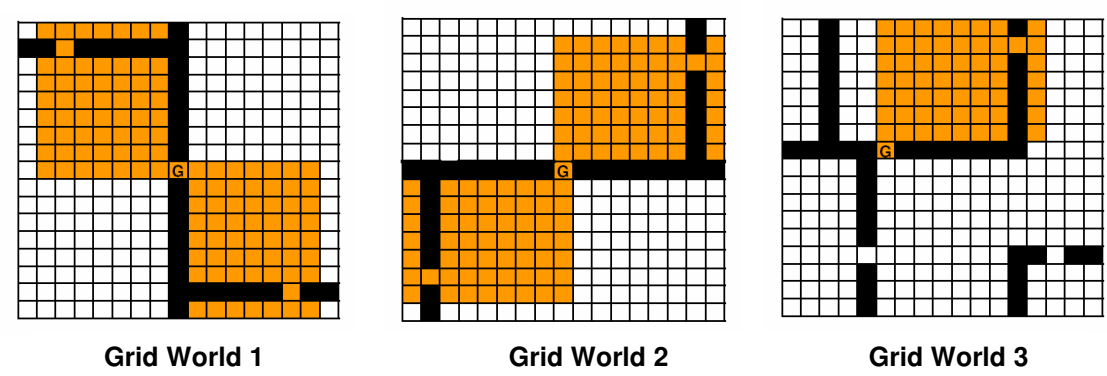


Figure 7.18 Parts of the State Space Around the Goal Doors Abstracted in the Approximate Generalized Policy GP3 (Regions Shown in Orange).

framework also calculates the utility of the sets of policies to make a decision about the optimal level of abstraction and thus on the best abstracted set of generalized policies and *task types*. The utility is calculated empirically using the trajectories generated from the learned situation-specific policies for the tasks instances. The utility in this scenario for set one that contains only GP 1 yields a utility of 13.67 while a utility of 15.454 is obtained for the second set of generalized policies which contains GP2 and GP3, thus indicating that there are two different *task types* represented in the set of situation-specific policies that were used for abstraction.

This demonstrates that the agent using the policy homomorphism framework is able to autonomously categorize *task types* using the policy utility of the generalized policies, alleviating the need to learn situation-specific task instances in order or to provide explicit information to the agent regarding the type of task it is currently engaged in. This, in turn provide a life long learning agent with a initial piece of decision autonomy.

7.3.3 Learning "CLEAN ROOMS" Task in a Three-Dimensional Non-Deterministic Grid World

To demonstrate the end to end performance of an RL agent that classifier extracts, and reuses multiple generalized policies over its lifetime to learn related task in a novel environment, the RL agent in this experiment initially learns set of situation-specific policies to reach doors, pickup objects and dropoff objects. The agent's initial action space consists of "FORWARD", "TURN-LEFT", "TURN-RIGHT", "PICK" and "DROP" actions. All actions are probabilistic. Actions "FORWARD", "TURN-LEFT", and "TURN-RIGHT" succeed with a probability of 0.8 and fail with a probability of 0.2. Actions "PICK" and "DROP" succeed with a probability of 1. Each action costs the agent -0.25 and the agent receives a reward of +100 for reaching the goal

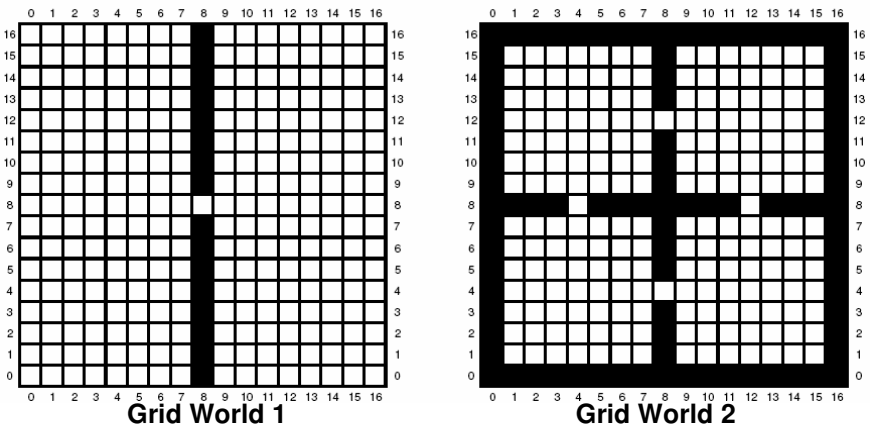


Figure 7.19 Grid Worlds used to Learn Situation-Specific Policies to Reach Doorways, Pick Objects, and Drop Objects

state (either a location, holding an object, or releasing an object, depending on the type of task instance). The agent's state space consists of the X, Y location of the agent, its orientation, the X, Y locations of the four nearest doors to the agent, the X, Y location, color and the holding bit of the four nearest objects to the agent. The agent starts at random locations within grid world domain shown in Figure 7.19 and learns situation-specific policies to reach of the doorways. Similarly it also learns one situation-specific policies to pickup objects and drop off objects with random colors placed at location (8, 3) , (13, 13) in Grid World 1 and (4, 4), (12, 4), (4, 12) (12, 12) in Grid World 2. These situation-specific policies are then used by the policy homomorphism framework to extract general policies of each identified *task type*. At the end of the abstraction process the agent extracts a "REACH DOOR", "PICK OBJECT" and "DROP OBJECT" general policy. These three abstracted generalized policies are then reused to learn a policy for a cleaning task in three-dimensional grid world domain shown in Figure 7.20. The lines connecting the grid location of "FLOOR 1" and "FLOOR 2" indicates a stairway connection between the floors at that location. The goal of the agent for this task is to pick up all the objects

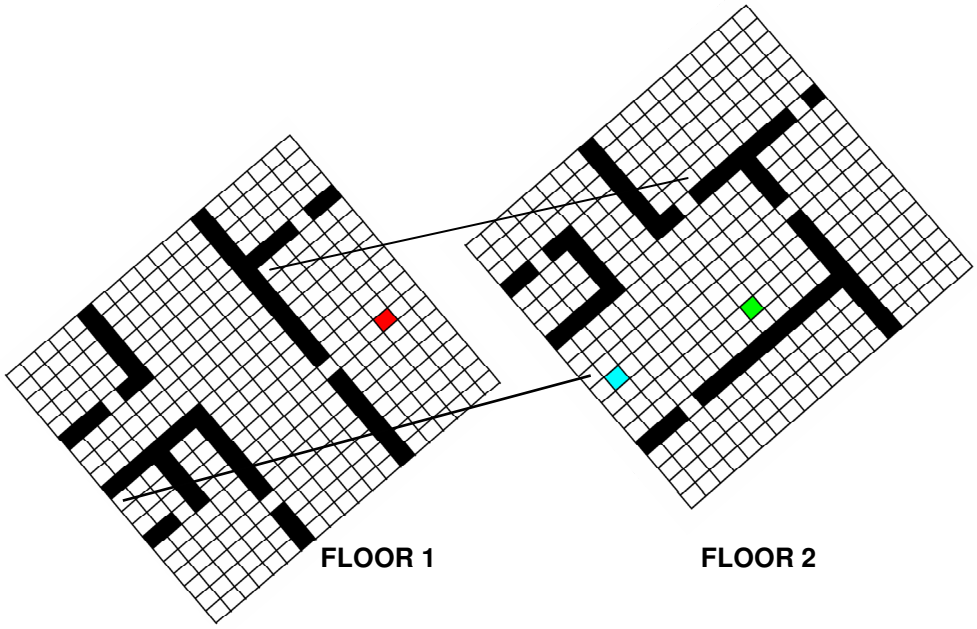


Figure 7.20 Three-Dimensional Non-Deterministic Grid World Domain

within the grid world domain and drop them in the green colored trash can. For this task, the agent's action space consists of the extracted general policies as higher level actions, and an extra "UP/ DOWN" primitive action besides the previous primitive actions. The new action succeeds with probability 1. The other properties of the grid world domain are the same as to the ones that were used during the process of learning situation-specific policies to reach doors, pick up objects and drop objects. The agent's state space now not only contains the X,Y location and orientation of the agent, the four nearest doors, and the four nearest objects but it also contains one extra variable for each agent , door and object to identify the floor level it is located at in this grid world domain. To learn the task the agent starts from a random location within the grid world and uses SMDP learning to learn a policy to clean the rooms by picking up and dropping objects in the trash can. Figure 7.21 shows the learning curves for the cleaning task with and without the generalized policy. Each curve is an average of 30 runs and the performance is presented in terms of the average reward per trial that the agent would receive under the policy learned at that point. Each step on the learning curve represents one lower level action. The vertical confidence intervals indicate one standard deviation in each direction. The learning curves show that the agent with the "REACH DOOR", "PICK OBJECT" and "DROP

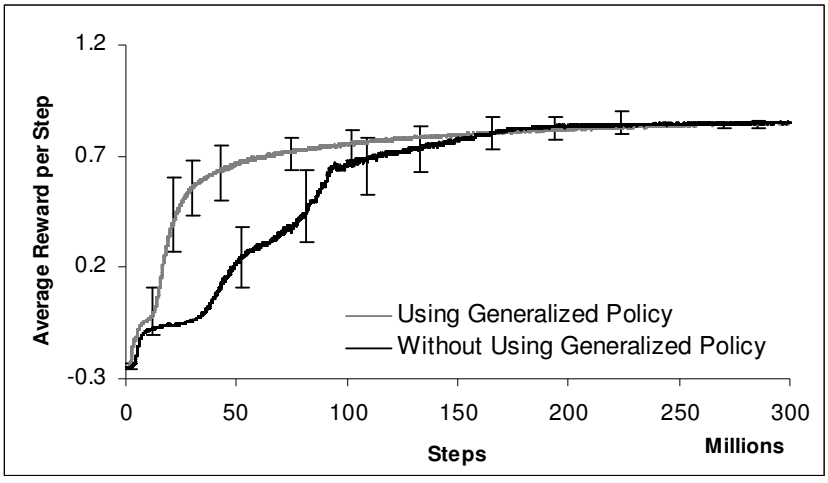


Figure 7.21 Learning Curves for "CLEAN ROOMS" Task in the Three-Dimensional Grid World Domain

OBJECT" generalized policies learns the task much faster than the simple agent that only has primitive actions.

The experiment in this chapter demonstrates that the policy homomorphism framework is successfully able to identify absolute, partial or approximately homomorphic policies and to extract a general policy from the situation-specific policies of each *task type*. The extracted general policies can further be successfully reused to learn similar tasks in novel environments. The results also demonstrate that reuse of knowledge gained to learn new tasks reduces the time it takes to learn them. Furthermore, it allows the agent to compress the state space by abstracting information important for task completion. Extending the policy homomorphism framework by incorporating a criterion that evaluates the usefulness of the policies enables the RL agent to autonomously decide on *task types*. This criterion further enables making decisions on the action set based on the usefulness of generalized policies.

CHAPTER 8

CONCLUSION AND FUTURE WORK

A life-long learning RL agent that performs tasks in the real world needs to continuously learn new tasks. In addition, these agents need to adapt what they have learned to perform successfully in a complex and dynamic environment. While traditional RL agents have the capability to learn new tasks, they face significant challenges in terms of scaling to complex domains because the policies learned by these agents do frequently not transfer and therefore the knowledge gained from learning to perform a given task in a specific situation cannot be used to learn related new tasks in novel scenarios. The main reason for this is that the policies learned are usually directly tied to the perceptual representation of the environment and as a result the policies become useless as soon as the environment or the way the perceptual information is represented changes. Another issue with traditional RL agents is that they often use the raw perceptual information to learn policies. These percepts, however, produce a huge amount of data and processing and basing decisions on this perceptual information can easily become a computationally intractable problem in complex environments. Furthermore, traditional RL agents generally make decisions as to what action they need to perform at each point in time. However, reasoning about actions at this scale and performing them in real time can become impossible as the complexity of the tasks these agents are learning increases.

This dissertation introduces the novel approach of policy homomorphism for policy generalization. The policy homomorphism framework uses a set of situation-specific policies to abstract reusable skills and concepts in the form of a general policy. The extracted general policy is represented by a set of functions. The first set of functions identifies situations where

the abstracted general policy is applicable and the second set of functions identifies the actions that need to be performed from each of the identified situations of the general policy to achieve its goal. The extracted concepts make up the perceptual signatures that are important for the agent to pay attention to in order to complete a given task. t extracted, the abstracted general policy can then be used as a higher level action to learn policies for related tasks in novel environments. The use of general policies as higher level actions allows the agent to abstract decisions to a higher level where the generalized policy is applicable and allows it to ignore the details of what primitive actions to take at each point in time (which is now decided by the generalized policy). To make this framework applicable for a life-long learning RL agent that encounters a range of different tasks over its lifetime, we further define a utility criterion which is used by the policy generalization component of the agent's learning framework to autonomously generalize and categorize skills and concepts. This criterion allows the agent to autonomously determine the different *task types* represented in the set of learned task specific policies and to learn a set of generalized policies for them that correctly reflect the task challenges encountered by the agent. The experiments performed in deterministic and non-deterministic domains demonstrate that the policy homomorphism can be successfully used to abstract a partial or approximate partial general policy from a set of basic policies of similar *task type*. Applying the learned general policies in novel grid world domains where the agent learns policies for related tasks with and without using the abstracted generalized policy shows that using a generalized policy results in, significant improvement in the amount of time the agent takes to learn a policy compared to the situation where the agent just uses the primitive actions. Experiments where the agent first learns tasks instances with different objectives also demonstrates that the utility criterion can be used to successfully identify and categorize skills and concepts and that transfer of the extracted abstract policies leads to a significant improvement when learning a new, more complex task in novel environment. Together these experiments illustrate the

potential of the presented framework to transfer control knowledge, learn new tasks, and facilitate scaling to more complex task domains.

Though this dissertation presents methods that allow extracting a set of general policies from a situation-specific policy instances, this dissertation still does not directly address how these generalized policies are managed and how they can be further used to build higher, and multiple levels of abstractions. As a result one of the future goals that can directly result from this work is finding methods that would allow building, learning and efficiently managing skills and concepts at multiple levels of abstraction. We also need to investigate how the utility of the generalized policies can be efficiently used to control the exploration rate during the learning of a new, related task. Also, there is need to devise methods that would allow to predict accurately which of the learned generalized polices is more useful given a situation when multiple of them are available. Finally, we need more investigation on how the complexity of decision making increases as the agent continuously learns more and more generalized policy increases.

REFERENCES

[Amarel 1968] Amarel, S. On representations of problems of reasoning about actions, Machine Intelligence, Volume 3, pp 131-171, 1968.

[Anzai and Simon 1979] Anzai Y., and Simon H.A, The Theory of Learning by Doing, Psychological Review, Volume 86(2), pp-120-140, 1979.

[Andre and Russell 2001] Andre D., and Russell S., Programmable Reinforcement Learning Agents, In Proceedings of Advances in Neural Information Processing Systems 13, pp 1019-1025, 2001.

[Asadi and Huber 2007] Asadi M., and Huber M., "Effective Control Knowledge Transfer Through Learning Skill and Representation Hierarchies", In the Proceedings of IJCAI, 2007, pp. 2054-2059.

[Barto et al. 1981] Barto A.G., Sutton R.S., and Brouwer P.S.. Associative search network: A reinforcement learning associative memory. Biological Cybernatics, Vol 40 (3), pp 201-211,1981.

[Bellman 1957a] Bellman R.E., A Markovian Decision Process. Journal of Mathematics and Mechanics, Vol 6, pp 679-684, 1957.

[Bellman 1957b] R. E. Bellman. Dynamic Programming. Princeton University Press, Princeton, NJ, 1957.

[Brooks 1986] R. A. Brooks. Achieving artificial intelligence through building robots. Technical Report A.I. Memo 899, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, MA, 1986.

[Bradtke 1995] Bradtke S.J., and Duff M.O., Reinforcement Learning Methods for Continuous Time Markov Decision Problems, In Advances in Neural Information Processing Systems 7, pp 393-400, 1995.

[Bakker 2004] Bakker B., and Schmidhuber J., "Hierarchical Reinforcement Learning, based on Subgoal Discovery and Subpolicy Specialization", In the Proc. of IAS, Vol. 8, pp 438-445, 2004.

[Dayan and Hinton 1993] Dayan P., and Hinton G.E., Feudal Reinforcement Learning, Advances in Neural Information Processing Systems 5, pp 271-278, 1993.

[Dean et al. 1997] T. Dean, R. Givan, and S. Leach, Model Reduction Techniques for Computing Approximately optimal Solutions for Markov Decision Processes, In Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97), pp 124–131, 1997.

[Dietterich 2000] Dietterich, T. G., An Overview of MAXQ Hierarchical Reinforcement Learning. Lecture Notes in Computer Science, 1864, pp 26–44, 2000.

[Digney 1996] Digney, B., Emergent Hierarchical Control Structures: Learning Reactive/Hierarchical Relationships in Reinforcement Environments, From Animals to Animats 4:

In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: SAB 96, MIT Press, Bradford Books, pp 363-372, 1996.

[Digney 1998] Digney, B., Learning Hierarchical Control Structures for Multiple Tasks and Changing Environments, From Animals to Animats 4:In Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior: SAB 98, MIT Press, 1998.

[Emerson and Sistla 1996] E. A. Emerson and A. P. Sistla. Symmetry and model checking. Formal Methods in System Design, Vol 9(1/2), pp 105–131, 1996.

[Fikes et al. 1972] Fikes R.E., Hart P.E. and Nilsson N.J., Learning and Executing Generalized Robot Plans, Arficial Intelligence, Volume (3), pp 251-288, 1972.

[Gibson 1977] Gibson J., The theory of affordances, In R. Shaw & J. Bransford (eds.), Perceiving, Acting and Knowing, Hillsdale, New Jersey: Erlbaum, 1977.

[Givan et al. 2000] Givan R., Leach S., Dean T., Bounded Parameter Markov Decision Processes, Artificial Intelligence, Vol 122 (1-2), pp 71-109, 2000.

[Gullapalli 1992] Gullapalli V., Reinforcement Learning and its Application to Control, Ph.D. Thesis, University of Massachusetts Amherst, 1992.

[Goel and Huber 2003] Goel S., Huber M., Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies. In Proceedings of the 16th International FLAIRS Conference. AAAI Press, pp 346-350, 2003.

[Huber and Grupen 1997] Manfred Huber and Roderic A. Grupen. Learning to coordinate controllers - reinforcement learning on a control basis. In IJCAI, pages 1366–1371, 1997.

[Hauskrecht et al. 1998] Hauskrecht M., Meuleau N., Boutilier C., Kaelbling L.P., and Dean T., Hierarchical Solution of Markov Decision Processes Using Macro-Actions, In Proceedings of the fourteenth Annual Conference on Uncertainty in Artificial Intelligence, pp 220-229, 1998.

[Jong and Stone 2005] Jong, N. K., and Stone, P., "State Abstraction Discovery from Irrelevant State Variables", In the Proc. of IJCAI, pp 752-757, 2005.

[Korf 1985 a] Korf R. E., Macro –Operators: A Weak Method for Learning. Artificial Intelligence, Volume 26, pp 35-77, 1985.

[Korf 1985 b] Korf R.E., Learning to Solve Problems by Searching for Macro-Operators. Pitman, Boston, MA, 1985.

[Knoblock 1990] Knoblock C.A., Learning Abstraction Hierarchies for Problem Solving, In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI), Volume 2, pp 923-928, 1990.

[Knoblock 1991] Knoblock C.A., Automatically Generating Abstractions for Problem Solving. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Technical Report CMU-CS-91-120, 1991.

[Kaelbling 1993] Kaelbling L., Hierarchical Learning In Stochastic Domains: Preliminary Results, In Proceeding of the Tenth International Conference on Machine Learning, pp 167-173, 1993.

[Kaelbling et al. 1996] Kaelbling, L.P., Littman, M.L., and Moore, A.W., "Reinforcement Learning: A Survey", Journal of Artificial Intelligence Research, Volume 4, 1996.

[Kim and Dean 2003] Kim, K., & Dean, T., Solving Factored MDPs using Non-Homogeneous Partitions. Artificial Intelligence, 147, 225–251, 2003.

[Konidaris and Barto 2006] Konidaris G, and Barto A, Building Portable Options: Skill Transfer in Reinforcement Learning, 2006, Tech. Report 2006-17.

[Laird et al 1986] Laird, J. E., Rosenbloom, P. S., and Newell, A. Chunking in Soar: the anatomy of a general learning mechanism. Machine Learning, 1(1), pp 11-46, 1986.

[Lakoff 1987] Lakoff, G., Women, Fire, and Dangerous Things. University of Chicago Press, 1987.

[Minton 1988] Minton S., Learning Search Control Knowledge: An Explanation-Based Approach, Kluwer Academic Publishers, 1988.

[Mandler 1992] Mandler, J. M., How to build a baby: II. Conceptual Primitives. Psychological Review 99(4), pp 587-604, 1992.

[Mitchell 1997] Mitchell T., "Machine Learning", McGraw Hill Publications, 1997.

[McGovern et al. 1997] McGovern A., Sutton R. S., and Fagg A.H., Roles of Macro-Action in Accelerating Reinfrocment Learning, In Proceedings of the 1997 Grace Hopper Celebration of Women in Computing, pp 13-18, 1997.

[**McGovern and Barto 2001]** McGovern A., Barto, A. G., Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density, In Proceedings of the eighteenth international conference on machine learning, pp 361-368, 2001.

[Moore et al. 1999] Moore A.W., Baird L.C., and Kaelbling L., Multi-Value-Functions: Efficient Automatic Action Hierarchies for Multiple Goal MDPs. In Dean T., editor, Proceeding of the International Joint Conference on Artificial Intelligence, Volume 2, pp 1316-1321, 1999.

[Newell andn Simon 1963] Newell,A., Simon, H. A, GPS: A Program that Simulates Human Thought", in Feigenbaum, E.A.; Feldman, J., Computers andThought , McGraw-Hill, 1963.

[Piaget 1952] Piaget J., The Origins of Intelligence in Children. Madison, CT, International Universities Press, 1952.

[Precup and Sutton 1997] Precup D., and Sutton R.S., Multi-Time Models for Temporally Abstract Planning, In Proceedings of Advances in Neural Information Processing Systems 10, pp 1050-1056, 1997.

[Precup et al. 1998] Precup D., Sutton R.S., and Singh S., Theoretical Results on Reinforcement Learning with Temporally Abstract Behaviors, In Proceedings of the Tenth European Conference on Machine Learning ECML 98, pp 382-393, 1998.

[Parr 1998] Parr, R., Hierarchical Control and Learning for Markov Decision Processes. Doctoral dissertation, University of California, Berkeley, CA, 1998.

[Precup 2000] D. Precup, Temporal Abstraction in Reinforcement Learning. Ph.D. thesis, University of Massachusetts, Amherst, MA, 2000.

[Parr and Russell 1997] Parr R. and Russell S., Reinforcement Learning with Hierarchies of Machines, In Proceedings of Advances in Neural Information Processing Systems 10, pp 361-368, 1997.

[Ravindran and Barto 2002] Ravindran, B., and Barto, A. G., "Model minimization in hierarchical reinforcement learning". In the Proc. of SARA, LNCS (2371), pp 196-211, 2002.

[Ravindran and Barto 2003] Ravindran, B., and Barto, A., "SMDP Homomorphisms: An Algebraic Approach to Abstraction in Semi-Markov Decision Processes", In the Proc of IJCAI, pp 1011-1016, 2003.

[Sacerdoti 1974] Sacerdoti E.D., Planning in a Hierarchy of Abstraction Spaces, Artificial Intelligence, Volume 5, pp 115-135, 1974.

[Singh 1991] Singh S. P., Transfer of Learning Across Compositions of Sequential Tasks, In Brinbaum L.A., and Collins G.C., editors, Proceedings of the Eighth International Workshop on Machine Learning , pp 348-352, 1991.

[Singh 1992b] Singh S. P., Reinforcement Learning with a hierarchy of Abstract Models, In Proceedings of the Tenth National Conference on Artificial Intelligence, pp 202-207, 1992b.

[Singh 1992a] Singh S. P., The Efficient Learning of Multiple Task Sequences, In Moody J., Hanson S, Lippman R., Editors, Advances in Neural Information Processing Systems: Proceedings of the 1991 Conference, pp 251-258,1992a.

[Singh 1992c] Singh S. P., Transfer of Learning by Composing Solutions for Elemental Sequential Tasks, Machine Learning, Volume 8, pp 323-339, 1992c.

[Singh 1994] Singh S. P., Learning to Solve Markovian Decision Processes, Ph.D. Thesis, University of Massachusetts, Amherst,1994.

[Sutton 1988] Sutton R.S., Learning to Predict by the Methods of Temporal Differences. Machine Learning, Vol(3), pp 9-44, 1988.

[Sutton and Barto 1998] R.S. Sutton, A.G. Barto, “Reinforcement Learning: An Introduction”, MIT Press, 1998.

[Sutton et al. 1999] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence, 112, pp 181-211, 1999.

[Sutton et al. 1998] R. S. Sutton, D. Precup, and S. Singh. Between Mdps and Semi-MDPs:Learning, Planning and Represewnting Knowledgee at Multiple Temporal Scales, Technical Report, pp 74-98, University of Massachusetts, Amherst, 1998.

[Thrun 1992] Thrun S.B., The role of exploration in learning control. In White, D. A., & Sofge, D. A. (Eds.), Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches. Van Nostrand Reinhold, New York, NY, 1992.

[Thrun and Schwartz 1995] Thrun S. B., and Schwartz A., Finding structure in reinforcement learning. In G. Tesauro, D. S. Touretzky, and T. Leen, editors, Advances in Neural Information Processing Systems: Proceedings of the 1994 Conference, pages 385-392, Cambridge, MA, 1995.

[Theocharous and Mahadevan 2002] Theocharous G., and Mahadevan S., Approximate Planning with Partially Observable Markov Decision Processes for Robot Navigation, In Proceedings of the 2002 IEEE Conference on Robotics and Automation, pp 1347-1352, 2002.

[Watkins 1989] Watkins, C. J. C. H., "Learning from delayed rewards", Ph.D. thesis, Psychology Department, University of Cambridge, 1989.

[Watkins and Dayan 1992] C. J. C. H. Watkins and P. Dayan. Q-learning. Machine Learning, 8, pp 279–292, 1992.

[Wolfe and Barto 2006] Wolfe, A. P., and Barto, A.G., "Defining Object Types and Options Using MDP Homomorphisms", In Proc. of ICML Workshop on Structural Knowledge Transfer for ML, 2006.

BIOGRAPHICAL INFORMATION

Srividhya Rajendran received her B.E. in Computer Engineering from B.V.M. Engineering College, S.P University, Anand, India in 1998. She earned her Masters in Computer Science and Engineering from the University of Texas at Arlington in 2003. Her masters thesis title was “Developing Focus of Attention Strategies using Reinforcement Learning”. She received her Ph.D. in Computer Science from the University of Texas at Arlington in August 2009. Her research interests are in the areas of Artificial Intelligence, Machine Learning, Data Mining and Robotics.