



Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019

Subgoal Discovery for Hierarchical Reinforcement Learning Using Learned Policies

Sandeep Kumar Goel
goel@cse.uta.edu

Technical Report CSE-2003-31

This report was also submitted as an M.S. thesis.

SUBGOAL DISCOVERY FOR HIERARCHICAL
REINFORCEMENT LEARNING USING
LEARNED POLICIES

The members of the Committee approve the master's
thesis of Sandeep Kumar Goel

Dr. Manfred Huber
Supervising Professor

Dr. Farhad Kamangar

Dr. Lynn Peterson

Copyright © by Sandeep Kumar Goel 2003

All Rights Reserved

SUBGOAL DISCOVERY FOR HIERARCHICAL
REINFORCEMENT LEARNING USING
LEARNED POLICIES

by

SANDEEP KUMAR GOEL

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

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

Dec 2003

ACKNOWLEDGEMENTS

Dr. Manfred Huber has been a great thesis advisor. I thank him for giving me an opportunity to work for him and for all the calm explanations. He helped me by shaping my critical thinking as well as by improving my expressive skills. I would like to thank Dr. Farhad Kamangar and Dr. Lynn Peterson for serving on my committee.

I would like to thank Dr. Zaruba, Dr. Alp and Mr. David Levine for everything that they have taught me.

I extend my appreciation to my friends—Raman, Vivek, Shashank, Amit, Birj, Gaurav and the rest—for their support and humor through all the conflicts of these past two years.

Finally, I would also like to thank my parents and sister for their endless love and constant support.

September 15, 2003

ABSTRACT

SUBGOAL DISCOVERY FOR HIERARCHICAL REINFORCEMENT LEARNING USING LEARNED POLICIES

Publication No. _____

Sandeep Kumar Goel, MS

The University of Texas at Arlington, 2003

Supervising Professor: Dr. Manfred Huber

Reinforcement learning has proven to be an effective method for creating intelligent agents in a wide range of applications. However, it suffers from the need for a large number of training episodes, a problem that is especially noticeable in large domains. Although the utility of hierarchy is commonly accepted, there has been relatively little research on autonomously discovering or creating useful hierarchies. A system is desirable that can scale reinforcement learning to complex real-world tasks and autonomously discover hierarchical structures within their learning and control systems.

This thesis introduces a method that allows a reinforcement learning agent to autonomously discover and create hierarchy from a learned policy model. A hierarchy of actions helps to create an abstraction which is an encapsulation of a set of actions into a single higher level action that allows an agent to learn while ignoring details that appear at finer levels. The main idea is to find subgoals in a learned policy model by searching for states that exhibit certain structural properties. These subgoals are used to create hierarchies of actions. The hierarchies of actions help the agent to explore more effectively and accelerate learning in other tasks in the same or similar environments where the same subgoals are useful. It is demonstrated that the hierarchical action sequences created with autonomously discovered subgoals can facilitate learning and enable effective knowledge transfer to related tasks.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iv
ABSTRACT	v
LIST OF FIGURES	ix
LIST OF TABLES.....	x
Chapter	
1. INTRODUCTION	1
1.1 Contribution of this thesis.....	3
2. RELATED WORK	6
2.1 Relevant AI Literature	6
2.2 Relevant RL Literature.....	8
3. LEARNING FRAMEWORK.....	14
3.1 Reinforcement Learning	14
3.2 Q-learning	18
3.3 Hierarchical Framework.....	20
4. FINDING USEFUL SUBGOALS	22
4.1 Autonomous Subgoal Discovery.....	22
4.1.1. Empirical Threshold	26
4.1.2. Randomly Connected Worlds	28

5. EXPERIMENTAL RESULTS	31
5.1 Subgoal Discovery.....	31
5.2 Hierarchical Policy Formation.....	36
6. CONCLUSIONS & FUTURE WORK	42
REFERENCES	44
BIOGRAPHICAL INFORMATION.....	51

LIST OF FIGURES

Figure	Page
3.1 The reinforcement learning framework.....	15
3.2 Hierarchical Framework.....	21
4.1 Histogram for the distribution of the gradient ratio in regular space (dark bars) and at subgoal states (light bars).	28
4.2 Histogram for the distribution of the gradient ratio in regular space (dark bars) and at subgoal states (light bars).	29
5.1 Grid-World.....	31
5.2 Action Space.....	32
5.3 Grid-World with the initial state in the upper left corner, the goal in the lower right portion and a random path under the learned policy.....	33
5.4 Count curve along a randomly chosen path through A subgoal state under the learned policy.....	34
5.5 Subgoal states discovered by the agent.....	35
5.6 New task with goal state in the left hand room.....	37
5.7 Comparison of learning speed using subgoal policies And using primitive action only.....	38
5.8 Comparison of learning speed using subgoal policies And using primitive action only.....	39
5.9 Comparison of learning speed using subgoal policies and using primitive actions only averaged over 10 different worlds	41

LIST OF TABLES

Table	Page
3.1 Q-learning algorithm.....	20

CHAPTER I

INTRODUCTION

Humans are effective problem solvers because they can systematically ignore irrelevant details in complex environments. For example while playing tennis we don't worry about planning the movements of our individual muscle fibers. If we consciously had to plan our muscle movements every time we wanted to play a stroke in tennis it would probably take much more time and an entire game of tennis would be nearly impossible. Instead, we create a higher level action that encapsulates the entire set of muscle movements that we use to play a stroke in tennis. Using hierarchical actions we can plan at a higher level that allows us to ignore details at the muscle level and to focus on larger tasks such as playing a game of tennis. A compact representation of the knowledge gained in one task that allows the knowledge to be re-used can facilitate solving complex tasks. This representation can enable an agent to learn and plan at a higher level.

Researchers in Artificial Intelligence (AI) have long studied the use of hierarchy to enable AI systems to solve large and complex problems. By allowing a system to ignore details that are not immediately relevant to the current task, the size of the space that an agent must search to find a solution can be reduced.

One approach is creating a hierarchy of actions. Hierarchy of actions encapsulates a complex set of actions into a single higher-level action that allows an agent to learn and plan at multiple scales in time. As an example, the “play tennis stroke” encapsulation discussed above is a hierarchy of actions because a human can plan to play tennis using this hierarchical action without the need to know how many muscles will be needed to swing the tennis racket and hit the ball, i.e. without the need to plan the muscle movements consciously. Another approach reduces the size of the state space by dropping irrelevant state variables. This approach generalizes or aggregates over state variables. Using the previous example, if the muscle movements required to play a tennis stroke can also be applied to play a squash stroke, then the agent can generalize across the “tennis” and “squash” variables to create an abstraction of “play a stroke.”

Hierarchy is useful for facilitating the control of complex systems. By allowing an agent to plan with hierarchical actions the effective depth that the agent must search to find a solution can be shortened. Also, by encapsulating a set of complex actions into a single higher-level action, such as the “play a stroke” example discussed above, a hierarchical structure can allow a more effective transfer of knowledge from one task to another. By doing so, the higher level actions should enable an agent to solve more difficult tasks as long as the tasks have some underlying characteristics in common.

The method presented in this document is illustrated in the context of machine learning. Machine learning techniques provide an agent with the ability to adapt to changes in the environment as well as the ability to improve performance by learning

from experience. Machine learning is helpful in tasks where the pre-programming of the agents is very difficult, yet the agents must effectively perform in a complex and changing environment. In particular, the focus here is on the machine learning approach known as reinforcement learning (Sutton and Barto, 1998). Unlike supervised learning approaches, which require an outside teacher to specify the correct actions to take at each step, a reinforcement learning agent learns directly from its interactions with its environment. Reinforcement learning has been demonstrated to be especially effective for control problems where the goal objective is known but the method to achieve the goal is not specified.

While learning from reinforcements has proven to be an effective and popular method for creating intelligent agents, a key scaling problem of reinforcement learning is that in large domains an enormous number of decisions are to be made. For many tasks, the algorithms can take impractical amounts of time to achieve an optimal level of performance. This limits the size of feasible problems for which a reinforcement learning system can approximate solutions. By studying techniques to discover subgoal automatically and create hierarchy, this thesis addresses the issue of the limitations on the size of feasible problems.

1.1. Contribution of this Thesis

This thesis introduces a novel method that allows an agent to autonomously discover subgoals and create a hierarchy from a learned policy model. Although much of the relevant research in AI has demonstrated the utility of hierarchy, there is comparatively little work on how to discover or create useful hierarchical structure. On

the other hand, it is highly desirable, in particular for a reinforcement learning agent to discover the subgoals and create hierarchies automatically.

The main idea presented in this thesis is to search in a learned policy model for states that exhibit certain structural properties. In particular, the method searches for “subgoals”. Conceptually, states that connect separated strongly connected spaces are categorized as subgoals. The subgoal will be defined formally in chapter IV. The method detects useful subgoals in the learned policy model and creates policies that achieve these subgoals. These policies are added to the action space of the agent, which means that the agent has a mapping from states in the environment to the actions that should be chosen to achieve the subgoal. These subgoals are then used to create hierarchies. The hierarchical actions help the agent to explore more effectively and accelerate learning in other tasks in the same or similar environments where the same subgoals are useful. This document presents results where our methods used to discover subgoals and create a hierarchy on a grid – world navigation task.

Before describing the methods for discovering subgoals in more detail, related research is presented in Chapter 2 within the context of the work in this thesis. In Chapter 3 an overview of reinforcement learning, Q-learning, and the hierarchical framework is given. Chapter 4 presents our technique for automatically discovering subgoals. After presenting the method, we discuss experimental results in a grid-world task in Chapter 5. These results make use of a reinforcement learning agent to generate the behavior used to detect the subgoals. Chapter 5 also explains hierarchy formation in

detail along with experimental results. The final chapter, Chapter 6, concludes and discusses plans for future research in this area.

CHAPTER II

RELATED WORK

This chapter discusses related research in the AI and RL (Reinforcement Learning) domains and puts it in the context of the approach presented in this thesis.

2.1 Relevant AI Literature

Much of the early research on hierarchical structures came from joint psychology and AI research where computational systems were built to further the understanding of human problem solving. Newell et al.'s (1963) General Problem Solver and the SOAR project (Laird et al., 1986), both use the concept of hierarchy to narrow the search space of the problem solver. By creating a hierarchy of sub-problems that are more easily solved, the path to a solution for the main problem is shortened which enables problem solving to work more effectively. SOAR formed its hierarchy by grouping together sequences of actions used to solve subproblems. Although this method helped the system to solve larger problems, SOAR sometimes suffered from the “utility problem” (Minton 1988) where too many chunks could actually slow down the system by increasing the branching factor of the search. The work of (Anzai and Simon 1979) also fits into this paradigm. These researchers analyzed human problem solving protocols to guide them in creating a general problem solving mechanism that can create subgoals and group together useful sequences of actions.

(Amarel 1968) discussed the Missionaries and Cannibals problem to state the need for macro operators or actions in problem solving. Amarel presented a series of different hand-crafted macro operators and demonstrated that the use of certain hierarchical structures could significantly reduce the search space of the problem and thus make the problem much easier to solve. He also discussed how a macro operator that takes an agent to “narrows” in a search space can be useful for problem solving. The narrows connect what Amarel terms “easily traversable areas.” Having a policy to reach the critical region, i.e. narrows regions, can significantly accelerate the search for a solution by enabling an agent to move to these important regions more easily.

Other early research on hierarchical structures in AI was focused on planning systems. For example, the STRIPS planner (Fikes et al., 1972) was one of the planning systems to make use of hierarchical structures. It used pre-defined hierarchies to facilitate planning. Later systems focused on approaches for generating planning hierarchies automatically. For example, the ABSTRIPS system (Sacerdoti, 1974) introduced an extension of STRIPS that could automate some of the generation of planning hierarchies. (Prieditis 1993) introduced a system called Absolver II which automatically found admissible heuristics for use in search systems by using a set of pre-defined allowable transformations on the STRIPS goal and operator space. By generalizing the original problem formulation, Absolver II was able to significantly decrease the time needed to find a solution. Knoblock’s (1990, 1991) planning system, ALPINE, automatically generated planning hierarchies by dropping literals from the goal description in an ordered manner to create a more abstract space in which to solve

the problem. (Korf 1985) system could automatically create open-loop macro operators for use in a means-ends problem solver. These macros were designed to abstract over subgoals whose solutions violated a part of the overall problem that was already solved. He used the Rubik's cube as an example, where a solution path for the whole cube must contain states where earlier goals such as "solve one color" are temporarily violated. By abstracting over the non-serializable part of a process, a means-ends problem solver can solve problems that were previously intractable. A drawback of Korf's method is that it generates all possible sequences before pruning the set to those that allow an agent to abstract over non-serializable subgoals.

The work discussed above on generating hierarchical structures automatically while planning is related to the research presented in this thesis mainly through the common focus on automatically generating useful hierarchies. In both domains, this means creating hierarchical structures that enable a system to be used in solving more difficult problems. But none of these systems presents an approach of discovering subgoals in order to facilitate the addition of higher level actions to the available action set.

2.2 Relevant RL Literature

Hierarchy is a key way of dealing with large state spaces and work has been underway in the reinforcement learning field to determine how to best use it. Hierarchical decomposition of reinforcement learning problems has been used to accelerate learning in large-scale MDP problems by exploiting structure in the problem

domain. The components of the hierarchy typically consist of related subtasks with varying degrees of specialization, and independent policies to achieve those subtasks.

The use of hierarchical structures in Reinforcement learning (RL) has been studied by several researchers. Much of the theoretical foundation for the use of hierarchy of actions in RL has been provided by Precup and Sutton (1997), Parr and Russell (1997), Precup et al. (1998), Parr (1998) and Sutton et al. (1999). These researchers have shown that the use of hierarchical actions transforms a Markov Decision Process (MDP) into a Semi-Markov Decision Process (SMDP) and that convergence results still hold for the learning algorithms known to converge in the absence of hierarchy.

Related theoretical work by Hauskrecht et al. (1998) involved the use of abstracting the action space locally to generate smaller and more abstract MDPs. They showed that the new MDPs could be solved more quickly than the original MDPs and that abstraction helped to facilitate knowledge transfer across tasks. Their work built on the theoretical foundations described above and provided some theoretical reasons as to why hierarchy of actions can be useful to an RL system.

One of the advantages to using hierarchical structures in RL is that the hierarchy can enable a system to learn to solve more complex problems. A traditional solution to the problem of scaling RL systems up to larger problems involved the use of shaping, or training the system on a series of related subproblems and then using the solutions to the subproblems to facilitate a solution for the next harder problem (Gullapalli, 1992; Singh, 1991, 1992a,c). These systems showed that training could successfully be used

by an RL system to approximate solutions in cases where a solution would have been very difficult to find otherwise. The main limitation to these ideas is that the training sequence must be defined a priori by the programmer.

Singh's (1992b, 1994) H-DYNA algorithm is an example of using hierarchical structures to facilitate learning in an RL framework. Other work includes Bradtke and Duff's (1995) SMDP learning algorithm, which learns action values for variable duration actions by backing up the discounted sum of the reward received while the action was executing. Macro Q-learning (McGovern et al., 1997) 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. Many of the RL algorithms that use abstraction create hierarchies of states or actions. The systems of Kaelbling (1993) and Dayan and Hinton (1993) are two early examples. Both of these methods create a hierarchy of value functions based on attributes of the state space and are able to accelerate learning compared to a flat, or non-hierarchical, system. This work was extended by Moore et al. (1999) to automatically generate hierarchies in goal directed systems. But again none of these systems focused on discovering subgoals in the state space to facilitate hierarchy. The work of Theodorou and Mahadevan (2002) also fits into this paradigm by demonstrating that hierarchical Partially Observable MDPs (POMDPs) can enable a robot to successfully solve more difficult tasks than are possible without a hierarchy. Ryan and Pendrith's (1998) RL-TOPs generates hierarchies in a partially automatic manner. Their work is distinguished by the use of a traditional planning paradigm where all actions must have specified pre- and post-

conditions. But none of these systems adds new actions to the available action set. Rather, the systems focus on restricting the search space of the RL agent to decrease the time needed to approximate an optimal solution. The MAXQ algorithm by Dietterich (1998) provides an alternate framework for finding hierarchical structures in RL. His system uses a fixed hierarchy, where each action at each level of the hierarchy has a separate value function. Each level of the hierarchy also has a value function to control the actions for that level. Makar et al. (2001) have extended the MAXQ framework to multiple agents and further demonstrated the use of hierarchies for complex learning tasks.

Another approach in RL is that of Drummond (1998). He proposed a system in which an RL agent could detect both walls and doorways through the use of techniques applied to the learned value function. This enabled the agent to re-use parts of the value function for task transfer. Although his method can identify doorways and walls in a two dimensional gridworld setting, his approach was aimed at detecting sharp changes in the value function and not at identifying subgoals.

Thrun and Schwartz's (1995) SKILLS algorithm examine optimal policies within the set of related MDPs. SKILLS can extract a pre-specified number of action sequences that are common across the tasks. These action sequences can be useful in other tasks that share this state space. A similar system by Bernstein (1999) uses the optimal policies for a given set of tasks to generate a single new hierarchy that he calls a "reuse option." An option is a higher level action that consists of a sequence of primitive actions. To do this, his method examines the probabilities of taking each

action in each state for a given optimal policy. The action distribution for each state in the reuse option is then formed by averaging the action probabilities from each of the optimal policies for that state. Both of these systems suffer from their requirement to completely solve a set of related MDPs. This limits the size of the problems to those that are small enough to approximately solve in a reasonable amount of time. Both of these systems can enable transfer of knowledge among similar MDPs as well as accelerate learning on new tasks. However, neither of these goals can be accomplished using these systems unless the set of tasks share the same state set.

Another system that can automatically discover hierarchy in an RL framework is Digney's (1996, 1998) Nested Q-learning algorithm, it creates new options online while an agent is learning using Q-learning. This system creates new options by examining two criteria: frequency of state visitations and the reward gradient at each state. States that are visited frequently or states where the reward gradient is high are chosen as subgoals for new options. Because newer actions can call existing actions as subroutines, Nested Q-learning creates a hierarchy of actions for the agent.

McGovern's (McGovern and Barto, 2001a) work is most closely related to the work presented in this thesis. McGovern's method uses diverse density to discover useful subgoals automatically by keeping track of successful and unsuccessful events. However, in the case of more complicated environments and rewards it can be difficult to accumulate and classify the sets of successful and unsuccessful trajectories needed to compute the density measure or frequency counts. In addition, these methods do not allow the agent to discover subgoals that are not explicitly part of the tasks used in the

process of discovering them. In this thesis the focus is on discovering subgoals by searching a learned policy model for certain structural properties. This method is able to discover subgoals even if they are not a part of the successful trajectories of the policy. If the agent can discover these subgoal states and learn policies to reach them, it can include these policies as actions and use them for effective exploration as well as to accelerate learning in other tasks in which the same subgoals are useful.

CHAPTER III

LEARNING FRAMEWORK

3.1 Reinforcement Learning

Machine learning techniques provide an agent with the ability to adapt to changes in the environment as well as the ability to improve performance by learning from experience. This ability is an important component for systems that need to interact with real world. Machine learning algorithms typically fall into three categories—supervised, unsupervised and reinforcement. Supervised learning involves learning from a teacher—the agent is provided with sets of inputs and the desired outputs. In terms of states and actions, the agent is taught the action it should take in each state. Unsupervised learning involves the grouping of instances based only on the information in the input. Reinforcement learning involves an agent learning through its interaction with the environment and a simple, scalar reward signal rather than from a teacher that specifies the action the agent should take in each state. In this sense, reinforcement learning may be considered an interactive, temporally extended problem solver. Kaelbling, Littman & Moore (1996) provide a detailed survey of the field of reinforcement learning. Reinforcement learning has been demonstrated to be especially effective for control problems where the goal objective is known but the method to achieve the goal is not specified.

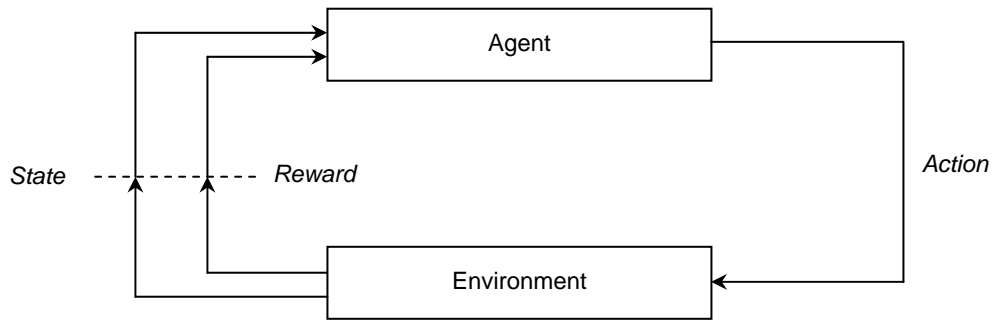


Figure 3.1 The reinforcement learning framework.

Figure 3.1 depicts the reinforcement learning framework for an agent that interacts with the environment at discrete time steps. At each time step, the agent is in some state in the environment, and selects an action for execution accordingly. It executes the action in the duration between this time step and the next. As a consequence of this action, the agent receives a numeric reward and moves into the next (and potentially different) state.

For the agent to take the best action and correctly predict its immediate reward, the state signals received from the environment must not only include the immediate sensations, but must also summarize all information contained in past sensations that is necessary to predict future outcomes. Which of the past sensations have to be retained for the state representation to be sufficient depends on the nature of the task itself. Such a state signal is said to be *Markov* or to have the *Markov property*. Formally, the Markov property implies that:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) \quad (3.1)$$

$P(s_{t+1} | s_t, a_t)$ is the probability of transitioning to state s_{t+1} from state s_t under the action a_t . A system that satisfies the Markov property is called a *Markov decision process*, or *MDP*. A *Markov decision problem* is an MDP together with a performance criterion, which assigns a total cost to each state relative to a specific policy. Singh's (1993) and Sutton & Barto's (1998) work is concerned with finding an optimal total-cost function for an infinite horizon discounted MDP. In an infinite horizon discounted MDP, the value of each state specifies the expected future reward for the task, and is defined as:

$$\begin{aligned}
 V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\
 &= \sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau}
 \end{aligned} \tag{3.2}$$

V is the value function, r_t is the reward at step t and γ is the discount rate parameter. In the infinite-horizon case, the agent must look ahead an infinite number of time steps or actions in determining the total-cost function. The agent is made to discount the contribution of actions that are not immediate to a given state in order to decrease its contribution to the total cost. In practice, a discounted cumulative reward is computed, by which future actions are discounted in their credibility; an action that yields a reward of r and is taken after n steps would contribute a discounted term of $\gamma^n r$ to the total future reward, where γ is the discount rate parameter. If γ is one, we look ahead to an infinite horizon in the computation of the future reward. If γ is zero, we look ahead to only the immediate reward obtained by the next action – a policy obtained in such a manner is considered as greedy.

There are different approaches to solving Markov decision problems. Linear programming can be used to solve certain Markov decision problems in exponential. Linear programs are typically solved using Dantzig's simplex method (Sutton and Barto 1998) or its variations. However, it has been shown that the worst-case running times of these methods are of exponential complexity. This may be attributed to the fact that linear programming is a very general problem solving approach and does not take advantage of the formulation of Markov decision problems. Better performance can be expected from MDP-specific algorithms such as policy iteration and value iteration, which fall under the category of dynamic programming. The solution obtained by value iteration typically improves as the running time increases, until the optimal value function is computed; it expands the look-ahead horizon of the model, starting with a one-stage finite horizon and increasing the horizon as the algorithm progresses. An automatic stopping rule can be specified according to the optimality of the obtained policy. One such termination criterion is the *Bellman residual* (Williams & Baird, 1993), which is used in this work to analyze the progress of the value iteration-based learning approaches. The *Bellman residual* approximates the *optimal value function*, V^* .

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s, a) [R(s'|s, a) + \gamma V^*(s')] \quad (3.3)$$

$P(s'|s, a)$ is the probability of transitioning from state s to state s' under action a and $R(s'|s, a)$ is the expected reward for taking action a in state s and transitioning to state s' . (Littman, Dean & Kaelbling, 1995) give a detailed analysis of the computational complexity involved in the solution of infinite-horizon problems.

3.2 Q-Learning

Reinforcement learning allows an agent to build a policy that allows it to perform a specific task optimally. This mechanism of learning is especially useful in domains which require that the agent interact with its environment in order to receive feedback on its progress towards the solution of the task. The agent takes an action, which leads it to a (potentially) different state and yields a reward that is some measure of how useful the action was in relation to the task at hand.

The learning agent in this thesis work uses an algorithm called Q-learning (Watkins, 1989). The objective of the agent is to learn to maximize the expected value of reward received over time. It does this by learning a (possibly stochastic) mapping from states to actions called a policy, $\Pi: S \rightarrow A$, i.e. a mapping from states $s \in S$ to actions $a \in A$. The criterion used by the agent in selecting the action in every state is the maximization of its future reward. More precisely, the objective is to choose each action so as to maximize the expected return:

$$R = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right] \quad (3.5)$$

Where $\gamma \in [0,1)$ is a discount-rate parameter and r_i refers to the reward at step i . The Q-function, $Q: S \times A \rightarrow \mathfrak{R}$ maps states $s \in S$ and actions $a \in A$ to scalar values. In particular, $Q(s, a)$ represents the expected discounted sum of future rewards if action a is taken in state s and the optimal policy is followed afterwards. Hence Q ,

once learned, allows the learner to maximize R by picking actions greedily with respect to Q:

$$\Pi(s) = \arg \max_{a \in A} Q(s, a) \quad (3.6)$$

The action value function Q is learned on-line through experimentation. Suppose that during learning the learner executes action a in state s , which leads to a new state s' and the immediate pay-off r_t . In this case Q-learning uses this state transition to update Q (s, a) according to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a)] \quad (3.7)$$

This update is referred to as one-step Q-learning. The reward r_{t+1} is that which is obtained on the completion of action a_t in state s_t . The scalar $\alpha \in [0,1]$ is the learning rate.

A learning algorithm such as Q-learning has the advantage of being able to select actions according to a control policy other than the policy being learned. Typically, a policy that is derived from Q is used. Such a policy might pursue actions according to merits other than measures of task optimality, such as exploration. For instance, a ϵ -greedy strategy derived from Q would be the following policy: a random, exploratory action is selected with a probability of ϵ and the action with the highest Q value among all the actions that lead out of the current state is selected with a probability of $(1-\epsilon)$. Similarly, previously untried actions may be given a higher precedence over other actions.

Table 3.1 Q -learning algorithm

1	Initialize $Q(s, a)$ arbitrarily
2	Repeat (for each episode):
3	Initialize $s \in S$, the initial state for the episode
4	Repeat (for each step of episode):
5	Choose $a \in A$ for s using a policy derived from Q (e.g. ϵ -greedy)
6	Take action a , observe reward r and next state $s' \in S$
7	$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
8	$s \leftarrow s'$
9	Until s is terminal

Table 3.1 shows the incremental one-step Q -learning algorithm. A single episode consists of starting at some initial state, selecting actions according to some policy, and modifying the Q values of the selected actions, until a terminal or accepting state is reached.

The algorithm guarantees the convergence of Q to the optimal Q value function under the following conditions. First, the system must make up a Markov Decision Process (MDP). This is essential in the derivation of equation 3.7, the Q update rule. Second, the action selection mechanism (line 5) must be able to reach each state in the model. Typically, we can ensure this by incorporating the probabilistic exploration of an action despite its not being the best action. Third, the reward function must be bounded; that is, for each state-action pair (s_t, a_t) , we must ensure that $r_t < C$ for some positive constant C .

3.3 Hierarchical Framework

The discovery of subgoals and addition of policies that lead to the subgoals is the way this document introduces hierarchy to reinforcement learning. Hierarchy facilitates the agent's exploration and speed of learning in related tasks in the same or similar environments and even transfers knowledge to the related tasks. Figure 3.2 chronicles the steps in hierarchical policy formation.

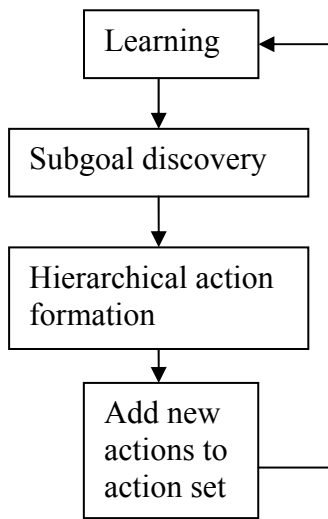


Figure 3.2 Hierarchical Framework

In reference to the work in this document an agent learns a task using Q-learning. Once the agent is finished learning the current task, it searches the policy model for subgoals. These subgoals are then used to learn policies to achieve them. These policies are called abstract actions. These higher level actions are then added to the already existing action set of the agent, which means that the agent has a mapping from states in the environment to the actions that should be chosen to achieve the subgoal. The hierarchical actions help the agent to explore more effectively and accelerate learning.

CHAPTER IV

FINDING USEFUL SUBGOALS

This chapter describes the method by which a reinforcement learning agent can discover subgoals automatically. By discovering subgoals and including policies to subgoals as actions in its action set, the agent can learn strategies for hierarchical decision making. The introduction of hierarchical structure in the learning system helps the agent to accelerate learning in other tasks in the same or similar environments where the same subgoals are useful. Section 4.1 explains how the agent can discover the subgoals by searching a learned policy model. This approach is illustrated using navigation tasks in a grid-world domain. It is important to note here that while this document develops and applies the learning and hierarchical policy formation techniques in the context of grid-world navigation, they are equally applicable in a wide range of other task domains.

4.1 Autonomous Subgoal Discovery

Subgoals provide an efficient means of decomposing problems. An example that shows that subgoals can be useful is a room to room navigation task where the agent should discover the utility of doorways as subgoals. If the agent can recognize that a doorway is a subgoal, then it can learn a policy to reach the doorway. This policy can accelerate learning on related tasks in the same or similar environments by allowing the agent to move between the rooms using single actions. The idea of using subgoals,

however, is not confined to grid-worlds or navigation tasks. Other tasks should also benefit from subgoal discovery. For example, consider a game in which the agent must find a key to open a door before it can proceed. If it can discover that having a key is a useful subgoal, then it will more quickly be able to learn how to advance from level to level (McGovern and Barto, 2001b).

In the approach described in this document, the focus is on discovering useful subgoals that can be defined in the agent's state space. Policies to those subgoals are then learned and added as actions in the agent's action space.

Definition: A *regular space* is a space that is uniformly connected. For example in a room the region near to the walls cannot be considered as regular space since the presence of walls affects the mobility in that region.

In a regular space every state will have approximately the same expected number of direct predecessors under a given policy, except for regions near the goal state or close to boundaries (where the space is not regular).

Definition: A state s is a *direct predecessor* to state s' if under the learned Policy the action in state s leads to s' .

Definition: The *Count metric* for a state s under a learned policy is given by the sum of count metrics of all the direct predecessors of s plus the number of the predecessors s has.

Definition: The *Count Curve* for a "Path" under a given policy is the plot of the Count metric of all the states along that path.

In a regular and unconstrained space, if the predecessor count under a given policy is accumulated for every state and a curve of these counts along any chosen path is plotted, the expected value of the ratio of gradients along this curve would be a positive constant. The reason is that the count metric along any path in a regular space increases according to the number of direct predecessor under the policy and since the expected number of predecessors is the same for any state, the ratio of the increase in the count metrics of two consecutive states would be expected to be constant.

The subgoals are the states that connect separate strongly connected spaces. Under a learned policy such spaces would behave like a funnel and the state at the narrow end of the funnel would represent a subgoal. To identify such states it is possible to evaluate the “Gradient Ratio” of the count curve before and after the subgoal state.

Definition: A “Subgoal” state is a state with the following structural property: the state space trajectories originating from a significantly larger than expected number of states leads to the subgoal state as compared to its successor state. Such states represent a “funnel” for the given policy.

Definition: The Gradient Ratio at any state is the ratio of the slopes of the count curve at the current state and the successor state.

Consider a path under a given policy going through a subgoal state. The predecessors of the subgoal state along this path lie in a relatively unconstrained space, however, the dynamics of the count metric changes strongly at the subgoal state. There will be a strong and sudden increase in the count metric and the curve will become steeper as the path approaches a subgoal state. On the other hand, the increase in the count metric can

be expected to be much lower for the successor state of the subgoal as it again lies in a relatively unconstrained space. Thus the ratio of the gradients at this point will be higher than expected and easily distinguishable. Let $C(s)$ represent the count of predecessors for a state s under a given policy, and $C_t(s)$ is the count of predecessors that can reach s in exactly t steps:

$$C_1(s) = \sum_{s \neq s'} P(s|s', \Pi(s')) \quad (4.1)$$

$$C_{t+1}(s) = \sum_{s \neq s'} P(s|s', \Pi(s')) C_t(s') \quad (4.2)$$

$$C(s) = \sum_{i=1}^n C_i(s) \quad (4.3)$$

Where n is the smallest index such that $C_{n+1} = C_n$ or $n = \text{number of states}$, whichever is smaller. The condition $s \neq s'$ prevents the counting of one step loops. $P(s|s', \Pi(s'))$ is the probability of reaching state s from state s' by taking action $\Pi(s')$ (in a deterministic world the probability is 1 or 0). If there are loops within the policy, then the counts for the states in the loop will become very high. This implies that, if no precautions are taken, the gradient criteria used here might also identify states in the loop as subgoals.

To calculate the ratio along a path under the given policy, let $C(s_1)$ be the predecessor count for the initial state of the path and $C(s_t)$ be the count for the state the

agent will be in after executing t steps from the initial state. The slope of the curve at step t , Δ_t can be computed as:

$$\Delta_t = C(s_t) - C(s_{t-1}) \quad (4.4)$$

To identify subgoals, the gradient ratio $\frac{\Delta_t}{\Delta_{t+1}}$ is computed if $\Delta_t > \Delta_{t+1}$ (If $\Delta_t < \Delta_{t+1}$ then the ratio is less than 1 and the state does not fit the criterion. Avoiding the computation of the ratio for such points thus saves computational effort). If the computed ratio is higher than a specified threshold, state s_t will be considered a potential subgoal.

4.1.1. Empirical Threshold

Assuming that there are a small number of subgoals as compared to the size of the space, the theory of the t-test is applied to compute a threshold. Based on this assumption it can be stated that the distribution of gradient ratios over the entire space represents approximately the distribution in a space free of subgoals. Another assumption made here is that the gradient ratios at any state are randomly drawn from the cumulative distribution. Given the assumption, it can be tested using a t-test whether the gradient ratio distribution at any state belongs to the cumulative distribution and its affirmation means that the state is not a subgoal. To run a t-test for a given state s , the mean, X , of a sample of gradient ratios of size N is computed by randomly choosing N trajectories going through state s under the learned policy. Then equation 4.5 is used to compute the t value. To avoid performing t-test at every state a slightly different approach is taken here. Based on desired p value (the probability of obtaining a

particular sample result given the null hypothesis) sample mean X required to pass the test is computed using formula:

$$t = \frac{X - \mu}{\sigma / \sqrt{N}} \quad (4.5)$$

X is mean of the sample, μ is the mean of whole distribution, σ is the standard deviation of the distribution and N is the sample size. X plays the role of threshold required in this approach because in order for the t-test to provide a positive result for the sample distribution at a given state s , there should be at least one path through s along which the gradient ratio at state s is equal or higher than X . Thus the paths are chosen randomly and it is tested if the gradient ratio at any state along the chosen path is higher than X . A positive answer qualifies the state for a t-test and the result of this test determines if the state is a subgoal or not. The p value can be chosen conservatively in order to avoid subgoals that are not useful.

To show that the gradient ratios in the unconstrained portion of the state space and at a subgoal state are easily distinguishable, histograms for the distribution of these ratios in randomly generated environments are shown in Figure 4.1. Randomly generated environment here means that rooms and goal are created randomly on a 20 x 20 grid where every state is connected through an action to each of its eight neighbors. A detailed explanation is presented in section 5.1.

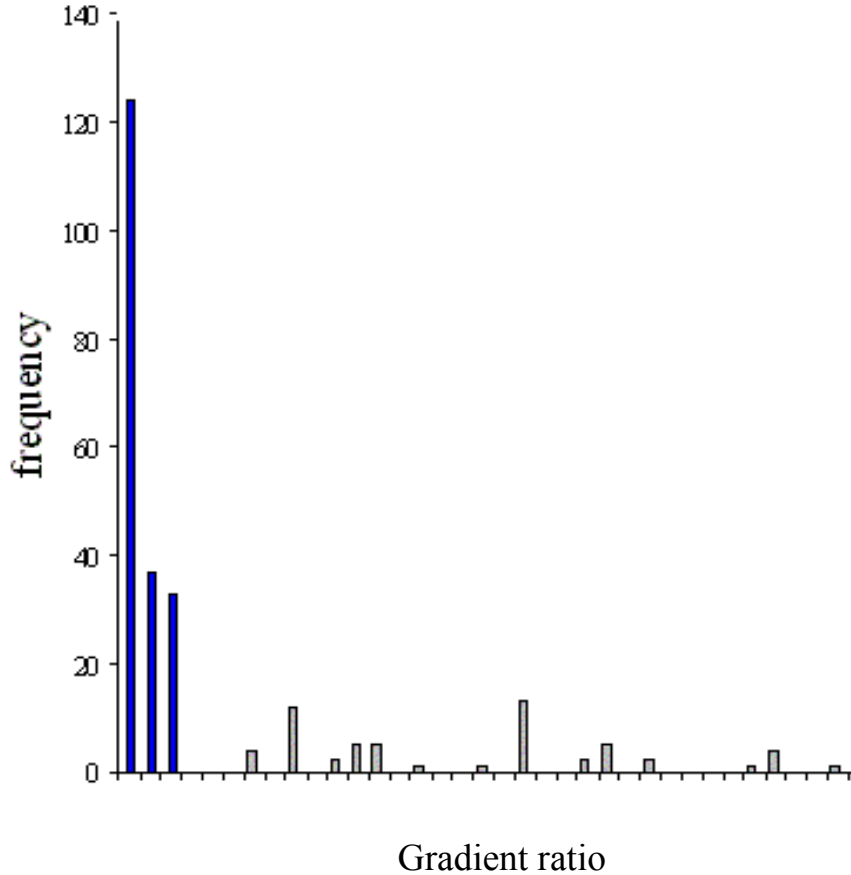


Figure 4.1 Histogram for the distribution of the gradient ratio in regular space (dark bars) and at subgoal states (light bars).

The Histogram shows data collected from 12 randomly generated 20x20 grid-worlds with randomly placed rooms and goals. Each run learns a policy model for the respective task using Q-learning and computes the count metric for every state using equations 4.1, 4.2, and 4.3. Gradient ratios for 40 random paths in each environment are shown in the histogram. Bin size for the histogram is 5.

4.1.2. Randomly Connected World

To show that the method for discovering subgoals discussed above is not confined to grid-worlds or navigation tasks, random worlds were generated for

experiments. Random worlds consist of 1600 states and every state has 10 actions. It is easy to visualize the space as a 40 x 40 grid, but every action in state s connects to a randomly chosen state s' in its local neighborhood. With a 0.5 probability an action connects to its immediate neighbor, with 0.25 probability it connects one level out and with another 0.25 probability it connects to a state two level further. To create a subgoal in such a world by the subgoal definition, pieces of such a random world are generated independently and connected randomly via a small number of states in different pieces.

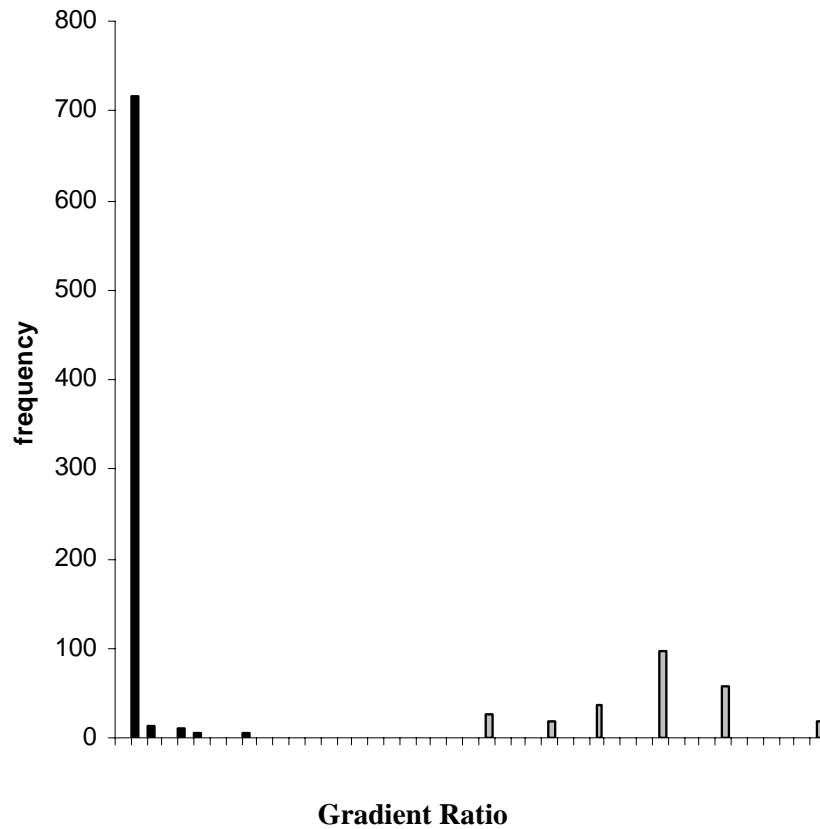


Figure 4.2 Histogram for the distribution of the gradient ratio in regular space (dark bars) and at subgoal states (light bars).

The agent learns a policy model for the respective tasks using Q-learning as explained in chapter III, in each random world. Then the count metric is established and gradient ratios are computed for these spaces with and without a subgoal. Figure 4.2 shows histogram for these gradient ratio distributions averaged over 25 such randomly connected worlds. The results showed that the gradient ratios in the unconstrained portion of the state space and at a subgoal state are again easily distinguishable

CHAPTER V
EXPERIMENTAL RESULTS

5.1 Subgoal Discovery

The subgoal extraction technique presented here has been illustrated using a simple gridworld navigation problem. Figure 5.1 shows a four-room example environment on a 20x20 grid. The grid has been divided into four rooms; bold black lines show walls while the breaks between otherwise continuous lines show the doorway for respective rooms.

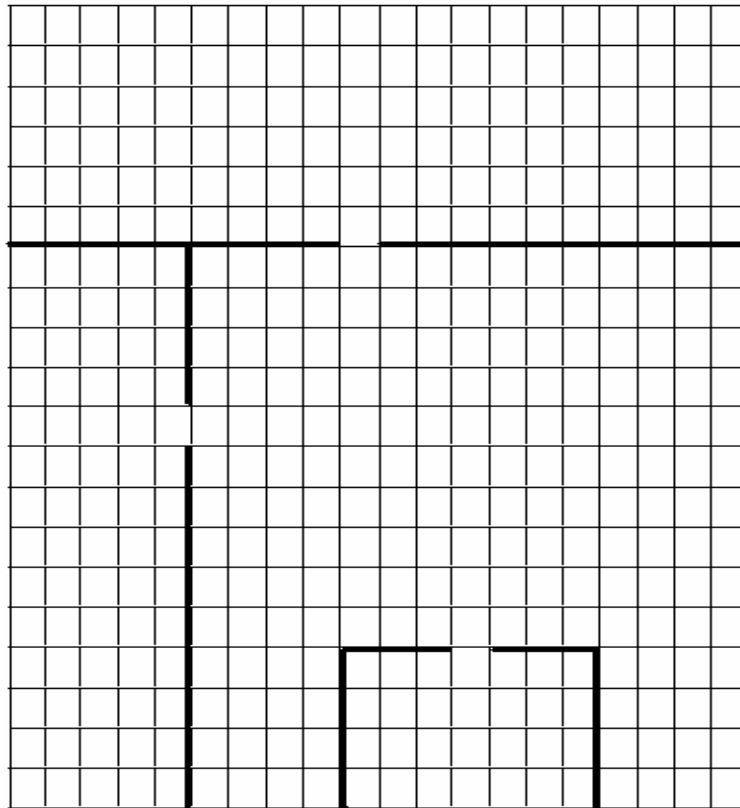


Figure 5.1 Grid-World

The action space shown below in the figure 4.2 consists of eight primitive actions (*North, East, West, South, North-west, North-east, South-west, and South-east*).

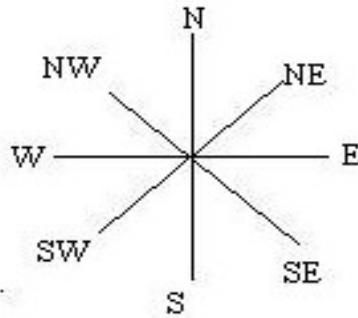


Figure 5.2 Action Space

For the grid-world navigation task, actions and rewards have been assumed to be purely deterministic. In other words, each action succeeds in moving the agent in the chosen direction with probability 1. The world is deterministic and each action succeeds in moving the agent in the chosen direction. With every action the agent receives a reward of -1 for a straight action i.e. (North, East, West, and South) and -1.2 for a diagonal action i.e. (North-west, North-east, South-west, and South-east). In addition, the agent gets a reward of +10 when it reaches the goal state.

The agent learns using the Q-learning algorithm shown in section 3.2 and ϵ -greedy exploration for learning. It starts with $\epsilon=0.90$ (which means 90% of the time it tries to explore by choosing a random action) and gradually decreases the exploration to $\epsilon=0.05$.

The agent then evaluates the ratio of gradients along the count curve by choosing random paths, and picks the states as subgoal state based on empirical threshold as discussed in section 4.1.1.

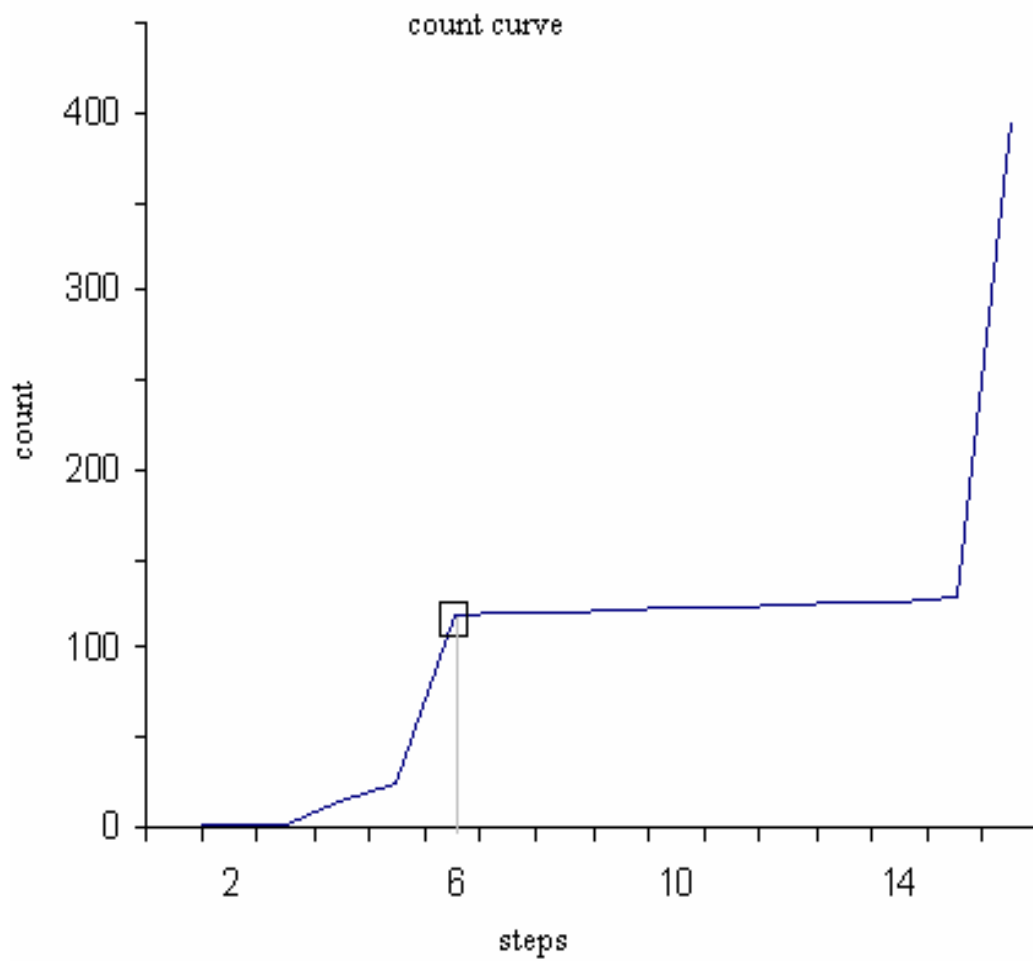


Figure 5.4 Count curve along a randomly chosen path through a subgoal state under the learned policy

For this experiment the count curve along one of the randomly chosen paths through a subgoal state is shown in Figure 5.4. The path chosen is indicated in Figure 5.3 and the subgoal state is highlighted both in Figure 5.3 and Figure 5.4. The value for the gradient ratio at step 4 (which is in regular space) is 1.444 while it is 95.0 at step 6

(which is a subgoal state). The mean of the distribution of gradient ratios over the space is 3.265 and the standard deviation is 13.08. For a p value less than 0.025 and N being 5, the computed threshold is 25. It is evident from the figure 4.1 that this threshold computed by the algorithm is safe enough to be used to pick a subgoal.

The subgoal states that the agent discovered in this experiment are shown in Figure 5.5. The subgoal state leading to the left room is identified here due to its structural properties under the policy and despite the fact that it does not lie on the successful paths between the start and the goal state. The agent did not discover the doorway in the smaller room as a subgoal state because the number of states for which the policy leads through the subgoal is small compared to the other rooms and hence the count for this subgoal state is not influenced significantly by the structural property of the state.

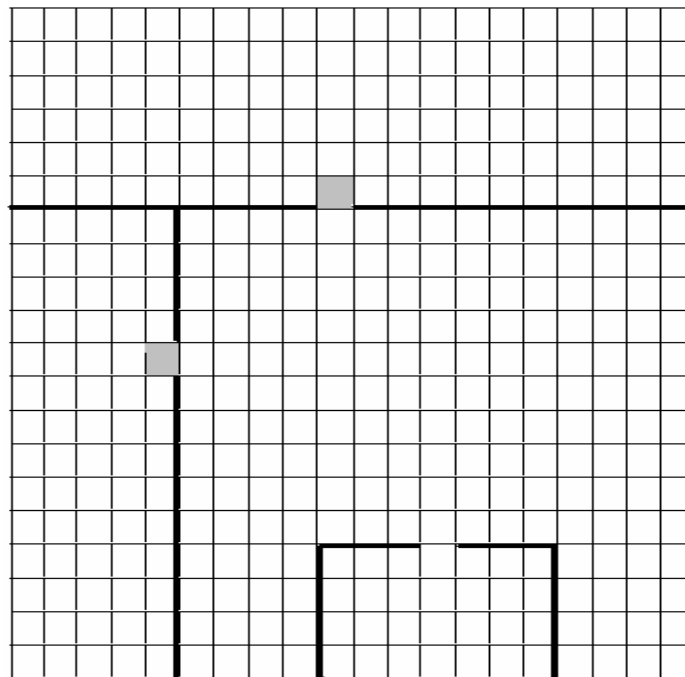


Figure 5.5 Subgoals states discovered by the agent (light gray states)

5.2 Hierarchical Policy Formation

The motivation for discovering subgoals is the effect that available policies that lead to subgoals have on the agent's exploration and speed of learning in related tasks in the same or similar environments. If the agent randomly selects exploratory primitive actions it is likely to remain within the more strongly connected regions of the state space. A policy for achieving a subgoal region, on the other hand, will tend to connect separate strongly connected areas. For example, in a room-to-room navigation task, navigation using primitive movement commands produces relatively strongly connected dynamics within each room but not between rooms. A doorway links two strongly connected regions. By adding a policy to reach a doorway subgoal the rooms become more closely connected. This allows the agent to more uniformly explore its environment. It has been shown that the effect on exploration is one of two main reasons that extended actions can be able to dramatically affect learning (McGovern, 1998).

To take advantage of the subgoal states, the agent uses Q-learning to learn a policy to each of the subgoals discovered in the previous step. These policies, which lead to respective subgoal states (subgoal policies) are added to the action set of the agent.

One reason that it is important for the learning agent to be able to detect subgoal states is the effect of subgoal policies on the rate of convergence to a solution. If the subgoals are useful, then learning should be accelerated. To ascertain that these subgoals help the agent to improve its policy more quickly, two experiments were

performed where the agent learned a new task with and without the subgoal policies. The same 20x20 grid-world with three rooms was used to illustrate the results. Subgoal policies were included in the action set of the agent (*Subg1*, *Subg2*). The task was changed by moving the goal to the left hand room as shown in Figure 5.6. The agent solves the new task using Q-learning with an exploration of 5%.

The action sequence under the policy learned for the new task when its action set included the subgoal policies is (*Subg2*, *South-west*, *South*, *South*, *South*, *South*) where *Subg2* refers to the subgoal policy which leads to the state as shown in Figure 5.6.

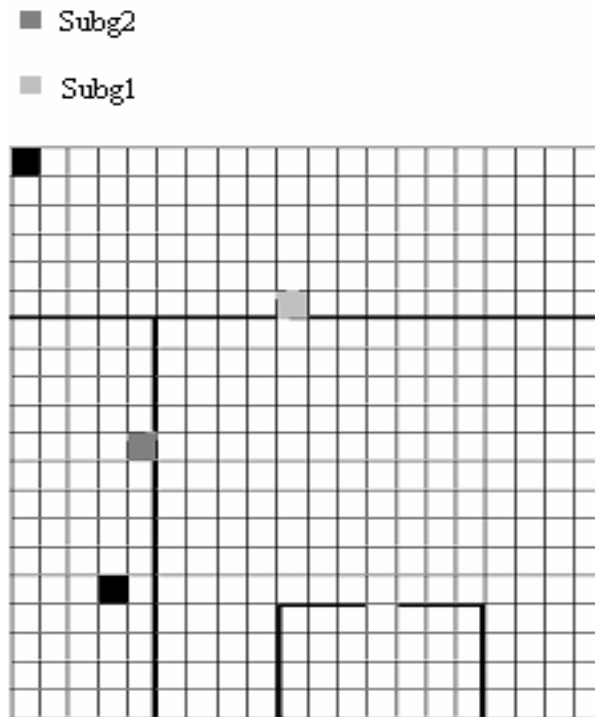


Figure 5.6. New task with goal state in the left hand room

Figure 5.7 shows the learning curves when the agent was using the subgoal policies and when it was using only primitive actions. The learning performance is compared in terms of the total reward that the agent would receive under the learned policy at that point of the learning process.

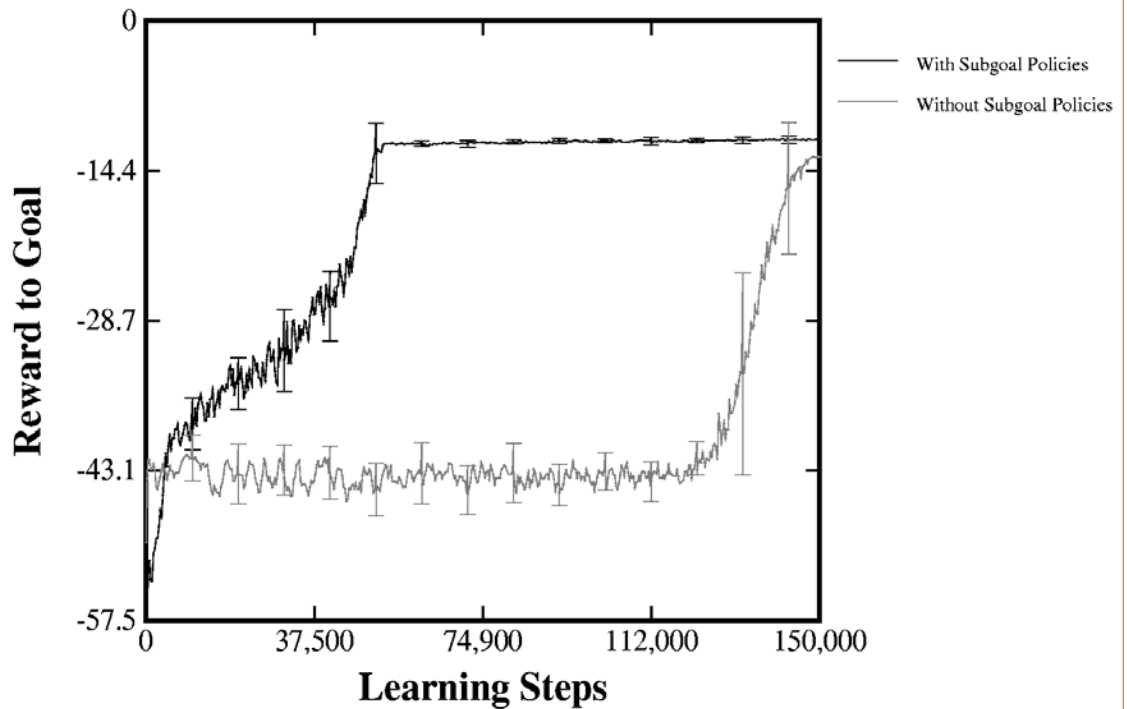


Figure 5.7 Comparison of learning speed using subgoal policies and using primitive actions only

The curves in Figure 5.7 are averaged over 10 learning runs. Only an initial part of the data is plotted to compare the two learning curves. With primitive actions only the agent is still learning after 150,000 learning steps while with subgoal policies the policy has already converged. After 400,000 learning steps the agent without subgoal policies also converges to the same overall performance. The vertical intervals along the curve indicate one standard deviation in each direction at that point.

To emphasize that the method for discovering subgoals discussed above is not confined to grid-worlds or navigation tasks a similar experiment was performed with randomly connected world discussed in section 4.1.2. Initial and goal state for the task were chosen randomly and Q-learning was used to learn the policy. The agent used the same procedure as discussed in previous experiment to discover subgoals and discovered three states as subgoal states. It used Q-learning to learn a policy to each of the subgoals discovered in the previous step and add them to its action set. To ascertain that these subgoals help the agent to improve its policy more quickly, two experiments were performed where the agent learned a new task with and without the subgoal policies.

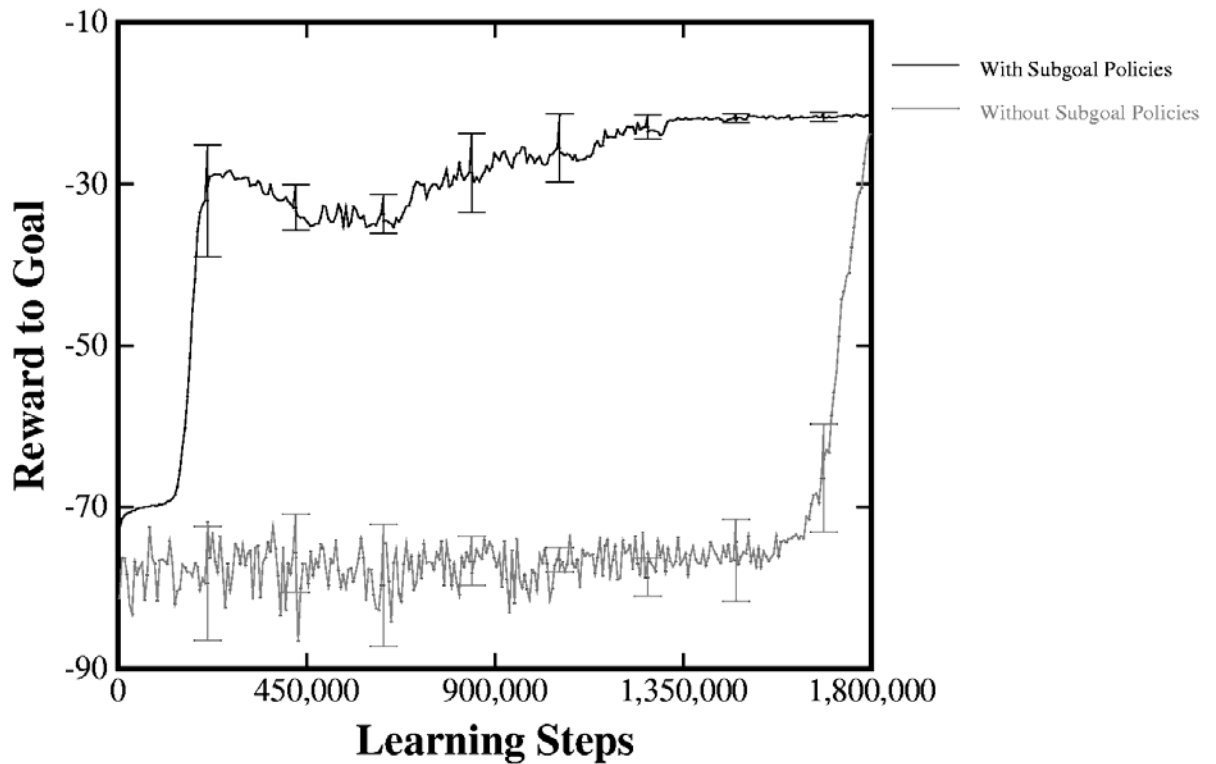


Figure 5.8 Comparison of learning speed using subgoal policies and using primitive actions only

The same random world was used to perform the experiments. Figure 5.8 shows the learning curves when the agent was using the subgoal policies and when it was using only primitive actions. The learning performance is compared in terms of the total reward that the agent would receive under the learned policy at that point of the learning process. The curves in Figure 5.8 are averaged over 10 learning runs. Only an initial part of the data is plotted to compare the two learning curves but eventually both converges to same level. With primitive actions only the agent is still learning while with subgoal policies the policy has already converged. The vertical intervals along the curve indicate one standard deviation in each direction at that point.

To show that the results discussed above do not depend on a particular world another experiment was performed using randomly connected worlds as discussed in section 4.1.2. Similar to the previous experiment, a randomly connected world was generated and Q-learning was used to learn the policy for a randomly chosen initial and goal state. The agent discovered the subgoals, learned the policies to achieve respective subgoals and added them to its action set. Now, the agent learns a policy for a new task in the same world with and without subgoal policies. The learning performance is recorded in terms of the total reward that the agent would receive under the learned policy at that point of the learning process for both cases, i.e. for the agent learning a policy with and without subgoal policies. The complete process is repeated for 10 different random worlds, i.e. for every run a complete new randomly connected world with the same number of states and primitive actions is generated. Figure 5.9 shows the

learning curves averaged over 10 different randomly connected worlds when the agent was using the subgoal policies and when it was using primitive actions only. Curves are scaled in order to have the same maximum reward value on the Y-axis. The vertical intervals along the curve indicate one standard deviation in each direction at that point.

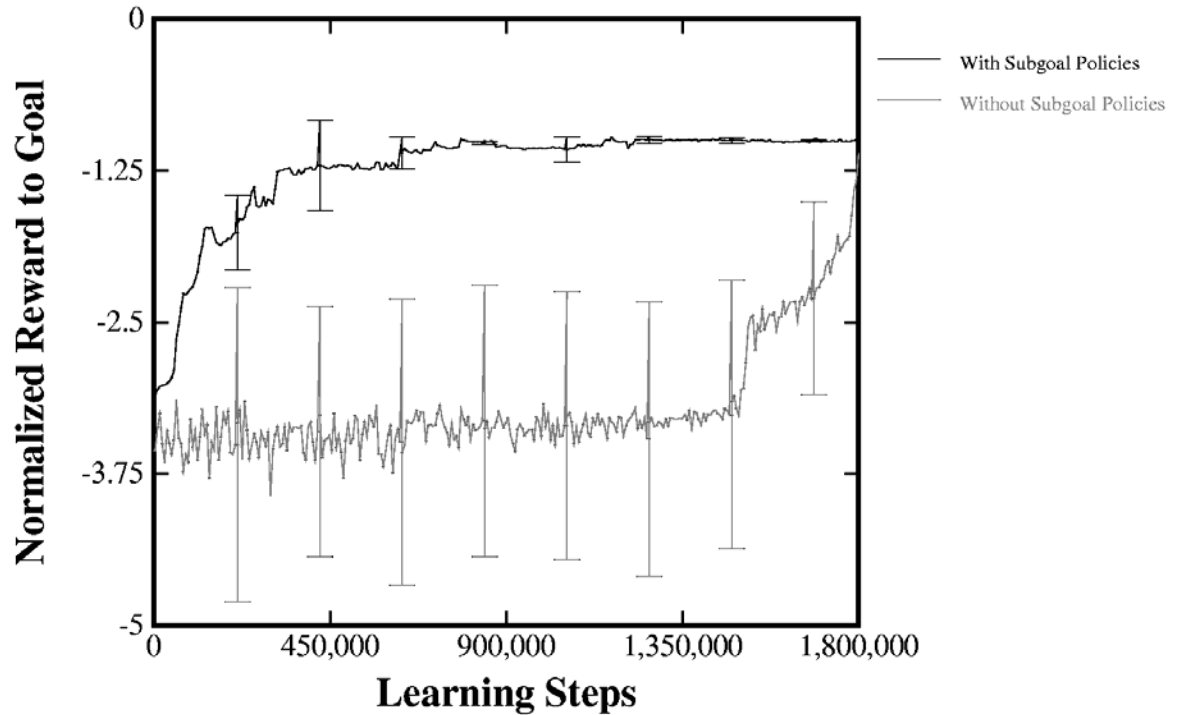


Figure 5.9 Comparison of learning speed using subgoal policies and using primitive actions only averaged over 10 different worlds.

CHAPTER VI

CONCLUSIONS & FUTURE WORK

An agent could potentially learn much faster if it could abstract the innumerable micro-decisions, and focus instead on a small set of important decision. This immediately raises the question of how to recognize hierarchical structures within learning and control systems and how to learn strategies for hierarchical decision making. Within the reinforcement learning paradigm, one way to do this is to introduce subgoals with their own reward functions, learn policies for achieving these subgoals, and then include these policies as actions. This strategy can facilitate skill transfer to other tasks and accelerate learning.

In this thesis, a method for automatically identifying subgoals and creating hierarchical actions has been presented. These subgoals are discovered by searching a learned policy model for states that exhibit a funneling property. The subgoals are identified by observing the dynamics along the count curve. The method presented here can also identify subgoals that are not an integral part of the current task learned. By creating totally random worlds for the purpose of experiments presented in this document the generality of this approach has been illustrated.

Hierarchical actions are created using subgoals which in turn are discovered by searching for states that connect separated strongly connected regions in the agent's

observation space. This has been formulated as a problem of finding states that exhibit certain structural properties in a learned policy model and the concept of studying dynamics along count curve has been used to identify subgoals. The method presented here has been illustrated in several simulated tasks and it has been showed that the introduction of hierarchy in reinforcement learning can both accelerate learning and facilitate transfer to related tasks.

One potential drawback of this method is the requirement of a threshold to differentiate between subgoal states and non-subgoal states. Though threshold can be established by the statistical tests over the distribution of gradient ratios, this might not be the best threshold. We are currently investigating how to establish a formal threshold instead of using an empirical one.

Our future research lies in the context of generalizing over the policies. Generalization means finding a common policy for achieving the goals of individual policies. Referring to the example of the tennis game given in Chapter I, “play a tennis stroke” is a higher level action, similarly a game of squash also has a “play a squash stroke” action but humans doesn’t have two different policies to execute them otherwise it would be very difficult to play different games. Instead humans has a general abstract action “play a stroke” generalized across the “tennis” and “squash” variables. Similarly, instead of having a different policy for every subgoal, a general policy would further accelerate learning and facilitate knowledge transfer

REFERENCES

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D., editor, *Machine Intelligence 3*, volume 3, pages 131–171. Elsevier, North Holland, Amsterdam, London, New York.

Andre, D. and Russell, S. J. (2001). Programmable reinforcement learning agents. In *Proceedings of Advances in Neural Information Processing Systems 13*, pages 1019–1025. MIT Press.

Bernstein, D. S. (1999). Reusing old policies to accelerate learning on new MDPs. Technical Report UM-CS-1999-026, Dept. of Computer Science, Univ. of Massachusetts, Amherst, MA.

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence*, 11:1–94.

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1550–1556.

Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous time Markov decision problems. In *Advances in Neural Information Processing Systems 7*. MIT Press.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. The MIT Press, Cambridge, MA.

Dean, T. and Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), pages 1121–1127, Montreal, CA.

Dietterich, T. G. (1998). Hierarchical reinforcement learning with the MAXQ value function decomposition. In Proceedings of the 15th International Conference on Machine Learning ICML'98, San Mateo, CA. Morgan Kaufmann.

Digney, B. (1996). Emergent hierarchical structures: Learning reactive / hierarchical relationships in reinforcement environments. From animals to animats 4: SAB 96. MIT Press/Bradford Books.

Digney, B. (1998). Learning hierarchical control structure for multiple tasks and changing environments. From animals to animats 5: SAB 98.

Drummond, C. (1998). Composing functions to speed up reinforcement learning in a changing world. In European Conference on Machine Learning, pages 370–381.

Gullapalli, V. (1992). Reinforcement Learning and its Application to Control. PhD thesis, University of Massachusetts, Amherst.

Gusfield, D. (1997). Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge.

Hauskrecht, M., Meuleau, N., Boutilier, C., Kaelbling, L. P., and Dean, T. (1998). Hierarchical solution of Markov decision processes using macro-actions. In Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence, pages 220–229.

Huber, M. (2000). A Hybrid Architecture for Adaptive Robot Control. PhD thesis, University of Massachusetts, Amherst.

Huber, M. and Grupen, R. A. (1997a). A feedback control structure for on-line learning tasks. *Robots and Autonomous Systems*, 22(3-4):303–315.

Huber, M. and Grupen, R. A. (1997b). Learning to coordinate controllers – reinforcement learning on a control basis. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1366–1371.

Huber, M. and Grupen, R. A. (1998). Learning robot control - using control policies as abstract actions. *Proceedings of the NIPS '98 Workshop on Abstraction and Hierarchy in Reinforcement Learning*.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317.

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

Kaelbling, L. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173. Morgan Kaufmann.

Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 923–928, Boston, MA. MIT Press.

Knoblock, C. A. (1991). Automatically Generating Abstractions for Problem Solving. PhD thesis, School of Computer Science, Carnegie Mellon University. Available as Technical report CMU-CS-91-120.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77.

Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321.

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

Makar, R., Mahadevan, S., and Ghavamzadeh, M. (2001). Hierarchical multi-agent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada.

Maron, O. (1998). Learning from Ambiguity. PhD thesis, Massachusetts Institute of Technology.

McCallum, A. K. (1995). Reinforcement Learning with Selective Perception and Hidden State. PhD thesis, University of Rochester, Rochester.

McGovern, A., and Barto, A. G. (2001a). Automatic Discovery of Subgoals in Reinforcement learning using Diverse Density. *Proceedings of the 18th International Conference on Machine Learning*, pages 361-368.

McGovern, A. (1998). Roles of macro-actions in accelerating reinforcement learning. Master's thesis, U. of Massachusetts, Amherst. Also Technical Report 98-70.

McGovern, A., and Barto, A. G. (2001b). Accelerating Reinforcement Learning through the Discovery of Useful Subgoals. Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space.

Moore, A. W., Baird, L. C., and Kaelbling, L. (1999). Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In Dean, T., editor, Proceedings of the International Joint Conference on Artificial Intelligence (*IJCAI99*), volume 2, pages 1316–1321, San Francisco, CA. Morgan Kauffman Publishers, Inc.

Newell, A., Shaw, J. C., and Simon, H. A. (1963). GPS, a program that simulates human thought. In Feigenbaum, E. A. and Feldman, J., editors, Computers and Thought, pages 279–293. McGraw-Hill, New York.

Parr, R. (1998). Hierarchical Control and Learning for Markov Decision Processes. PhD thesis, University of California at Berkeley.

Parr, R. and Russell, S. (1997). Reinforcement learning with hierarchies of machines. In Proceedings of Advances in Neural Information Processing Systems 10. MIT Press.

Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers, San Mateo, California.

Precup, D. (2000). Temporal Abstraction in Reinforcement Learning. PhD thesis, University of Massachusetts, Amherst.

Precup, D. and Sutton, R. S. (1997). Multi-time models for temporally abstract planning. In Proceedings of Advances in Neural Information Processing Systems 10. MIT Press.

Precup, D., Sutton, R. S., and Singh, S. (1998). Theoretical results on reinforcement learning with temporally abstract behaviors. In Proceedings of the Tenth European Conference on Machine Learning, ECML'98. Springer-Verlag.

Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics. *Machine Learning*, 12:117–141.

Ryan, M. and Pendrith, M. (1998). RL-tops: An architecture for modularity and re-use in reinforcement learning. In Proceedings of the 15th International Conference on Machine Learning ICML'98, San Mateo, CA. Morgan Kaufmann.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135.

Singh, S. P. (1992b). Reinforcement learning with a hierarchy of abstract models. In Proceedings of the Tenth National Conference on Artificial Intelligence, pages 202–207, Menlo Park, CA. AAAI Press/MIT Press.

Singh, S. P. (1992c). Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, 8:323–339.

Singh, S. P. (1994). Learning to Solve Markovian Decision Processes. PhD thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44.

Sutton, R. S. and Barto, A. G. (1998). Reinforcement Learning. An Introduction. MIT Press, Cambridge, MA.

Sutton, R. S., Precup, D., and Singh, S. (1998). Between MDPs and Semi-MDPs: learning, planning, and representing knowledge at multiple temporal scales. Technical Report 98-74, University of Massachusetts, Amherst.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Theocharous, G. and Mahadevan, S. (2002). Approximate planning with hierarchical partially observable markov decision processes for robot navigation. In *Proceedings of the 2002 IEEE Conference on Robotics and Automation (ICRA)*, Washington, DC.

Thrun, S. B. and Schwartz, A. (1995). Finding structure in reinforcement learning. In G. Tesauro, D. Touretzky, T. L., editor, *Advances in Neural Information Processing Systems: Proceedings of the 1994 Conference*, pages 385–392, San Mateo, CA. Morgan Kaufmann.

Watkins, C. J. C. H. (1989). Learning from delayed rewards. PhD thesis, Cambridge University, Cambridge, England.

BIOGRAPHICAL INFORMATION

In many respects, my career and my experiences with people and events have been seamless in that I cannot separate one from another. Without doubt, the thread of one's life should be within the matrix of the total human experience. -- *Les Prix Nobel*, 1994

Sandeep Goel was born on October 28, 1979, in Panipat, India. In his teens, he attended M.A.S.D. school, a "magnet" public high school with a strong science tradition. In 1996 he won National Talent Search Examination scholarship, passed Common Engineering Entrance Test and entered Maharishi Dayanad University in 1997. He studied computer science and earned his Bachelor's of Engineering in 2001. He moved to United States for his graduate studies and he entered The University of Texas at Arlington in 2001. He pursued research in Artificial Intelligence and joined Dr. Manfred Huber's research group at the university. After completing his Masters he intends pursue his PhD.