Task Localization, Similarity, and Transfer; Towards a Reinforcement Learning Task Library System

James Lamond Carroll
Brigham Young University - Provo
TASK LOCALIZATION, SIMILARITY, AND TRANSFER;
TOWARDS A REINFORCEMENT LEARNING TASK LIBRARY SYSTEM

by

James Lamond Carroll

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
August 2005
Copyright © 2005 James Lamond Carroll
All Rights Reserved
GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

James Lamond Carroll

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

__________________________________________  ____________________________
Date                                      Kevin D. Seppi, Chair

__________________________________________  ____________________________
Date                                      Michael A. Goodrich

__________________________________________  ____________________________
Date                                      Eric G. Mercer
As chair of the candidate’s graduate committee, I have read the thesis of James La-
mond Carroll in its final form and have found that (1) its format, citations, and 
bibliographical style are consistent and acceptable and fulfill university and depart-
ment style requirements; (2) its illustrative materials including figures, tables, and 
charts are in place; and (3) the final manuscript is satisfactory to the graduate com-
mittee and is ready for submission to the university library.

Date

Kevin D. Seppi
Chair, Graduate Committee

Accepted for the Department

David W. Embley
Graduate Coordinator

Accepted for the College

G. Rex Bryce
Associate Dean, College of Physical and Math-
ematical Sciences
This thesis develops methods of task localization, task similarity discovery, and task transfer for eventual use in a reinforcement learning task library system, which can effectively “learn to learn,” improving its performance as it encounters various tasks over the lifetime of the learning system.
There are many people who have contributed to this thesis, and many more who have contributed to my educational development which prepared me for this task. There are even more who contributed to my continued (although at times disputed) sanity while performing this task. Due to the limitations of space, and my own limitations of memory, I will only be able to mention a few of the many people who deserve credit. Although many helped during the composition of this thesis, I am solely responsible for any errors contained therein.

I would first like to thank my wife, Heidi Carroll, whom I love dearly, and who has been exceptionally patient with me while I worked on this research. I spent far too many hours away from her company. I appreciate her encouragement, thoughts, help, and prayers.

I would also like to thank my parents, Jim and Gwen Carroll, for teaching me, and for making education a priority in my life. Much of the good in my life I can trace to their influence.

I would also like to thank Dr. Peterson. Much of this research originated with my initial employment in his lab. The mentoring that I received at his hands was the real beginning of my education. He taught me more than math, numbers, and reward functions. His dedication to the gospel, and our wonderful conversations about its teachings have truly inspired my life.

Dr. Seppi, my current advisor, also deserves considerable credit for aiding me in my research, and pushing me to finish. His many suggestions have added mathematical rigor that otherwise would be lacking.

My friends in the aML and MILD research labs have helped me socially and
have contributed substantially to this research. I would like to thank: Nancy Owens Fulda (who developed the idea of soft transfer, contributed to the development of memory-guided exploration, and was a co-author on many of the papers upon which this thesis is based), David Wingate, Chris Monson, Reid Broadbent, Laramie Levitt, Ron Bjarnason, and Dr. Michael Goodrich.

I would also like to thank the Computer Science faculty of Brigham Young University, especially Dr. Mark Clement, Dr. Bryan Morse, Dr. Mike Goodrich, and Dr. Tom Sederberg, who taught me so well both in class and through example.

I would also like to thank Dr. Donald Parry and Dr. Stephen Ricks for influencing my preferred future career path.

I would also like to recognize the assistance that I have received from a heavenly source in insuring that my life fell upon its intended path, and for the more direct aid that I received in completing my course work and in succeeding at my research endeavors.
# Contents

1 Introduction and Motivation 1  
   1.1 Lifelong Learning 1  
   1.2 Reinforcement Learning 3  
   1.3 Task Transfer 5  
   1.4 Task Similarity Discovery 5  
   1.5 Task Localization 6  
   1.6 Conclusion 7  

2 Related Works 9  
   2.1 Lifelong Learning 9  
   2.2 Reinforcement Learning 10  
   2.3 Lifelong Reinforcement Learning 11  

3 Definitions and Notation 13  

4 MDP Suite 17  
   4.1 Simple Decision Task 17  
   4.2 Simple Grid World 19  
   4.3 Complex Grid World 20  
   4.4 Nomad II Robot Simulator 22
5 Task Localization

5.1 Task Localization Methods

5.1.1 MSE Technique

5.1.2 Trivial Bayesian Technique

5.2 Algorithm: Bayesian Task Localization Technique, (BTLT)

5.3 Methodology

5.4 Results and Discussion

5.5 Task Localization Conclusions and Future Work

6 Task Similarity Discovery

6.1 Introduction

6.2 Desirable Properties of a Task Similarity Measure

6.3 There is no Best Similarity Measure

6.4 Proposed Task Similarity Measures

6.5 Task Clustering

6.5.1 Clustering Motivation

6.6 Methodology

6.7 Evaluation

6.7.1 Moving Goal Experiments

6.7.2 Expanding Obstacle Experiments

6.7.3 Clustering

6.8 Task Similarity Discovery Conclusion

6.9 Future Research

7 Task Transfer

7.1 Introduction to Task Transfer

7.2 Transfer Techniques
List of Figures

4.1 Simple decision task. ........................................... 18
4.2 Simple grid-world. ............................................. 19
4.3 Complex grid-world ............................................ 21
4.4 Nomad II Simulator Display ................................. 22
4.5 Nomad II Simulator Display ................................. 23

5.1 Examples of Rewards That are Not Distributed Normally .... 29
5.2 Trivial Bayes ................................................... 39
5.3 Random Exploration .......................................... 40
5.4 Task Switching .................................................. 41
5.5 Average Reward vs Random .................................. 42
5.6 Average Reward Close vs Scratch ............................ 43
5.7 Average Reward Close Up .................................... 44

6.1 \(d_Q\) ................................................................. 57
6.2 \(d_Q\) Partially Learned ......................................... 58
6.3 \(d_P\) ................................................................. 58
6.4 \(d_P\) Partially Learned ......................................... 59
6.5 \(d_R\) ................................................................. 59
6.6 \(d_R\) Partially Learned ......................................... 60
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7</td>
<td>$d_Q$</td>
<td>61</td>
</tr>
<tr>
<td>6.8</td>
<td>$d_Q$ Partially Learned</td>
<td>62</td>
</tr>
<tr>
<td>6.9</td>
<td>$d_P$</td>
<td>62</td>
</tr>
<tr>
<td>6.10</td>
<td>$d_P$ Partially Learned</td>
<td>63</td>
</tr>
<tr>
<td>6.11</td>
<td>$d_R$</td>
<td>63</td>
</tr>
<tr>
<td>6.12</td>
<td>$d_R$ Partially Learned</td>
<td>64</td>
</tr>
<tr>
<td>6.13</td>
<td>Cluster Tree for both $d_Q$ and $d_P$</td>
<td>66</td>
</tr>
<tr>
<td>6.14</td>
<td>Cluster tree 3, based on $d_R.$</td>
<td>66</td>
</tr>
<tr>
<td>7.1</td>
<td>A source task can bias the learning of a target task.</td>
<td>72</td>
</tr>
<tr>
<td>7.2</td>
<td>Symmetry in a Simple Decision Task</td>
<td>76</td>
</tr>
<tr>
<td>7.3</td>
<td>Symmetry in a Simple Decision Task</td>
<td>76</td>
</tr>
<tr>
<td>7.4</td>
<td>Unlearning and Information Loss</td>
<td>77</td>
</tr>
<tr>
<td>7.5</td>
<td>Fixing Problem</td>
<td>99</td>
</tr>
<tr>
<td>7.6</td>
<td>Suboptimal Solution</td>
<td>100</td>
</tr>
<tr>
<td>7.7</td>
<td>Social Welfare</td>
<td>100</td>
</tr>
<tr>
<td>7.8</td>
<td>Related and Unrelated Transfer</td>
<td>101</td>
</tr>
<tr>
<td>7.9</td>
<td>Transfer Comparison</td>
<td>102</td>
</tr>
<tr>
<td>7.10</td>
<td>Nomad II Transfer Results</td>
<td>103</td>
</tr>
<tr>
<td>7.11</td>
<td>Simple three room stochastic environment.</td>
<td>104</td>
</tr>
<tr>
<td>7.12</td>
<td>Direct vs. Scratch</td>
<td>104</td>
</tr>
<tr>
<td>7.13</td>
<td>Fixed vs. Scratch</td>
<td>105</td>
</tr>
<tr>
<td>7.14</td>
<td>Fixed vs. Dynamic</td>
<td>106</td>
</tr>
<tr>
<td>7.15</td>
<td>Q-value Task Clustering</td>
<td>107</td>
</tr>
<tr>
<td>7.16</td>
<td>R-value Task Clustering</td>
<td>107</td>
</tr>
<tr>
<td>7.17</td>
<td>Clustered Transfer</td>
<td>108</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction and Motivation

Reinforcement learning is a powerful method for learning control. Unfortunately this method is often intractable in practice. This thesis extends the ideas of “lifelong learning” to the domain of reinforcement learning through task localization, task similarity discovery, and task transfer in order to improve the tractability of reinforcement learning.

In this chapter we will first discuss the role of “lifelong learning” in the general field of AI, and then discuss its extension into reinforcement learning. We will then introduce, define and motivate our technique for creating a lifelong reinforcement learning system through task localization, task similarity measures, and task transfer.

1.1 Lifelong Learning

Thrun and Mitchell [35] identified the following four major challenges to developing intelligent agents: knowledge (the need for accurate information and models), engineering (the need to make information computer accessible), tractability (the complexity of robot domains), and precision (the difference between the real world
and computer models, simulations, or plans caused by imprecise sensors and actu-
ators). One approach to overcoming these problems is with the “life-long learning” paradigm for machine learning (also known as “learning to learn”) [35] [8]. In the life-long learning approach, an agent encounters many different tasks over its lifetime. Often these tasks are related. Through the appropriate transfer of knowledge, it is possible for an agent to learn successive tasks with greater efficiency. In essence, the agent “learns to learn” [1]. This technique can address the four challenges to developing intelligent agents listed above.

**Knowledge** The agent can fill in the holes in its training set with knowledge drawn from other tasks. Thus, less training data is required for each new task encountered. This also allows more accurate world models to be built with less data.

**Engineering** Because the information and models are learned, they are computer accessible and do not need to be hard coded.

**Tractability** By transferring information from simpler problems to more complex problems, the tractability issues can be significantly reduced.

**Precision** By transferring information from one sensor/actuator configuration to another, the agent can adapt automatically to changes in precision and behavior in the sensors and actuators.

In short, many of the problems in machine learning could be solved or reduced through the “life-long learning” or “learning to learn” paradigm. Our work attempts to extend the lifelong learning approach to reinforcement learning.
1.2 Reinforcement Learning

Reinforcement learning is a method for learning a control policy by maximizing a reward function. We will focus on Q-learning, one of the more common methods for reinforcement learning.

In Q-learning, the expected discounted reward for taking an action $a$ in state $s$ is stored as a Q-value denoted $Q(a, s)$.

The Q Function is defined as:

$$Q(s, a) = r_{s,a} + \gamma \sum_{s'} p(s'|s, a) U(s'),$$

where $r_{s,a}$ is the reward given for taking action $a$ in state $s$, $U(s')$ is the utility of being in state $s'$, and $\gamma$ is a temporal discount factor and $p(s'|s, a)$ is the transition probability. This function can be learned using the following update equation:

$$Q(s, a) \leftarrow (1 - \alpha) * Q(s, a) + \alpha * (r_{s,a} + \gamma * V(s')),$$

with the value of being in a state $s$ being given by:

$$V(s) = \max_{\hat{a}} Q(s, \hat{a}),$$

and where $\alpha$ is a learning rate.

Unfortunately, Q-learning (and reinforcement learning in general) is often slow, fails to scale well for many complicated problems, and requires a large amount of training data. Several methods for overcoming these problems have been proposed [17]. However, none of these techniques have made reinforcement learning truly tractable for large problems.

Human beings are capable of solving many complex and novel control problems with very little training data. One likely reason for this ability is that humans are adept at applying information from past problems to new situations [30].
We propose a task library system as part of the “lifelong learning” paradigm, in which an agent improves its learning ability as it is exposed to each successive task. This allows the system to apply information from past problems to aid in the solution of new problems. Little has been said concerning the theoretical framework for learning to learn in the reinforcement learning domain [1], and much remains to be done in this area.

Most of the work in lifelong reinforcement learning that has been done has required extensive design intervention (the manual intervention of the designer or user of the system at the expense of system autonomy). A major contribution of a task library system would be the automation of the lifelong learning process. Such a system could also be useful for the automation of the shaping process [28] [27] [32]. Shaping is a technique for acquiring complex control policies by successively performing task transfer with increasingly complex versions of the problem.

This thesis represents the first steps towards creating such a task library system. The first three major parts of the task library system that we address are: 1) task localization, 2) similarity discovery, and 3) task transfer. Task localization determines if a given task is already in the library. If localization determines that the new task is not already in the library then similarity discovery determines which tasks from the library are most similar to the new task to be learned. Task transfer is the process whereby a similar task from the library is used to improve the learning of a new task.

The remainder of the introduction will give an overview of the three parts of the task library system in general. For purposes of clarity, we will introduce these parts in the opposite order in which they are performed by the agent, that is, task transfer, task similarity discovery, and task localization.
1.3  Task Transfer

Task transfer is the process whereby information from one task (known as the source task) is used to improve the learning of a related task (known as the target task). For any life-long learning system to succeed it will be important for an agent to be able to transfer information successfully between tasks.

There are several different mechanisms for transfer in reinforcement learning [3] [2] [14]. We will show that transfer from similar tasks can be helpful, while transfer from tasks with low similarity can be detrimental. This effect has also been observed by others [2]. When information is only transferred from a sub-portion of the problem, then the technique is known as a “sub transfer” or “piecewise transfer” technique.

We will introduce three new task transfer mechanisms, compare their behavior to each other, and compare their behavior to known transfer techniques in several environments.

1.4  Task Similarity Discovery

Similarity discovery is the process by which similar tasks are selected from the library to use for transfer. This is necessary since task transfer is only effective when the source task(s) are similar to the target task. There are many different measures of task similarity that could be used, and tasks can be similar in one respect while differing significantly in another.

We will analyze different similarity measures. We will also show that it is possible to use a given similarity measure to cluster similar tasks together, thereby allowing the agent to extract a set of invariants from such a cluster that can be used in transfer.
1.5 Task Localization

Task localization is the process of determining whether or not a target task is identical to a given source task. A library of previously learned tasks is only directly useful if there is a technique for recognizing when a situation matches one that has already been learned. Without this ability, the agent would be forced to re-learn the task. Furthermore, if the system only recognizes the identity of a target task after completely re-learning it, then it is too late to exploit any information from the source task to improve the learning of the target task.

If the reward structure of a task is given to the agent, task localization is simple. However, since the reward function is not always given to the agent in terms of its states and actions, and since rewards are often received in a stochastic manner, a complete task library system must be able to localize the agent in task space by simply observing the distribution of the rewards received. Such a situation arises in multiagent situations where the identity or behavior of the other agent is unknown but may have been encountered before, in any search where the goal location is not known but may have been encountered before, and in control when the system dynamics remain the same but the desired behavior changes.

A task localization algorithm should, therefore, have three main properties: 1) efficiency, meaning that it is able to localize with as few examples as possible so that localization can be performed before a given task is thoroughly re-learned; 2) robustness, meaning that it functions regardless of the distributions of the reward structure encountered; and 3) adaptability, meaning that it will adapt to a new situation in reasonable time even if the target task is not in the library.

We will introduce a task localization technique that has all three of the desirable properties mentioned above, and compare its performance to several naïve approaches
1.6 Conclusion

“Learning to learn” or “lifelong learning” has been shown to improve many AI algorithms, but little work has been done on extending this work to a reinforcement learning domain. We propose a task library system that could automate the “learning to learn” process in reinforcement learning. We take the first three steps towards the creation of such a system, namely: task localization, task similarity discovery, and task transfer.

The remainder of this thesis will be organized as follows: Chapter 2 will discuss related work, Chapter 3 will give some important definitions and notation, Chapter 4 will discuss a suite of tasks that will be used to test our algorithms, Chapter 5 will discuss task localization, Chapter 6 will discuss task similarity and task clustering, Chapter 7 will discuss task transfer, and Chapter 8 will give conclusions and propose future work.
Chapter 2

Related Works

This thesis extends the idea of lifelong learning to the domain of reinforcement learning. There are several categories of related work. First, techniques for lifelong learning in general; second, techniques for improving reinforcement learning without lifelong learning; and third, previous attempts at applying lifelong learning techniques to reinforcement learning.

2.1 Lifelong Learning

Considerable past work has been done in “learning to learn,” and “lifelong learning.” Most of this work has focused on classification problems and neural networks.

In the realm of Neural Networks, EBNNs (Explanation Based Neural Networks) allow information from related tasks to influence the training of a new neural network by changing the update rule to bias updates toward information from other tasks [23] [20]. Daniel Silver found that sharing a hidden layer between related tasks could improve learning and generalization between related tasks [31]. These examples are basically task transfer techniques for neural networks in classification, which is related
CHAPTER 2. RELATED WORKS

to our techniques for transfer in reinforcement learning.

Some of the work that has been done in the classification domain has a more direct relationship to our work in reinforcement learning. For example, Sebastian Thrun’s work on classification task clustering (what he called the TC algorithm) [36] was a major inspiration for our Reinforcement Learning Task Clustering algorithm. He defined a similarity metric to measure task relatedness between classification tasks and then selectively applied a transfer technique designed for “knn” learners, with the cluster of tasks that were similar to the target task.

In the realm of classification, very little work has been done to develop task similarity measures. For Thrun’s TC algorithm, he defined the distance between tasks based on how similarly they optimally weighted each feature.

Adam Peterson’s work is also related. Rather than measuring the distance between classification tasks, he measured the distance between classifiers based on the amount of overlap in how they classified individual examples [24].

2.2 Reinforcement Learning

In general, reinforcement learning is too slow to be practical for large problems. This thesis explores lifelong learning as a solution to this problem but several other techniques have been proposed. There are far too many techniques for improving reinforcement learning to exhaustively list them all here, but the following are typical.

There are several techniques that focus on changing the exploration strategy in various ways. This sort of work is related to active learning, selective sampling, and value of information, in that the agent autonomously attempts to determine which actions to take, and therefore, which data to acquire. Thrun’s work on exploration in reinforcement learning [34] is an early example and is typical of the domain.
Another technique for speeding reinforcement learning involves changing the update orders. Some of these techniques simply update several steps into the past [25], while others use a queue to prioritize the updates that will do the most good [22].

Other approaches have involved a hierarchical-feudal approach. In Feudal Reinforcement Learning [9], various learners are in charge of areas of the state space, and each one controls the agent when the agent is within its own area. These controllers are organized hierarchically and give rewards to the controllers over which they have responsibility.

Another technique to improve reinforcement learning is to employ a function approximator to the reward function. Such a system can be unstable due to the feedback loops involved (the estimation of the value function in one state is dependant on the estimation of the value function in another state, and vice versa), however under certain techniques the value function can be stably approximated [16].

There are many more techniques for improving reinforcement learning. The most important of these techniques have been summarized by Kaelbling [17]. Although several of these approaches have shown some promise, none of them have truly made reinforcement learning tractable in general.

2.3 Lifelong Reinforcement Learning

The lifelong learning approach is perhaps one of the most promising avenues for making reinforcement learning tractable. Unfortunately very little has been done on “learning to learn” in reinforcement learning [1].

The few exceptions include the following: Ring introduced the CHILD algorithm which uses Temporal Transition Hierarchies to generate Q-values for agent actions based on percept information from recent observations [29]. Maclin and Shavlik used
EBNNs to create an advice taking reinforcement learning agent by using EBNNs to model the reinforcement learning process and then using information from past tasks to influence the neural network of the new task [18]. Thrun uses a library of learned tasks and a description length parameter to determine a set of sub-policies that many tasks share called SKILLS [38].

Several different mechanisms for transfer in reinforcement learning have been proposed [14] [2] [3]. Dixon independently developed a technique similar to our memory-guided exploration, which guides an agent’s exploration of the environment using information from past tasks [14]. Fixed sub-transfer (one of the task transfer techniques that we compare against) was introduced by Bowling and Veloso [2] [3].

In general, transfer from similar tasks has been found to be helpful, while transfer from tasks with a low similarity can be disasterous [25]. We therefore attempt to develop transfer techniques that are more robust to differences between the source and target tasks.

All of the previous work in reinforcement learning task transfer has focused upon the single source task to single target task case. Full task libraries will likely require the simultaneous use of multiple source tasks.

The scarcity of prior work in this area is strange, and has motivated our work in this area.
Chapter 3

Definitions and Notation

In this chapter we will introduce several definitions and notational conventions that will be used in the remainder of this thesis. These include notation for markov decision processes, task libraries, inductive bias, task transfer mechanisms, task similarity measures, and “best” task similarity measures. We assume that the reader is familiar with the basic concepts of Markov Decision Processes, MDPs, and reinforcement learning [17].

Definition 1: Markov Decision Process

We represent a Markov Decision Process (MDP) as a 4-tuple, $(S, A, P(s'|s, a), f(r|s, a, s'))$ where $S$ is the state space, $A$ is the action space, $P(s'|s, a)$ is the transition matrix, and represents the probability of reaching state $s'$ from state $s$ when performing action $a$, and $f(r|s, a, s')$ is the probability density of rewards $r$ received when performing action $a$ in state $s$ and transitioning to state $s'$.

Since our rewards are received stochastically, let

$$R_k(s, a) \sim \sum_{s'} f_k(r|s, a, s') P(s'|s, a),$$

and let $r_k(s, a)$ be data drawn from the random variable $R_k(s, a)$. 

13
Definition 2: Task Library

A library of related MDPs is denoted by $L$ with identical state space $\mathbb{S}$ and identical action space $\mathbb{A}$. Task $l \in L$ is characterized by its unique stochastic payoff function, denoted by $f_l(r \mid s, a, s')$, and its state transition probabilities $P_l(s' \mid s, a)$.

Let $n$ be the number of tasks in our library $L$ and let $k$ represent the new task from which we are sampling and trying to match to some task in our library. For simplicity we define $T_i$ such that $P(T_i) = P(k = i)$, or the probability that our new task $k$ is the same as some task $i$ in our library and where $i$ can be $1...n$, or $n + 1$ if the task is new and not in the library.

Definition 3: Multiple Goals Markov Decision Process

When transition probabilities are the same for all identical state-action pairs across all tasks in $L$, then the collection of tasks is known as a Multiple-goal Markov Decision Problem, or MGMDP [15]. Formally: if $\forall j, i \in L, \forall s, s' \in \mathbb{S}, \forall a \in \mathbb{A} \ P_j(s' \mid s, a) = P_i(s' \mid s, a)$ then $L$ is an MGMDP. In an MGMDP, the stochastic reward function $f(r \mid s, a, s')$ is the only difference between any two tasks in the library.

As an example of a MGMDP, imagine a set of mazes (grid worlds), with the goal in different locations, but with everything else left the same. This would be a simple example of an MGMDP; although, more complex examples could be imagined. The reward functions of the tasks can differ in any way, so long as the transition matrix remained the same, and the states and actions are left untouched.

Definition 4: Inductive Bias

Inductive bias is a difficult concept to define. For this thesis we will use the definition of inductive bias from Mitchell [21]. Consider a concept learning algorithm $A$ for the set of instances $X$. Let $c$ be an arbitrary concept defined over $X$, and let $D_c = \{ < x, c(x) > \}$ be an arbitrary set of training examples of $c$. Let $A(x_i, D_c)$
denote the classification assigned to the instance $x_i$ by A after training on data $D_c$.

The inductive bias of A is any minimal set of assertions B such that for any target concept $c$ and corresponding training examples $D_c$

$$(\forall x_i \in X)[(B \land D_c \land x_i) \implies A(x_i, D_c)]$$

An example of an inductive bias is the way decision trees favor smaller, simpler trees over more complex ones. This bias is brought about by the way in which the trees are formed.

**Definition 5: Task Transfer Mechanism**

A task transfer mechanism is any mechanism for using information from one MDP in order to influence the learning of another MDP. In essence, task transfer allows information from one MDP (the source task) to form an inductive bias used in the learning of another MDP (the target task). For example, if the transfer technique was to initialize the target task’s value function to the value function of the source task, then those initial values form a set of assumptions B, while $D_c$ is the agent’s actual experience in the world. Together B and $D_c$ allow the agent to create a set of classifications for its actions in each state. Thus reinforcement learning can be thought of as a specific type of classification task, where the classifications are the actions to be taken in each state in order to maximize some reward. Furthermore since the agent would produce a different set of classifications if it had a different initialization for its value function, the transfer technique and source task can be described as imposing an inductive bias on the learning of the target task.

**Definition 6: Task Similarity Measure**

A task similarity measure is a quantification of the similarity between task $l \in L$ and task $i \in L$. Its inverse can be thought of as the “distance” between task $l$, and $i$,
$d(l, i)$ so the smaller $d(l, i)$, the more similar the two tasks are.

**Definition 7: “Best” Similarity Measure**

The “best” task similarity measure $d_B(k, i)$ is the measure that provides the weak partial ordering of the distance from an arbitrary target task $k$ and the other tasks in $L$ so that their ordering represents their relative advantage that would be gained by using each task $i \in L$ as a source task when learning task $k$ for any transfer technique. Thus if $d_B(k, i_1) > d(k, i_2)$, then we can assume that $i_2$ is more similar to task $k$ than is task $i_1$, and therefore provides a greater transfer advantage when learning task $k$ for all transfer techniques.

Later we will prove that no such global best similarity measure exists independent of a specific task transfer technique.
Chapter 4

MDP Suite

We will use the following suite of Markov Decision Problems to test our various algorithms: a simple decision task, a simple grid world, a complex grid world, and a Nomad II Robot simulator. This chapter discusses each of these test environments:

4.1 Simple Decision Task

We will begin with the most trivial of our Markov Decision Tasks, which is important for its pedagogical value when discussing task transfer.

Figure 4.1 shows a simple decision task. In this world, $S$ is a set of nodes arranged in a tree shape. $A$ is one dimensional with the number of actions equal to the branching factor $b$. The agent starts at node 0 and performs actions that take it to one of the branches of the tree, terminating at one of the leaf nodes of the tree. The goal is placed in one of the leaf nodes, and the agent’s task is to find it. $f(r|s, a, s') = 1$ if the agent takes an action that leads it to a state $s'$ that contains a goal, and 0 otherwise. Once the agent reaches a leaf node, the task begins over, and the agent is placed at the top of the tree. The transitions, $P(s'|s, a)$, are deterministic. The tasks
can vary in depth $d$ and branching factor $b$ yielding tasks of arbitrary complexity.

In the first phase of training, the agent is trained with the goal in a randomly chosen fixed location. Once the agent has learned this task, the task is changed by moving the goal to a neighboring branch of the tree. In this second phase, the agent’s task is to adapt to this new situation. The closer the target goal location is to the source goal location, the more the policies of the two tasks will overlap, and the more “similar” the two problems become. We will discuss the problem of determining task similarity in greater depth in Chapter 6.

This type of task is similar to problems like chess, where states that are topologically similar can have vastly different utilities, and where (in general) the agent is not allowed to return to prior states. States that are topologically close to the goal often have relatively low Q-values, even though there is a substantial overlap in the policies necessary to reach those states. This makes task transfer extremely difficult. These properties also cause many task transfer algorithms to perform differently in this situation than they do in conventional navigation tasks where the Q-values of
neighboring states are similar. This task also has different convergence properties than other tasks under certain transfer techniques, which will be analyzed later in this thesis.

\section*{4.2 Simple Grid World}

In this simple accessible grid world, \( S \) is a set of discrete \( x \) and \( y \) positions. The agent’s task is to navigate from the start to the goal while avoiding the walls. The reward function \( f(r|s,a,s') \) is -1 when the agent hits a wall, and +1 when the agent finds the goal (see Figure 4.2).

A consists of four actions: North, South, East, and West. Both a stochastic and
a deterministic version of this world will be used. In the stochastic version, \( P(s'|s, a) \) is such that an action will move the agent at right angles to its intended direction some percentage of the time, and this percentage can vary from task to task.

This grid world allows the rapid creation of a set of tasks that can be easily engineered to produce various levels of similarity. A suite of thoroughly learned tasks can easily be generated in this world. Moving the goal various amounts, removing the obstacle in the center, or swapping the start and the goal, can produce various levels of similarity.

This world has very different properties than the decision task. Unlike the simple decision task, in this task neighboring states have similar Q-values, which will cause some transfer techniques to function differently than they would in the simple decision task. Furthermore, the ability to return to previously visited states can create convergence problems with some off-policy controllers [33]. These problems are not present in the simple decision task. This will facilitate the exploration of the convergence properties of our transfer algorithms.

### 4.3 Complex Grid World

We use a complex grid world to test our task localization algorithms. In this world \( S \) consists of a discrete x and y position, as well as a direction which the agent faces. The agent can face one of eight directions. There are four actions in \( A \). The agent can either turn 45° to the right, turn 45° to the left, go forward, or go backward. This generates a much larger state space than in a traditional grid world.

Moves are probabilistic. \( P(s'|s, a) \) is such that when an agent moves forward or backward, the agent can either move one space clockwise or counterclockwise from its expected destination with some probability, which can vary from task to task.
4.3. COMPLEX GRID WORLD

The complex grid world can have multiple goals that can either be absorbing or not. Goals can generate rewards probabilistically when they are reached according to a normally distributed reward function with a given mean and variance $f(r|s, a, s') \sim N(m_{sas'}, v_{sas'})$. Each goal can set its mean and variance independently. Although the rewards pay off according to a normal distribution, because the transitions into these states are random, the reward seen by the agents in any state-action pair $R_k(s, a)$ will only be normal if the transitions are set to be deterministic.

A reward of -3 is given whenever an agent hits a wall, and a reward of -1 is given
whenever an agent moves backward. These payoffs are consistent for all states and all tasks. Therefore, this world provides a large potential for generalization within a single task and between tasks in a library. This allows us to place strong priors on certain payoffs that will be consistent across all tasks, while weaker priors can be placed on other states, which are more likely to vary between tasks.

4.4 Nomad II Robot Simulator

![Nomad II simulator display](image)

Figure 4.4: Nomad II simulator display, showing sonars and agent policy for a simple obstacle avoidance task.

In order to test the transfer techniques in a more realistic setting, we used a simulator for Nomad II Robots. $S$ consists of an $x$ and a $y$ position, together with the direction that the agent is facing. These values are approximately continuous. This environment is simply a large empty room. The agent’s sensors consist of eight
Figure 4.5: Nomad II simulator display, showing sonars and agent policy for a more difficult wall following task.

sonars. Since this environment is approximately continuous and the state space is extremely large, the agent uses a CMAC as a function approximator.

Two tasks were used in this world, an obstacle avoidance task (Figure 4.4) and a wall following task (Figure 4.5). The obstacle avoidance task was used as a source task to aid the learning of the more difficult wall following task.
Chapter 5

Task Localization

This chapter will discuss various algorithms for task localization and show that they are insufficient. We will then introduce our algorithm for task localization (the Bayesian Task Localization Technique, BTLT), and show that our technique has the three desirable properties discussed in Chapter 1. We will then illustrate our algorithm’s performance using various tasks created using the complex grid world (see Chapter 4).

In the MGMDP case, it is sufficient to look at the distribution of the rewards received in order to localize in task space. In the non-MGMDP case, the situation is more complex, and transition probabilities must also be considered. This is beyond the scope of this thesis, and we will assume that all tasks in this Chapter have the MGMDP property, and we will leave the non-MGMDP case for future work.

5.1 Task Localization Methods

There are three task localization methods that we will discuss. We will first discuss two simple solutions to the task localization problem in the MGMDP case, a Mean
Squared Error Technique and a Trivial Bayesian Technique. We will show that these techniques are insufficient, and that a better solution is needed. This will motivate our solution to the task localization problem which will be given in the next section.

5.1.1 MSE Technique

Perhaps the most straightforward technique for task localization is to sample from the target task \( k \) for some time, and then assume that the task is equivalent to the task in the library with the lowest mean squared difference in expected reward values [7]:

\[
\text{Task} = \arg\min_{i \in L} \sum_{s,a} (E[R_k(s, a)] - E[R_i(s, a)])^2,
\]

where \( k \) is the target task, and \( i \) is some task in the task library \( L \).

Because this technique does not take into account any weighting of states, it cannot take advantage of the fact that samples from the reward structure in one state may be more important than samples taken from another. Furthermore, this technique does not take into account the number of times that a specific \( R_k(s, a) \) has been sampled. If \( R_k(s, a) \) for some \( s, a \) has a high variance, then it must be sampled more than \( R_k(s, a) \) for another \( s, a \) with a low variance to achieve the same confidence in its expected reward. It is important to take this confidence into account because, to be useful, localization must be performed before the task is fully re-learned and therefore while the confidences are still relatively low.

Furthermore, the Mean Squared Error Technique does not return a probability, but a single task that minimizes the mean squared error. Therefore this technique requires that the task actually be in the library in order to function. In order to compute the probability that the target task is unique and not in the library, a more
5.1. TASK LOCALIZATION METHODS

As we will show in Chapter 6 the MSE technique can be used as an appropriate distance measure between two learned tasks. When used as a task similarity measure this technique is known as $d_Q$. Unfortunately this distance measure only functions when the tasks are thoroughly learned, when it is no longer useful for task localization (see Figure 6.2). Furthermore, it is not adaptable to a situation where the target task is not in the library.

5.1.2 Trivial Bayesian Technique

We will now show that a trivial Bayesian approach to this problem is insufficient, thereby justifying the more complex solution, which we will discuss later. If we were performing localization in the state space rather than in the task space, a standard approach would be to update the state probabilities at each step based on the current percepts [4]. A similar approach could be applied to localization in task space. In task space $P(r_k(s, a)|T_i)$ is the probability of the observed reward from task $k$ if task $k$ is equivalent to task $i$. The probability that we want, the probability that task $k$ is identical to task $i$, $P(T_i|r_k(s, a))$, can then be found by Bayes law:

$$P(T_i|r(s, a)) = \frac{P(r(s, a)|T_i)P(T_i)}{P(r(s, a))}$$

where

$$P(r(s, a)) = \sum_{i=0}^{n} P(r(s, a)|T_i)P(T_i).$$

Although a similar method has proven effective for localization in the state space, it has several problems when localizing in task space. First, the computation of $P(r(s, a)|T_i)$ is more difficult. If we assume that the rewards are distributed according to a parametric distribution, then the computation of $P(r(s, a)|T_i)$ is simply the statistically sound method is needed.
likelihood obtained from looking up \( r(s, a) \) in our parametric model of \( R_k(s, a) \). However, there are many situations in standard reinforcement learning where it is unclear what parametric model should be used. Figure 5.1 illustrates a simple case in which the reward function is not normally distributed. In fact, each state-action pair can have its own unique distribution, with no apparent pattern. Thus we must either keep full histograms for each state-action pair (which would be intractable), or we must assume that they follow some parametric distribution. Unfortunately, in standard reinforcement learning situations, relatively common samples of the reward function can appear to be very unlikely under the normal assumption and it is unclear which other parametric distribution could model such situations accurately.

Experimentally, this technique failed to converge in all but the most trivial examples. A detailed discussion of this technique is not central to the thesis, except to point out that in practical situations this technique must be discarded, and a technique that handles variations from the normal assumption in a more robust manner must be considered.

### 5.2 Algorithm: Bayesian Task Localization Technique, (BTLT)

Here we introduce a Bayesian technique that exhibits all three of our desirable qualities: efficiency, robustness, and adaptability. This algorithm is called the Bayesian Task Localization Technique (BTLT).

Suppose that \( R_k(s, a) \) is a random variable (see Chapter 3) with an unknown distribution and with an unknown parameter mean and standard deviation. We will model these two unknown values with two distributions of belief: \( M_k(s, a) \),
5.2. ALGORITHM: BAYESIAN TASK LOCALIZATION TECHNIQUE, (BTLT)

Figure 5.1: This illustrates a situation where $R(s, a)$ is not distributed normally. The agent either hits or misses the obstacle to its right when attempting to move forward depending upon the amount of slippage in its wheels (modeled by $P(s|s, a)$). Thus the agent either received a negative reward if the obstacle is hit, or a reward of 0 if the obstacle is missed. The negative reward, although rare, appears to be nearly impossible under the normal assumption.

and $S_k(s, a)$ which model our belief about the unknown mean and standard deviation respectively. Because $R_k(s, a)$ is not always distributed normally, estimations of $P(T_i|r(s, a))$ fail as shown in the previous section, because likely values can appear extremely unlikely due to violations in the normal assumption. An estimation of $P(T_i|M_k(s, a))$ can avoid this problem, because $E[M_k(s, a)]$ can be computed using a sum of $r_k(s, a)_1...r_k(s, a)_n$ samples drawn from $R_k(s, a)$. This sum will be normally distributed according to the central limit theorem so long as $n$ (the number of samples) is sufficiently large. Thus an individual sample $P(T_i|r(s, a))$, that would appear unlikely because of our violation of the normal assumption, can yield a likely
$P(T_i|M_k(s, a))$ so long as $n$ is sufficiently large.

Furthermore, by treating $M_k(s, a)$ as a distribution, we can model both the expected value of the mean and our confidence in that estimation. This is helpful because localization is usually performed before the task is thoroughly learned and thus while the number of samples is still small. By having a confidence in the mean for every state-action pair, those means that are the most confident can contribute the most to our localization, thus allowing localization with fewer samples. Furthermore, this also allows us to insert an empirical prior on $M_k(s, a)$ so that it can be estimated with fewer samples.

We chose to estimate $M_k(s, a)$ and $S_k(s, a)$ with a normal gamma model \[^{[11]}\] \[^{[13]}\] \[^{[12]}\] the conditional distribution of $M_k(s, a)$ when $1/S_k(s, a)^2 = u$, with $u > 0$, is a normal distribution with mean $\theta$ and precision $\tau u$ such that $-\infty < \theta < \infty$ and $\tau > 0$. The marginal distribution of $1/S_k(s, a)^2$ is a gamma distribution with parameters $\alpha$ and $\beta$ such that $\alpha > 0$ and $\beta > 0$. These four simple parameters are sufficient to represent our model of $R(s, a)$. The posterior joint distribution of $M_k(s, a)$ and $1/S_k(s, a)^2$ given a set of $n$ samples from $R_k(s, a), r_k(s, a)_1...r_k(s, a)_n$, is also a normal gamma distribution parameterized by $\tau$, $\theta$, $\alpha$, and $\beta$, and is updated as follows:

$$
\tau' = \tau + n,
$$

$$
\theta' = \frac{\tau \theta + n r_k(s, a)}{\tau + n},
$$

$$
\alpha' = \alpha + \frac{n}{2},
$$

$$
\beta' = \beta + \frac{1}{2} \sum_{i=1}^{n} (r_k(s, a)_i - \overline{r_k(s, a)})^2 + \frac{\tau n(r_k(s, a) - \theta)^2}{2(\tau + n)}.
$$
5.2. ALGORITHM: BAYESIAN TASK LOCALIZATION TECHNIQUE, (BTLT)

The marginal for $M_k(s, a)$ is a $t$ distribution with $2\alpha$ degrees of freedom and variance $\beta/\tau(\alpha - 1)$ [12].

Strictly speaking, this model requires that $R_k(s, a)$ be normally distributed; however, the estimation of $M_k(s, a)$ provided by this technique is robust to deviations from the normal assumption in $R_k(s, a)$ because in the normal gamma model, $E[M_k(s, a)] = \theta$. Note that $\theta$ is basically a running average of individual $r_k(s, a)$’s. This value’s computation will be correct regardless of variations from the normal assumption in $R_k(s, a)$. $Var[M_k(s, a)]$ may be slightly low due to the violations of the normal assumption, but empirically, this value provides an excellent approximation to our confidence in our estimation for the parameter mean (see Section 5.4).

Prior distributions for $M_k(s, a)$ are computed empirically from the other states in our task, and from the other tasks in $L$. Although task localization would not be performed when learning the first task the system encounters, the normal gamma model must still be built for all tasks in the library. Because there are no other tasks in the library when the first task is learned, priors must be estimated subjectively or drawn empirically from the other state-action pairs in the same task. As more tasks are inserted into the library, more information can be drawn from the corresponding state-action pairs from the other tasks in the library in order to create better priors. With these priors in place, we can more efficiently model $M(s, a)$ for our target and source tasks. Now the computation for $P(T_i)$ is fairly straightforward:

$$P(T_i|M_k(s, a)) = \frac{P(M_k(s, a)|T_i)P(T_i)}{P(M_k(s, a))}$$

by Bayes Theorem, and

$$P(M_k(s, a)) = \sum P(M_k(s, a)|T_i)P(T_i),$$

where $P(M_k(s, a)|T_i)$ can be found by computing the likelihood of $E[M_k(s, a)]$ in
the $t$ distribution with mean $= E[M_i(s,a)]$ and $Var = E[S_i(s,a)]/n_k(s,a)$ where $n_k(s,a)$ is the number of samples taken for action $a$ in state $s$ and task $k$, and with $2\alpha_i(s,a)$ degrees of freedom. This is true if we assume that our estimation of the mean and standard deviation are approximately equal to the true mean and standard deviation, $E[M_i(s,a)] \approx \mu_i(s,a)$ and $E[S_i(s,a)] \approx \sigma_i(s,a)$ where $\mu_i(s,a)$ and $\sigma_i(s,a)$ are the true values for the mean and standard deviation of $R_i(s,a)$. This will be true as long as the sample size $n_i(s,a)$ is sufficiently large. We therefore assume that we have thoroughly learned the tasks in the library, but we have made no such assumption about the target task $k$ that we are attempting to localize.

This means that we can use this technique to localize a target task $k$ within a library before $k$ is thoroughly learned, as long as all the source tasks in our library are thoroughly learned. Unfortunately, this technique requires that task $k$ be in the library. If the target task $k$ is simply added to the library as another task, $n + 1$, and the localization technique is run, because $E[M_k(s,a)]$ is not approximately equal to $\mu_k(s,a)$ for low $n_k(s,a)$, the algorithm does not function correctly until the new task is thoroughly learned, at which time it is too late to be of use.

The solution to this problem is to assume that the task is in the library and then determine the task in the library that is most likely identical with the target task. We will call this task $g$. Then a second statistical test is used to determine if $k = g$. This is a simple hypothesis test with two hypothesis, $H_0 : \mu_k(s,a) - \mu_g(s,a) = 0$, $H_1 : \mu_k(s,a) - \mu_g(s,a) \neq 0$ for all $s$ and $a$. Since the reward structure is continuous, the probability of $H_0$ is always 0. But the desired behavior for the agent is to assume that task $k$ is equal to task $g$ unless there is enough evidence to reject this hypothesis. Under $H_0$ we would expect $E[M_k(s,a)] - E[M_g(s,a)] \sim N(0, Var[M_k(s,a)] + Var[M_g(s,a)])$. If $E[M_k(s,a)] - E[M_g(s,a)]$ is within a 95% confidence interval of $N(0, Var[M_k(s,a)] + Var[M_g(s,a)])$, then we keep the null hy-
5.3 METHODOLOGY

We created a set of 14 tasks, each with a single goal placed randomly throughout the complex grid world (see Chapter 4). We then chose one of those tasks to be the target task. We attempted to determine whether the agent could recognize the task in its library which matched the target task. The agent made this determination by repeatedly sampling from each reward as it moved through the world. No other information was given to the agent to help it to localize.

In some of the experiments, the agent guided its exploration of the world with task switching. In this case the agent would perform the actions that maximized its expected utility in the task in its library that it currently considered to be the most
likely match. We also tested the case where the agent explored its world randomly. In the experiments, we tested varying amounts of randomness in the transition probabilities, specifically deterministic transitions and transitions with 1% randomness. Although this probability varied from experiment to experiment, it was kept identical for all the tasks in each library to maintain the MGMDP property.

5.4 Results and Discussion

The MSE technique provided a task similarity measure, and could pick out the most similar task by finding the task with the minimum distance (see Chapter 6), but, as we previously discussed, if the task was not in the library, it had no mechanism for determining when to reject the hypothesis that the current task was somewhere in the library [7]. Furthermore this technique would return the task with the minimum distance, but could not give any measure of how certain it was that the task with the current minimum distance was the correct task. The simple Mean Squared Error Technique did not require the normal assumption, and eventually converged to the correct solution even in situations without normally distributed rewards (see Chapter 6).

The Trivial Bayesian Technique functioned so long as the rewards were distributed normally (in our experiments, the case where the transition probabilities were deterministic). However, no consistent convergence was noted when the world randomness generated non-normal rewards. In these cases, the agent would compute inappropriately small probabilities for some of the rewards received, which would cause the probabilities for the tasks to either swing wildly or quickly converge to the wrong answer and remain there. Figure 5.2 is an example of the sort of divergent behavior encountered. In this case, the agent converged to task nine, instead of task fourteen,
which was the correct answer. This happened very early, and the agent never recovered. Which task appeared most probable often depended upon slight variations in the proportion of times these situations were seen in each task in the library.

BTLT performed much better. Even in the case where the rewards were not distributed normally (due to non-deterministic transitions), BTLT was able to localize after sampling from the goal state a few times (see Figure 5.3). Notice that our assumption that the task is in the library dictates that the probabilities in the graph sum to one at any given time. Often the probabilities would swing quite suddenly to the correct answer when an essential piece of information was sampled during the agent’s exploration of the world. Because there is randomness in the world, and in the agent’s exploration of the world, the results varied from trial to trial. We ran this experiment 5 times with the same target task and 10 times with different target tasks. In all cases, the agent correctly localized to the correct task. The agent converged as soon as the agent sufficiently sampled the reward from the goal, which was the key sample required to localize in this environment. In situations when the goal was in less accessible locations, the agent naturally took longer to generate the key samples required for localization with a random exploration pattern. The agent had to sample the goal no more than three times before it converged to the correct solution, and sometimes required no more than a single sample from the goal location.

Figure 5.4 is representative of the results encountered when task switching was employed instead of random exploration. As with random exploration, the agent converged to the correct task in all experiments, however it took much longer to localize when using task switching than it did when using random exploration (compare Figures 5.3 and 5.4). This is because the agent spent time performing actions from the incorrect tasks while the probabilities were being recomputed and was less likely to stumble into the key sample that would have allowed the agent to localize more
rapidly. However, with task switching, the average reward received by the agent was much higher than with random exploration (see Figure 5.5). Notice that localization still took place long before an agent could have learned the task from scratch. This sort of improvement was seen in all the experiments in this domain.

One unexpected result was that when using task switching, the agent initially explored its domain by performing policies that coincided with tasks in its library, and therefore received fewer negative rewards during its exploration phase. This added benefit happens because fault avoidant behavior is often uniform across tasks (see Figure 5.7).

When a goal was seen which did not exist in any task in the library, the agent would recognize that the task was not in the library with very few examples of an unexpected reward, as long as the confidence intervals were set correctly. We noticed that the results were very sensitive to this parameter (see 5.5 Task Localization Conclusions and Future Work). We also noticed that to avoid mistakenly rejecting the hypothesis that the task is in the library, results from state-action pairs with too few samples (in our case approximately less than or equal to six) should be ignored.

In some pathological cases (for example if we placed two goals in the world, one where a goal was in one of the tasks in the library, and another, far out of the way in a corner), BTLT can initially converge to the wrong solution. However, if the second goal was sufficiently sampled, the agent would realize that the task was novel. This is the standard exploration vs. exploitation tradeoff, and indicates that this method should be combined with one of the many well understood exploration methods.

In our case, moving the goals from task to task created a situation that was relatively easy to differentiate, and this is why our agent only had to sample from the goal location a few times in order to localize correctly. A more complex case could be imagined in which the goal is not moved, but the mean of its payoff is shifted slightly.
In such situations we would expect the agent to require more observations of the goal state in order to localize. The closer the means are, the more difficult localization would become, and the more samples would be required. However, in such situations, the more difficult the tasks are to differentiate, the more similar the tasks’ policies are. The more similar two tasks’ policies are, the less a delay in localization effects the agent’s payoffs. Thus, the more samples required for localization, the less a delay in localization matters, in terms of the agent’s policy and rewards.

5.5 Task Localization Conclusions and Future Work

We have shown how task localization, one of three major steps in the creation of a task library system, can be accomplished with a Bayesian approach in the MGMDP case. We have shown that the Trivial Bayesian Technique fails when the rewards received are not distributed normally. We believe that this will be the case in most reinforcement learning problems. Furthermore, the MSE technique cannot determine a confidence in its proposed localization and has no method of determining if the task is not in the library.

BTLT overcomes these problems by placing priors on the frequencies with which tasks are observed and on the reward structure of the tasks in its library. This allows the agent to make more appropriate guesses about the reward structure with fewer observations, and it allows the agent to localize in task space with fewer observations.

Although random exploration can allow faster localization, task switching can avoid many negative rewards received while the task space is being explored. Several parameters must be tuned if the hypothesis that the task is not in the library must be tested.

Although never encountered in our experiments, we believe that pathological cases
could be constructed in which the agent could converge to a sub-optimal solution without sufficient exploration. BTLT should therefore be combined with some other exploration vs. exploitation tradeoff technique. These techniques are well understood and have been widely studied.

There are several problems that we have left for future work. Determining when the task is not any of the tasks in the library is an extremely difficult problem, and is highly sensitive to the confidence interval used and the priors set on the target task. Although we were able to find settings that worked, more research should be performed to analyze how this process is affected by the parameters, and how it can be stabilized so as to function in a more robust manner. Another important step would be to extend this work to include the non-MGMDP case.

There is more work to do in the case where the goals are not moved, but the means of the goals’ payoffs are shifted. Although we ran several experiments in this case, which allowed us to make the general statements in the discussion section above, more experiments are needed to quantify this effect. This is difficult because the effect is clearly a function of more than just the difference in the goal means. The size of the world, the amount that the goal means are moved, and the standard deviation of the goal payoffs also play a part.
Figure 5.2: An example of incorrect convergence in the complex grid world as Trivial Bayes localizes, assuming that the task is in the library using a random exploration technique. World randomness is 1%, and therefore, the rewards were not normally distributed in this experiment. The y-axis is the probability of $P(T_i)$, for each of the 14 tasks, and the x-axis is the number of 10 world steps taken. The correct answer is task 14. Notice that the Trivial Bayes technique quickly converges to task 9. Although only the first few steps are shown in this graph, the agent never recovered.
Figure 5.3: The probabilities of 14 tasks in the complex grid world as BTLT localizes assuming that the task is in the library and using a random exploration technique. The y-axis is the probability, and the x-axis is the number of thousands of world steps taken. World randomness is 1%, and therefore, the rewards were not normally distributed in this experiment.
Figure 5.4: The probabilities of 14 tasks as BTLT localizes, assuming that the task is in the library, with task switching to the most probable task at any given time.
Figure 5.5: A comparison of the average reward received when localizing with task switching vs. localizing with random exploration. Although task switching can cause the agent to take longer to localize, the average reward received during localization is much higher than for random exploration.
Figure 5.6: Average reward received when learning from scratch vs BTTLT based task switching when the task is in the library. When compared to learning from scratch the time to localize appears immediate.
Figure 5.7: A closeup of Figure 5.6 showing that the number of negative rewards received before localization were considerably less than when learning from scratch.
Chapter 6

Task Similarity Discovery

In the event that localization determines that the task is not already in the library, it will be necessary to determine which tasks in the library are similar to the current task, how similar they are, and in what way they are similar so that this information can be used in transfer [see Chapter 7].

This chapter defines similarity in terms of tasks, proposes several possible task similarity measures, evaluates their performance, and proposes future work that should be performed in this area.

6.1 Introduction

In an extensive reinforcement learning task library there may be many tasks that are related to the target task, as well as many tasks that are totally unrelated to the target task. Since (as we will show in Chapter 7) task transfer algorithms are all extremely sensitive to the nature and amount of similarity that is present between the source and the target tasks, a method for quantifying the similarity of two tasks is needed.
Unfortunately, task “similarity” is an ill defined term. What does it mean for two tasks to be “similar” and for another two tasks to “not be similar?” How can levels of similarity be quantified? The problem is even more complex because tasks can be similar in several ways; thus, a task may be very similar in one respect, while being very dis-similar in another.

One possible method for defining similarity is in terms of content, meaning that similar tasks share specific features. However, it is unclear how much weight should be given to each shared feature. Even if this problem could be overcome, humans find similarity between tasks in more complex ways, for example, through analogy or metaphor. Analogies are “those problems that share a similar deep structure but not necessarily specific content” [30]. This means that tasks that have no superficial features in common can still be considered similar.

We define task similarity in this thesis with respect to a given transfer technique, where the level of similarity under a given transfer technique is an approximation to the “advantage” gained by using one task to speed the learning of another task. Furthermore, this definition of task similarity places the need to identify “deep structure” on the shoulders of the task transfer mechanism since if such a technique is developed, the advantage gained by using that technique can then be quantified.

### 6.2 Desirable Properties of a Task Similarity Measure

One of the standard uses for task similarity measures is to select a source task that can be used when learning a given target task. This is an important step since transfer from similar tasks can greatly speed the learning of the target task, while transfer
from an unrelated source task can greatly degrade performance.

Therefore, one possible task similarity measure would be to actually learn a target task given a source task, and somehow measure the “advantage” gained by using the source task compared to learning the target task from scratch, and call that difference the measure of similarity between the two tasks. This can capture the deep, analogical, or metaphorical similarity between the two tasks to the extent that the transfer technique is capable of utilizing such similarity. In some ways, this technique could be the best method for measuring similarity between two tasks.

However, in some ways this technique is not helpful. Although it produces a measure of similarity, it only produces such a measure after the transfer experiment has been run. If the point is to use the task similarity measure to choose a task to use in transfer, then this task similarity measure produces a measure of similarity after the task is learned, and therefore, once it is too late for most applications to use such a measure.

It should be noted that a measure of similarity need not be a “metric” or a “measure” in the mathematical sense of those terms, nor should it be. The process of shaping [25], for example, is based on the idea that it is faster to learn an intermediary task, and then use that task to aid the learning of a more complex task, than it is to learn the more complex task from scratch, i.e. $time(initial, goal) > time(initial, intermediate) + time(intermediate, goal)$. Thus, shaping depends on the fact that the triangle inequality does not hold for task similarity when task similarity measures the advantage of using one task to aid the learning of another. Nor is it clear that the properties of symmetry or identity should necessarily hold.

We formally define the term “task similarity measure” and its inverse “task distance measure $d(i, j)$,” as a heuristic function that has the following desirable properties:
1. The task similarity measure should provide an approximation to the amount of
learning improvement that we would get when using the source task to learn
the target task under a given transfer technique.

2. If \( d(l, i) > d(l, k) \) then we would hope that using task \( k \) to aid in the learning
of task \( l \), would provide a better bias for learning task \( l \) than using task \( i \).
Thus, the task similarity measure should provide an approximation to a partial
ordering for the similarity between task \( i \) and the rest of the tasks in \( L \) (where
\( L \) is a task library or set of tasks).

3. The task similarity measure should be computable without actually running
the transfer experiment. In other words, it should be able to produce an ap-
proximate partial ordering before task \( l \) has been thoroughly learned, while the
information could still be of use to aid in the learning of \( l \). The ordering can
be refined as the experiment runs, but the measure should provide a useful
approximation before the learning is complete.

6.3 There is no Best Similarity Measure

Having a “best” measure of similarity is like having a “best” inductive bias. We
would like to be able to say with some certainty exactly how similar two things are,
or that one pair of tasks is more or less similar than another pair of tasks. But given
the endless possibilities for analogies and metaphors, such a measure is impossible.

**Theorem 1: There is No “Best” Similarity Measure**

*Given Definition 7 of a “best” similarity measure from Chapter 3, there is no best
measure for similarity for all possible tasks in a task library \( L \) and for all task transfer
techniques \( t \) and for all learning algorithms.*
6.3. **THERE IS NO BEST SIMILARITY MEASURE**

Intuitively, this is because the transfer technique imposes an inductive bias on the learning of the target task. Since there is no best inductive bias for learning all target tasks (the “no free lunch theorem”), there is no best transfer technique to employ for all target tasks. Furthermore, some source tasks are more useful under one transfer technique than they are under another. Therefore, if similarity is defined as the expected usefulness of using one source task to speed the learning of another target task, there can be no “best” similarity measure apart from the transfer technique employed.

**Proof:**

The proof is by contradiction. Assume that such a “best” task similarity measure exists, and call that measure \( B \), and the distance measure that it imposes \( d_B \). There are three possible cases. 1) if task \( i \) imposes a better learning bias than task \( g \) when used as a source task while learning task \( k \), then we would expect a “best” similarity measure such that \( d_B(k, i) < d_B(k, g) \) for all task transfer techniques. 2) if \( g \) imposes a better learning bias than task \( i \), then the best similarity measure should yield \( d_B(k, i) > d_B(k, g) \) for all task transfer techniques. 3) if \( d_B(k, i) = d_B(k, g) \) then the two tasks are equally similar. Under Definition 7, one of these three cases must hold for a given \( k, g, \) and \( i \), and for all possible task transfer techniques. We will therefore show that there exists tasks \( k, g \) and \( i \) and two transfer techniques \( t_1 \) and \( t_2 \) such that \( d(k, i, t_1) < d(k, g, t_1) \) and \( d(k, i, t_2) > d(k, g, t_2) \), where \( d(x, y, t) \) is the distance between tasks \( x \) and \( y \), when using transfer technique \( t \). Since \( d(k, i, t_1) < d(k, g, t_1) \implies d_B(k, i) < d_B(k, g) \) and \( d(k, i, t_2) > d(k, g, t_2) \implies d_B(k, i) > d_B(k, g) \) this yields a contradiction.

Let \( k, g, \) and \( i, \) be binary functions \( f_w(x) = y \) where \( w \) is the task, \( k, g \) or \( i, \) and \( y \) is either 1 or 0 and \( x \) is an integer. Furthermore, let us assume that this simple
function represents either a reward function \( r(s, a) \) or a value function \( Q(s, a) \) in an MDP. In this proof, we have limited our value function to a single dimension and to binary responses, but a simple case that leads to a contradiction is sufficient to show that a property does not hold in general.

Let \( f_i(x) = f_k(x) \) for all \( x \) except \( x=0 \) and \( x=1 \), where \( f_i(x) = 1 \) if \( f_k(x) = 0 \) and \( f_i(x) = 0 \) if \( f_k(x) = 1 \). Let \( f_g(0) = 0 \) and let \( f_g(x) = f_k(x - 1) \) for \( x > 0 \).

Now we define two transfer techniques, \( t_1 \) and \( t_2 \), and an associated learning technique, \( r \), that is attempting to learn the value function of the target task \( k \). Let \( r \) be any learning technique such that the initial values of \( r \) can be “seeded” in some way such that the better the initial seeding of \( r \) the better its performance will be. This is the case in most learning techniques, and so long as these assumptions hold, \( r \) can be any learning technique. Thus, \( r(s_j) \) will represent the learning technique \( r \) seeded according to some technique \( s_j \). The manner in which \( r \) is seeded will define the transfer technique. Our transfer techniques will “seed” the initial guesses for the value function of the target task with different values from the source task, and then \( r \) will be allowed to learn normally.

Transfer technique \( t_1 \) uses \( r(s_1) \) as its learning technique, with \( s_1 \) such that \( f_k(x) = f_l(x) \), where \( l \) is an arbitrary source task. Transfer technique \( t_1 \) takes as its initial guess for the value function the corresponding value from whatever source function we give it. This technique is analogous to direct transfer which will be discussed in the next chapter, and is useful if a goal has moved some small amount. \( t_2 \) uses \( r(s_2) \) as its learning technique with \( s_2 \) such that \( f_k(x) = f_l(x - 1) \). Thus, \( t_2 \) initialization of each state is based on the source task \( l \)'s classification for the previous state. This transfer technique could be useful if a perceptor has been miss-calibrated, and is shifting all its inputs by some small amount.

Since we assumed that the nearness of our initial guess affects the quality of our
6.3. **THERE IS NO BEST SIMILARITY MEASURE**

learning, if \( t_1 \) is used as our transfer technique, then the preferred source task is \( i \), and if \( t_2 \) is used, then the preferred source task is \( g \), and thus \( d(k, i, t_1) < d(k, g, t_1) \) and \( d(k, i, t_2) > d(k, g, t_2) \). This implies that \( d_B(k, i) < d_B(k, g) \) and that \( d_B(k, i) > d_B(k, g) \), which contradicts our definition of a “best” similarity measure. Thus, the only reasonable description of similarity is in terms of the transfer mechanism used: \( d(k, i, t_1) < d(k, g, t_1) \) and \( d(k, i, t_2) > d(k, g, t_2) \).

Thus, our partial ordering of tasks is dependant on the transfer technique employed. Furthermore, there are situations where both transfer techniques could be useful. No one transfer technique can dominate in all situations, and therefore, no one similarity measure can dominate in all situations. Even more importantly, we can then say that no partial ordering of similarity between a set of tasks \( L \) and a given target task \( k \) exists independent of a given transfer technique.

**Discussion:**

The task transfer technique and the choice of source task(s) together impose a bias upon the learning of the target task. Because, as is well understood, there is no such thing as a “best” bias in the general case, there is therefore no best bias for the set of all possible target tasks. There is, therefore, no transfer technique that will be preferred for all source task and target task combinations.

Since there is no “best” transfer technique there can be no “best” similarity measure since we have already proven that if the transfer technique changes, the partial ordering on the source tasks can also change.

The above proof in no way shows that similarity measures are not useful in task libraries. It simply shows that any measure of similarity must be dependent on the task transfer method used.
6.4 Proposed Task Similarity Measures

We propose four task similarity measures, $d_T$, $d_P$, $d_Q$, and $d_R$.

$d_T$ is the technique already described above, where the transfer experiment is actually run, and the “advantage” is quantifiably measured. This technique actually requires that the transfer experiment be run, and is therefore not helpful in choosing tasks for transfer, but could have other conceivable uses. The exact manner in which the advantage should be quantified is one of the major challenges with $d_T$. We used several techniques, including the average reward received within some window of time, and the time to “convergence.” We used $d_T$ primarily as a reference against which to compare the other similarity measures.

Policy overlap ($d_P$) finds the number of states with identical policy (maximum utility). This is perhaps the most obvious approximation to $d_T$; however, this can be problematical in states with two or more actions with nearly equal utility. If the action with the maximum utility differs, but the difference in utility is small, should there really be no policy overlap? Furthermore, differences in policy in one state may be more important than differences in another. None of these features are captured by a simple policy overlap task similarity measure. Sebastian Thrun used $d_P$ together with a description length parameter to learn sub-skills in a suite of tasks [38].

If Q-learning is used, another simple task similarity measure can be constructed from the mean squared error between the Q-values $Q(s, a)$ (the expected discounted future reward for taking action $a$ in state $s$) of the source and target tasks. We call this task distance measure $d_Q$.

If the expected immediate reward for taking action $a$ in state $s$ (the R-values or $R(s, a)$) are stored, we can construct another task distance measure, $d_R$ from the mean squared error between these values in the source and target tasks.
Which task similarity measure (or combination of task similarity measures) to use will depend upon the types of tasks in the library and the transfer techniques that the agent uses.

### 6.5 Task Clustering

#### 6.5.1 Clustering Motivation

Once a task similarity measure has been established, it is possible to use that measure to cluster the tasks in a library. There could be many advantages to this clustering of tasks in a library. As the agent explores a portion of the space of a target task and finds that this portion is similar to the same portion in a group of clustered tasks, then it may be reasonable to assume that the new target task might also share similarities with this cluster of tasks in other parts of its state space. Thus, clustering could simplify the process of picking a set of tasks from which to transfer. Task clustering could even lead to the automation of this process.

Sebastian Thrun has already shown how task clustering might work with the generation of a task library system in the domain of classification tasks [36] [37]. In reinforcement learning, the situation is more complex. This section extends Thrun’s TC (Task Clustering) algorithm into the realm of reinforcement learning. We called this extension the RLTC (Reinforcement Learning Task Clustering) algorithm.

Task clustering may be useful in an eventual multiple task transfer mechanism because it may be easier to find a cluster of similar tasks than to determine the most similar task in a library. Furthermore, transferring features from a cluster of similar tasks may be more effective than transferring from the most similar task in the library, because the cluster of tasks may be more likely to capture invariants that...
all such tasks share rather than details specific to a given task [7]. Although Chapter 7 will demonstrate that multiple task transfer is possible, a detailed investigation of multiple task transfer is beyond the scope of this thesis.

The Reinforcement Learning Task Clustering Algorithm

We use a simple hierarchical merging clustering algorithm. Such clustering techniques create a tree structure. Leaf nodes correspond to the tasks being clustered. Parent nodes link successively more dissimilar tasks, or clusters of tasks, as we move from the leaves to the root of the tree. The parent nodes in the tree represent the cluster containing all of the leaf nodes which are descendants of that parent.

Our technique is capable of generating a cluster tree provided that $\forall_{i,j} \exists d(i, j)$, where $i$ and $j$ are any two tasks in the library. Our clustering algorithm takes as input a list of tasks and the distance between all possible pairs. Formally, the algorithm takes a list of $n$ tasks, $T_1...T_n$, and a corresponding $(n \times n)$ array, $D$, of distances $d(i, j)$, where $1 \leq i, j \leq n$. We then create a list of clusters $O$. Initially each task is placed in its own cluster, and thus, $O$ starts with $n$ clusters, with one task each.

The distance between any two clusters in the cluster list $d(C_i,C_j)$ (where $C_x$ is a specific cluster in $O$) is then computed as the average of the distance of all possible pairs of tasks from $C_i$ and $C_j$ (the edges in a bipartite, complete graph, $K_{m,n}$, comprised of the $m$ tasks in $C_i$ and the $n$ tasks in $C_j$). As the number of tasks in $C_i$ and $C_j$ becomes large we can use a subset or sub-sample to compute the mean distance, and thus, bound the computational complexity.

In each iteration of the algorithm, we select $l^*, k^* = \arg \min_{l,k} d_C(C_l, C_k)$ (where $l$ and $k$ iterate over every cluster) to find the two clusters, $C_{l^*}$ and $C_{k^*}$, which are closest to each other. These two clusters are used as the children of the next parent in what will grow into a binary tree. The two clusters $C_{l^*}$ and $C_{k^*}$ are then removed from the
6.6. METHODOLOGY

cluster list $O$, and are merged into a new cluster which is then added back into the list of clusters $O$. The cluster distances are then recomputed for all the cluster pairs in $O$. This process is repeated until all tasks have been connected into the binary tree, the root of which is the only cluster left in $O$.

This binary tree could then be collapsed by merging parent and grandparent nodes which link clusters or individual tasks that are approximately the same distance apart. In future work, we intend to explore the use of a t-tests on the distances to determine whether a particular cluster split is statistically significant, or whether it should be collapsed.

Different cluster trees will be generated with each distance measure used, and each tree will hopefully capture a different set of features that tasks might have in common. By analyzing the different trees generated by different distance measures it will be possible to better compare the different properties of each distance measure.

6.6 Methodology

To test our task similarity measures we performed three classes of experiments: the moving goal experiments, the expanding obstacle experiments, and the clustering experiments. In all cases, the complex grid world (see Section 4.3) was used, and the transfer mechanism employed was direct transfer (for more information on transfer mechanisms see Chapter 7).

1. In the moving goal case, we generated 74 different tasks by placing one goal in each task in varying positions chosen to create an approximately uniform distribution of tasks across the grid world maze. We picked one task, and used each of the other tasks as source tasks to speed the learning of this one target task generating $d_T$. We then analyzed how this advantage was approximated by
the other three distance measures $d_Q$, $d_R$, and $d_P$. We computed the distance from each task to the target task both with the target task thoroughly learned (ran until convergence) and partly learned (prematurely stopped at 300,000 steps). This allowed us to determine how quickly the measures were able to generate their approximations.

2. For the expanding obstacle set of experiments, we placed a single obstacle in the center of the complex grid world, and allowed this obstacle to vary in size from task to task. As before, we tested the distance measures both thoroughly and partially learned.

3. For clustering, we placed several goals in the general area of the bottom left, and several goals in the general area of the upper right. In half of the tasks, we removed all the obstacles from the complex grid world, generating a large open room. In the other half, we placed a single obstacle of uniform size and shape in the center of the world. This allowed us to compare each distance measure’s sensitivity to general policy trends (bottom left to upper right, or upper right to bottom left) with each distance measure’s sensitivity to the existence of the obstacle.

6.7 Evaluation

In this section we empirically evaluate the task similarity measures proposed in Section 6.4. We evaluate each task similarity measure relative to $d_T$, (the actual advantage gained by using one task to improve learning of another) in the moving goal, the expanding obstacle, and task clustering cases.
6.7. EVALUATION

Figure 6.1: $d_Q$ on the x axis vs $d_T$ on the y axis in terms time to convergence in 1000 step units. Smaller distance measures indicate more similarity between tasks, and therefore take less time to adapt using direct transfer. The moving goal case.

6.7.1 Moving Goal Experiments

In the case of the moving goal, we used a task with the goal in the upper right corner, and found the distance between that task and the other tasks in the library with goals scattered throughout the space.

The $d_Q(i,k)$ distance measure provides a good approximation to the speedup expected when using task $i$ to speed learning of task $k$, when using direct transfer as a transfer mechanism, as shown in Figure 6.1. Although the data is hetroscedastic (specifically, the variance increases as $d_Q$ increases), the number of iterations to learn a task rises approximately linearly with $d_Q$. In this experiment, the source and target tasks were learned to convergence, and this represents the best possible results for $d_Q$. The problem with this task similarity measure is that it requires the Q-values
Figure 6.2: $d_Q$ on the x axis vs $d_T$ on the y axis, after 300,000 steps. When the target task is partially learned, $d_Q$ provides a poor approximation to transfer time in the moving goal case.

Figure 6.3: $d_P$ on the x axis vs $d_T$ on the y axis in units of a thousand steps, the moving goal case.
Figure 6.4: $d_P$ on the x axis vs $d_T$ on the y axis in units of a thousand steps. When the target task is partially learned the distance measure provides a poor approximation to transfer time in the moving goal case.

Figure 6.5: $d_R$ on the x axis vs $d_T$ on the y axis in units of a thousand steps, the moving goal case.
Figure 6.6: $d_R$ on the x axis vs $d_T$ on the y axis in units of a thousand steps, the moving goal case. Partially learned.

to be known (or at least well approximated) in both the source and the target task. If the Q-values are not fully learned, $d_Q$ does not work well, as shown in Figure 6.2. Since the point of transfer is to aid in the learning of the Q-values in the target task, requiring fully learned Q-values in the target task is unrealistic.

The moving goal results for $d_P$ are very similar to $d_Q$ (see Figure 6.3). This measure also requires that the tasks are thoroughly learned before providing an accurate estimate of task similarity in the moving goal case (see Figure 6.4).

$d_R$ converges long before $d_Q$ or $d_P$. In general, rewards can be learned long before the correct Q-values (and the correct policies) can be propagated back through the state space. The problem with $d_R$ in this case, is that if there are two tasks in the library, one with a reward moved a small amount from the target location and another with the reward moved a large distance from the target location, both will appear to be equally similar to the target task. In our moving goal test all of the tasks in the
6.7. EVALUATION

Figure 6.7: $d_Q$ on the x axis vs $d_T$ on the y axis, computed with average reward, expanding obstacle case.

library are the same except for the location of the goal, which is the very feature that $d_R$ cannot differentiate (see Figure 6.6). Since $d_R$ does not perform in this case even when thoroughly learned, it is no surprise that it also failed when partially learned.

6.7.2 Expanding Obstacle Experiments

In the expanding obstacle case, $d_Q$, $d_P$, and $d_R$ were all good approximations to $d_T$ when the target task was thoroughly learned (see Figures 6.7, 6.9, and 6.11). When the target task was not thoroughly learned, $d_Q$ and $d_R$ were good approximations to $d_T$, while $d_P$ was not (see Figures 6.8, 6.10, and 6.12).

Note that $d_Q$ was a good approximation to $d_T$ before the Q-values in the target task were thoroughly learned in the expanding obstacle case, but failed to do so in the moving goal case. The difference between the moving goal result and expanding obstacle result can be easily explained by the following example. In the moving goal
Figure 6.8: \(d_Q\) on the x axis vs \(d_T\) on the y axis. When the target task is partially learned the distance measure provides a good approximation to transfer time in the expanding obstacle case.

Figure 6.9: \(d_P\) on the x axis vs \(d_T\) on the y axis, expanding obstacle case.
Figure 6.10: $d_P$ on the x axis vs $d_T$ on the y axis. When the target task is partially learned $d_P(i, k)$ cannot provide a good approximation to transfer time in the expanding obstacle case, unlike $d_Q(i, k)$.

Figure 6.11: $d_R$ on the x axis vs $d_T$ on the y axis, expanding obstacle case.
Figure 6.12: $d_R$ on the x axis vs $d_T$ on the y axis. When the target task is partially learned $d_R(i, k)$ can provide a good approximation to transfer time in the expanding obstacle case.

case, it often happens that there are some source tasks with a goal that is a small distance from the goal’s location in the target task, and other source tasks with the goal placed a large distance from the goal’s location in the target task. If the agent has only learned the Q-values close to the goals, then the distances computed will be the same regardless of the distance the goal was moved until the Q-values back up a sufficient distance. However, in the expanding obstacle case, the agent can quickly learn the Q-values going into the obstacle. Since the other Q-values will be nearly the same during the initial stages of learning, the Q-values going into the obstacle dominate in the distance computation and will provide an excellent approximation to the difference in the final Q-values that would eventually be learned by the agent. Thus, in this case, the initial approximation provided by $d_Q$ is approximately correct, while it is not correct in the moving goal case.

$d_P$’s behavior in the expanding obstacle case is very similar to its behavior in the
moving goal case when the tasks were thoroughly learned, except that in our complex grid world maze there are often several optimal policies that can lead to the goal. This caused the task similarity measure to be more noisy than in the moving goal case.

Unlike $d_Q$, $d_P$ was unable to provide a good approximation before the policy was thoroughly learned even in the expanding obstacle case. This makes sense because $d_P$ measures the overlap in the optimal policy. Although the agent quickly ruled out actions that took it into the obstacle, it had not yet learned the optimal policy, and this technique doesn’t take into account what it had learned about what not to do.

In the expanding obstacle case, $d_R$ was a good approximation to $d_T$. However, $d_R$ was not a good approximation to $d_t$ in the moving goal case. This is because in the expanding obstacle case the difference in the number of negative rewards going into the obstacle provided an excellent approximation to the transfer value of a given task. Thus this measure cannot capture the distance that a goal is moved, but can easily capture the fact that new goals or obstacles have been added to the problem. These results are summarized in Table 6.1.

### 6.7.3 Clustering

The cluster tree based upon $d_Q$ correctly separated out tasks that had the goal near the upper right from those tasks that had the goal near the bottom left. These
Figure 6.13: Cluster tree 1 and 2, The same cluster tree was created by both $d_Q$ and $d_P$.

Figure 6.14: Cluster tree 3, based on $d_R$. 
categories of tasks were then further broken down into tasks that had an obstacle from those tasks that had no obstacle (see Figure 6.13). This shows that $d_Q$ is more sensitive to the general policy of the task than it is to the presence of obstacles, yet it was able to detect the presence of the obstacles.

Like $d_Q$, $d_P$ is more sensitive to the general policy than it is to the presence or absence of obstacles in the task (see Figure 6.13).

When using $d_R$, the tasks were sensitive to the presence of the obstacle in the center, but were unable to capture the general policy trends (see Figure 6.14).

### 6.8 Task Similarity Discovery Conclusion

We have shown that no best measure for similarity between any two tasks in a MDP exists apart from the transfer mechanism employed. We have also proposed that task similarity could be defined as the average amount of speedup expected when using one source task to speed the learning of another target task under a given transfer technique. We proposed three properties that a task similarity measure should have.

Creating approximations to such a task similarity measure that can be evaluated before the actual target task is learned can help an agent use the information in its task library to adapt to new situations more quickly than learning from scratch. Such a measure is useful because an agent can use this measure to determine which tasks to use in transfer given a particular transfer technique. We have shown that different task similarity measures can capture different types of similarity. This could indicate that an agent should have more than one transfer mechanism, which could help the agent to determine not only which source task to use, but also which transfer technique to use given the type of similarity between the source and target tasks.

Different distance measures often capture different types of differences between
tasks. Both $d_Q$ and $d_P$ best capture trends in the overall policy but are only accurate after the Q-values or the policy have been learned. However, approximations to these values can be computed before the Q-values or policy have been thoroughly learned. The accuracy of the approximation will depend upon the type of task being learned, and the task similarity measure employed. In the moving goal case the task must be more thoroughly learned than in the expanding obstacle case for $d_Q$ to provide an accurate approximation. In both cases $d_P$ requires the task to be more thoroughly learned than does $d_Q$. $d_R$ can be computed before the policy has been learned, but is less sensitive to overall policy trends. In the moving goal case $d_R$ is not just less sensitive to over all policy trends, it is incapable of determining the difference between moving a goal a short distance vs. moving the goal a long distance.

6.9 Future Research

In this chapter we have shown that there are some situations in which no best similarity measure exists. However, it is our belief that this holds for most if not all situations. It may be possible for a task transfer technique to be created that would cause any two tasks appear arbitrarily similar. If this is the case, then a proof to the effect would be an important part of any future work in this direction, as this observation could have important psychological and philosophical implications. It would imply that the human tendency to view a pair of tasks as more or less similar than another pair of tasks is dependant on the brain’s techniques for using one problem to understand another. Under another set of techniques this order could be very different.

A heuristic with the ability to return a useful approximation of task similarity before the tasks are learned, like $d_R$, but which captures differences in policies like $d_Q$
or $d_P$ should be developed.

A possible Bayesian approach to task similarity measures would be to take what we currently believe about the probability functions $r_i(s, a)$, and $r_l(s, a)$ and compute some measure of similarity between the two functions.

An analogy based similarity measure would be extremely useful, however it is currently not clear how to build such a measure. The appropriate $d_T$ could be easily computed if an analogy based transfer technique could be developed, but it is unclear how such a transfer technique could be constructed.

Currently the source task(s) for transfer are hand-picked by the user. Once distance measures have been determined that can accurately approximate the task similarity before the target task is learned, the entire transfer process could be automated. Once the agent has determined that its current situation does not match any task in its library (through localization), it should then find the most similar task in its library, and apply transfer.

Eventually the system should compute multiple similarity measures for the tasks in the library and then automatically select a set of tasks and transfer techniques suited to the target task.
Chapter 7

Task Transfer

After an appropriate source task has been identified, the agent can utilize that task to bias the learning of the target task in a way that will improve the learning of the target task. This process is known as task transfer.

This chapter will explore some of the issues involved with task transfer in the model free case. We will show that task transfer is possible, but that the current techniques for task transfer have several problems, especially when the similarity between the source and target task is low. We will explore the reasons behind these problems and introduce several new techniques that overcome these problems and show empirically that they outperform the state of the art. Specifically, we will show that they degrade more gracefully as the target and source tasks become less similar.

7.1 Introduction to Task Transfer

Since some portions of the source task will be relevant to the target task and some portions will not be relevant, task transfer requires the agent to determine which features from the source task to apply to the target task and which to ignore (see
Figure 7.1). Appropriate task transfer is a complex problem and lies at the very heart of life-long learning.

![Figure 7.1: A source task can bias the learning of a target task.](image)

There are several situations that can be encountered in task transfer. If the agent’s actuators and sensors remain constant, then either the environment has changed, or the reward structure has changed, or both [35]. If the environment has changed, the state transition probabilities will no longer be the same. Alternately if the reward structure has changed, the state transition probabilities will be the same but the reward received for taking those transitions will have changed (the MGMDP case). Combinations of these two situations can also be encountered. The more complex case occurs when the agent itself changes. In this case, the state space can grow or contract (if preceptors are added or removed), or the agent’s action space can grow or contract (if actuators are added or removed).

Task transfer algorithms must attempt to use information from the source task, when such information is useful in the context of the target task, while simultaneously allowing specialization in those cases when the tasks differ. This problem is similar to the problem of overfitting and generalization encountered in classification problems, but there are several differences. For example, if a classification algorithm performs
well on its training data but poorly with a wider set of data, we say that it has “overfit” the training data. In the case of task transfer, we want the agents to “overfit” on each of their individual tasks, and we do not want them to learn a policy that is best for all of the tasks, because such a policy would likely be meaningless and inappropriate for any one of the tasks. Thus, we want to allow specialization on the individual tasks, while still being able to gather information that will be useful in similar situations. This use of information in a related situation is not generalization in the standard sense; although, if the tasks could be adequately and completely parameterized then generalization techniques could be used to find a policy for a new task given its parameterization. Unfortunately, such a parameterization is difficult to formulate, especially when such a parameterization would, by necessity, include information that the agent may not have access to until after the task is learned, such as transition probabilities and reward functions.

7.2 Transfer Techniques

Many methods have been proposed for task transfer in a reinforcement learning context. Some methods are model-based and focus on transferring action models and other model information from one task to another. Other transfer mechanisms are model-free. We will focus our research on the model-free techniques. Some of the past model free techniques are:

- Direct transfer [3][2][25][6][14], which uses the learned Q-values from the source task to initialize the Q-values of the target task
- Fixed sub-transfer [3][2], which is a piecewise transfer mechanism that uses a portion of the source policy to initialize the target task, and then fixes those
values.

In this chapter we develop and compare the following new transfer techniques:

- **Soft transfer** \([25][6]\), which uses a weighted average of direct transfer and some fixed value (as in learning from scratch) to initialize the Q-values in the target task to make the transferred policy more adaptable.

- **Memory-guided exploration** \([25][6]\), which uses the past task to guide the initial exploration of the agent in the target task.

- **Dynamic sub-transfer** [5] which is a piecewise transfer mechanism that uses a portion of the source policy in the target task without fixing the values of the sub portion that was transferred.

Of these algorithms, memory-guided exploration and dynamic sub-transfer were developed as part of this thesis, while soft transfer was developed primarily by Nancy Owens, but with some participation from myself [6].

We will now discuss each of these transfer mechanisms in more detail:

### 7.2.1 Direct Transfer

Direct transfer of Q-values is the most straightforward method of performing task transfer in Q-learning. Q-values are typically initialized to some constant value, \(Q(s, a) = I\). Rather than using a fixed value for \(I\), direct transfer takes the final learned Q-values from the source task and initializes the target task with the Q-values from the earlier source task.

\[
\forall s, \forall a, I_{target}(s, a) = Q_{source}(s, a).
\]

In effect, the Q-values from the source task are used to initialize the target task, and then normal updates are allowed to adjust to any differences between the two tasks.
7.2. TRANSFER TECHNIQUES

Direct Transfer Problems

In some situations, direct transfer can perform poorly. It has been previously shown that when tasks are sufficiently dissimilar, direct transfer of Q-values can be much worse than learning from scratch [3]. We believe that there are two main reasons for this which we call the “unlearning” problem and the “information loss” problem.

The “unlearning” problem is that it often takes longer to unlearn the incorrect portion of the prior policy than it would take to learn the entire policy from scratch. The “information loss” problem is the loss of correct information during transfer. Until the Q-values converge to their final state, valuable information can be lost. Even in the similar portion of the Q-space where correct information was transferred, incorrect updates from the section which is not similar can mar this correctly transferred information. This means that the agent sometimes loses all the relevant transferred information, while attempting to unlearn the irrelevant portion.

These two problems can be best seen with an example from the simple decision task from Section 4.1. Once the source task is thoroughly learned, a target task is generated by moving the goal location one state to the right or left of its location in the source task. Thus there is considerable policy overlap between the source and the target task, and we would, therefore, expect direct transfer to perform well on this task. However, this is not the case. Direct transfer consistently takes slightly longer in this task than learning from scratch because of the unlearning and information loss problems.

If the agent trains on the source task until all of the Q-values reach their optimal values, all of the Q-values in the whole tree will be zero, except for those on the branch leading towards the goal. For example consider Figure 7.2. In this simple decision problem, when the reward is moved just one position to the right of its position in
Figure 7.2: The bottom level of a larger case of the simple decision task. Many iterations are required to generate the correct policy.

Figure 7.3: The bottom level of a larger case of the simple decision task. Only one iteration is required to generate the correct policy.

the source task, the agent has a very difficult re-learning task. Before the agent’s policy will be correct, the agent must drop the left Q-value of .9 down, and the right value Q-value of 0 up until their values cross. This can take some time if the learning rate is low. This illustrates the unlearning problem. This is in comparison to the decision problem shown in Figure 7.3 where only one iteration is required to find the optimal policy, (the policy can be correct even if the Q-values are not yet at their optimal values, as long as their relative values are correct). The agent will find the correct policy in one iteration even if the agent makes the wrong choice. This is in comparison to the many iterations required in the previous example. The unlearning problem can best be overcome with a higher learning rate, but such a higher rate
7.2. TRANSFER TECHNIQUES

Figure 7.4: Illustrates the unlearning and information loss problem in a more complex case.

contributes to the information loss problem which we will now discuss in more detail.

Consider the example from Figure 7.4. If a learning rate of 1 is used, then during the first episode of training the agent (incorrectly) assumes that the goal will be exactly where it was before. When the agent reaches the old goal location the agent receives a reward of 0, and updates the Q-value to be zero. At this point, all of the Q-values leading towards leaf nodes are zero. The agent, is therefore forced to choose randomly. Even if the agent chooses correctly, a 0 will be backed up the tree to the
next level. This 0 will then be propagated backwards each successive episode until the value of the root node is also updated to zero. The agent is then required to do a blind search for the reward. Although all the branches leading to the goal but the last step were correct, all this correct information was lost during training before it could be utilized. This illustrates the information loss problem. Of course this is an extreme example, and the information loss problem (although present) is less noticeable in environments like the grid world and complex grid world for reasons which we will discuss later.

We might assume that the learning rate or the discount factor is the problem here. To reduce the information loss problem, it is tempting to lower the learning rate, however, that would contribute to the unlearning problem. Thus, the learning rate must balance the agent’s need to unlearn incorrect old information, while preserving old information which was correct. Experimentally, the best performance was achieved in this world with a learning rate near 0.5. Changing the discount factor had little effect [25].

To help unlearn old incorrect information, some have tried an algorithm that propagates the change in Q-values more quickly, (similar to prioritized sweeping [22]) known as “modified prioritized sweeping.” Modified prioritized sweeping doesn’t use a model of the environment, it simply keeps a history of the past \( n \) updated nodes, which will be the state’s predecessors, and propagates updates back to these nodes. Modified prioritized sweeping decreased the overall search time for both shaped and traditional reinforcement learning, but did not decrease the shaping time in comparison to the original learning time [25].

Several alternate exploration strategies [34] including recency-based, counter-based, and error-based exploration have all been tried [25]. None of these methods worked in conjunction with direct transfer of Q-values for the same two reasons: First, if
the learning rate is too high, correct information is overwritten as new Q-values are updated. Second, if the learning rate is low enough to prevent the overwriting of good information, it takes too long to unlearn the incorrect portion of the previously learned policy.

### 7.2.2 Soft Transfer

Soft Transfer attempts to minimize the unlearning problem, and thereby make direct transfer more robust to differences between the source and target tasks. Soft transfer preserves the current policy from the source task while “softening” the Q-values, making them easier to change. This is accomplished by initializing the target’s Q-values using a weighted average between the source Q-values and the standard tabular initialization value $I$ in the following manner:

$$Q_{target}^o(s, a) = (1 - W)I + W(Q_{source}^F(s, a)),$$

where $Q_o$ is the initial Q-value, and $Q_F$ is the final Q-value.

If $W = 1$, then soft transfer is equivalent to direct transfer and if $W = 0$, then the agent learns from scratch. By picking $0 < W < 1$ the agent can unlearn the incorrect portions of its policy easier. Care needs to be taken to set this parameter appropriately. If $W$ is too high, the agent will spend too much time unlearning the incorrect information from the past task, and if it is too low the agent will lose useful information before the new information is learned.

### 7.2.3 Memory-guided Exploration

Memory-guided exploration stores two or more sets of Q-values: $Q_{target}^o(s, a)$ and $Q_{source}^o(s, a)$. One set of Q-values represents the agent’s experience in the target
task while the other set of Q-values represents the stored memory of the source task. The source Q-values are never changed, preventing accidental loss of pertinent information, thus preventing the “information loss” problem. The new Q-values are updated normally as the agent explores the environment, thus reducing the time necessary to “unlearn” the incorrect portions of the policy.

In memory-guided exploration, actions are chosen based on a weighted sum of the source Q-values, the target Q-values, and some random noise according to:

$$a_t = \max_a \left[ W Q^{source}(s,a) + (1 - W) Q^{target}(s,a) + \tau \right],$$

where $W$ (the weight given the information from our source) and $\tau$ (the random noise) both decay over time. Exploration is generated by the $\tau$ term. The source Q-values are only used to guide the agent’s initial exploration, since $W$ decays over time. Memory-guided exploration is an off-policy controller [33], and this raises convergence issues. These issues can be overcome, and they will be discussed in detail below. In short, once $W$ decays sufficiently, all the standard convergence proofs hold, thus this method is guaranteed to eventually converge.

A process similar to memory-guided exploration was independently developed by Dixon et al. [14]. Although his technique is similar to Memory-guided Exploration, they used a probabilistic switching mechanism to choose between the agents prior task and the current task rather than our weighted average. Furthermore, they did not adequately deal with the convergence issues involved as discussed below.

**Convergence Issues**

The off-policy nature of memory-guided exploration causes three specific convergence problems: 1) required exploration bias, 2) unvisited transitions, and 3) local exploration vs. global exploitation. Although these issues were not present in the Simple
Decision Task, they needed to be dealt with before results could be obtained in any of the grid worlds.

**Required Exploration Bias**

Memory-guided exploration requires an additional bias towards unexplored areas. In Memory-guided Exploration, $W$ decays asymptotically toward 0. The purpose of decaying $W$ is to allow the agent to explore additional areas of the state space. However, the agent will still have some bias towards the previous policy because $W$ will not fall all the way to zero. In order to insure sufficient exploration as $W$ decays, some additional weaker bias toward the unexplored areas of the state space must dominate, otherwise the agent will never leave the location of the previous goal.

This problem is best illustrated when $I = 0$, and can be easily solved by choosing $I > 0$. This biases the agent’s exploration away from areas where he has already been because the Q-values will be discounted each time they are visited. If $I$ is chosen to be 0 this cannot happen because $0 \times \gamma = 0$. Alternately, any other type of exploration bias could be employed, such as recency or counter-based exploration [34]. Care needs to be taken to ensure that this additional bias is initially smaller than the bias towards the prior policy.

**Unvisited Transitions**

Off-policy controllers can diverge from the true Q-values, unless the controller is a random Monte Carlo controller [33]. In Memory-guided Exploration this divergence can happen because the prior knowledge bias causes the agent to visit the states according to a different distribution than would the policy that maximizes the current reward function [14]. Effectively, certain transitions never get updated, which then may be backpropogated by the max operator to other neighboring states.

As an example, the memory biases the agent away from obstacles. This behavior
is one of the major advantages of this method. We wanted the agent’s memory to be strong enough to insure that the agent would not explore transitions that formerly mapped to obstacles until it had first explored all other options. Because the transitions mapping to obstacles are never visited, the Q-values mapping to obstacles will remain at \( I \) and the Q-values surrounding obstacles can therefore not drop below \( I \times \gamma^d \) where \( d \) is the distance from the obstacle. This means that when these un-updated Q-values are the maximum for that state, they backpropogate to all the neighboring states. This effect remains until \( W \) drops sufficiently to allow the previously unvisited states to be visited. Effectively this creates an artificial bias towards states next to obstacles because transitions into those obstacles are not taken because of the memory bias.

It was not immediately clear whether these issues would affect memory-guided exploration because the agent’s memory of old tasks only affects the agent’s initial exploration. Although convergence is assured once \( W \) decays to insignificance (as has been previously proven by Dixon in [14]), we found that the agent’s behavior was often divergent until the decay of \( W \), at which time the agent would behave like an agent learning from scratch. Although the agent eventually converged, the advantages of the transfer were nullified.

There are two solutions to this problem. The first is to allow the old Q-values to bias the update of the new Q-values by the same amount that the old Q-values affect exploration. The update equation becomes:

\[
\Delta Q(s_t, a_t) = \alpha(R(s_t, a_t) + \gamma \max_a(Q^C(s_{t+1}, a_t))),
\]

where \( Q^C = W \times Q^{source}(s_{t+1}, a) + (1 - W)Q^{target}(s_{t+1}, a) \). In effect this creates a new “composite” on-policy controller.

The other option is to keep local values for \( W \). This is effective because divergent
behavior manifests itself as repeated visits to the same few states in an infinite loop. Because $W$ decays each time the agent visits a state, the local values of $W$ can be decayed in areas of divergence while remaining high for the rest of the state space. This allows the agent to behave like an on-policy controller in areas of divergence, while allowing the agent the exploration advantages of behaving as an off-policy controller in areas where divergence is not an issue.

**Local Exploration vs. Global Exploitation**

There is another reason to keep $W$ local. The max Q-values in the source task will approximate $\gamma^d$ where $d$ is the shortest path to the old goal from a given state. This means that in areas of the state space that are far from the old goal, the difference between the Q-values of the best choice and the worst choice is often very small, whereas these differences are comparatively high near the old goal. Therefore the agent’s memory bias is stronger near the old goal than it is in states further away.

If $W$ is not local, the agent will stay near the old goal until the global $W$ has decayed sufficiently for the agent’s exploration bias to overcome the agent’s strong memory bias at the old goal location. Because the agent’s memory bias is weaker near the start, when the agent is moved to the start it will begin learning tabula rasa. Thus the agent has lost any exploratory advantage that could have been drawn from its memory bias in the rest of the state space.

Keeping $W$ local to each state solves this problem. When $W$ is local, the agent moves to the location of the source goal, and stays there until the local $W$ decays sufficiently. Once the values of $W$ near the old goal have decayed, the agent begins to explore near the location of the source goal, while $W$ remains high in other areas. This allows the agent to exploit the old policy in the other areas of its state space while exploring in areas where $W$ has decayed.
Thus the convergence problems inherent in memory-guided exploration can be alleviated by simply keeping $W$ local, and by slowly decaying it to 0. Since all the $W$’s will eventually decay to 0, and since the algorithm behaves like regular Q-learning once they do, convergence is guaranteed at that point.

### 7.2.4 Fixed Sub-transfer

Another model-free method for task transfer we call fixed sub-transfer. This method was introduced by Bowling and Veloso [2][3] and attempts to overcome the difficulties inherent in direct transfer. This method grew out of the SKILLS algorithm, which was introduced by Thrun and Schwartz in 1995 [38].

The SKILLS algorithm attempts to find partially defined action policies called skills which occur in more than one task. Skills are found using a description length argument. The SKILLS algorithm minimizes a function of the form

$$E = PL + \mu * DL,$$

where $PL$ is a performance loss, and $DL$ is a description length. By minimizing $E$ this algorithm effects a piecewise decomposition across multiple tasks. The skills so identified are the skills that are applicable across many separate tasks. It would be reasonable to use these partially defined policies over multiple tasks.

In fixed sub-transfer, a portion of the source task is used in the target task. The portions that are used by the target task are those that have been deemed “similar.” The source task and the target task share the same Q-values in the similar section, and the Q-values are not updated in that section (they are fixed). In the portion that is not deemed similar, the agent initializes its Q-values to some $I$ and learns these sections from scratch[2]. Bowling and Veloso do not give a name to this technique, but we call it fixed sub-transfer because only a portion of the source task is used by
7.2. TRANSFER TECHNIQUES

the target task and the Q-values in the similar portion are fixed.

There are several advantages to this approach aside from a reduced description length. This method avoids the unlearning problem because the sections that are not similar are not transferred. The information loss problem is also avoided because the correct portions of the policy are fixed.

There are also several drawbacks to this approach:

- Some method is required to determine which sub-skill(s) should be used by the target task. This usually requires “design intervention” (or manual intervention by the human user of the system [25]) because until the new task is learned it is difficult to determine its similarity with the skills which the agent already knows.

- The fixed sub-portion must be chosen correctly, and if chosen incorrectly can render the problem un-learnable.

- Any technique that fixes portions of the Q-space is prone to divergence along the boundary between the fixed portions of the policy and the unfixed portion as discussed below.

- Fixing portions of the policy often leads to a sub-optimal end policy. The sub-optimality of such a solution has been quantitatively bounded [2][3].

The design intervention necessary in fixed sub-transfer is problematic, but is not insurmountable, especially if the task has been broken down into sub-tasks by the SKILLS algorithm. It should be possible to notice from simple observation which of the automatically extracted skills are still valid in the new context. An example where this can be done, previously discussed by Bowling and Veloso [2], is robot soccer. In robot soccer a user might notice that a policy learned in a simulator performs well in
the real world when the agent is not shooting the ball and is at a distance from the
wall, but performs poorly otherwise. The sections away from the ball and the wall
can therefore be deemed “similar.”

The convergence issues are more problematical. When we implemented Bowling
and Veloso’s algorithm we found that the agent diverged along the boundary between
the fixed and unfixed portions of the state space. This is because the fixed portion can
often preserve a slightly higher expected discounted value than the states which are
now closer to the moved goal (see Figure 7.5). This causes the agent to transition into
the fixed portion, and then move back in an infinite loop. Because the incorrect values
are fixed, this problem will never resolve itself. Bowling and Veloso never mention
how they dealt with this divergence. We chose the simple solution of updating all
transitions that cross back into the fixed portion with an expected discounted reward
of 0. This “quick fix” would not work in all situations and these divergence issues are
a major drawback of fixed sub-transfer. These problems need to be more thoroughly
addressed before this method could be used extensively in real world applications. As
will be shown, dynamic sub-transfer elegantly sidesteps these divergence issues.

7.2.5 Dynamic Sub-transfer

In dynamic sub-transfer, the agent transfers information in the portions of the state
space that are considered similar while initializing the rest of the state space to a
constant $I$ just as in fixed sub-transfer. However, the portions that are transferred
are not fixed as in fixed sub-transfer, but allowed to adjust normally.

As in fixed sub-transfer, by only transferring information in areas deemed “simi-
lar,” the unlearning problem is avoided. By not fixing the portions that were trans-
ferred, this method creates the possibility of the information loss problem which was
avoided in fixed sub-transfer. We hypothesize that the danger presented by the information loss problem was less than the danger presented by the convergence and sub-optimality problems inherent in fixing the policy. Further, the information loss problem can be reduced by changing the learning rate. Normally such a change would exacerbate the unlearning problem, however, through our selective initialization this side-effect could be avoided. We therefore hypothesize that the dynamic sub-transfer method would retain most of the learning-rate improvements seen with fixed sub-transfer while ensuring convergence, and removing any sub-optimality in the final solution. These hypotheses will be tested in section 7.5.3.

Although the dynamic sub-transfer algorithm specifically deals with the problems of divergence and sub-optimality associated with fixed sub-transfer, this approach does not deal with the necessity of design intervention inherent in all sub-transfer methods. The user must still manually decide which portions of the state space to consider similar, although such automation may be possible in the future using the SKILLS algorithm.

7.3 Multiple Task Transfer

A full analysis of multiple source transfer is beyond the scope of this thesis. However, because the thesis is designed to lay the groundwork for an eventual system that would incorporate multiple task transfer, it is important to show that multiple source transfer is at least feasible. We have done this by showing that a naïve implementation can have some success in a limited case.

The actions of the learning agent with multiple source tasks are based upon an amalgamation of advice from the past tasks as well as the advice that the agent derives from its actual experience in the world. This can be thought of as a type of
There are several voting schemes that could be used in order to construct this social welfare function. One solution is to take the average of the advice of all the source tasks, and then use those values to initialize the Q-values for the new task. The agent could then update those values with its own experience (in the single source domain this is known as “direct transfer”). This method has the advantage of allowing the target’s own experience to eventually take a greater role as time progresses, while the information from the source tasks take a greater role at first. Unfortunately, averaging policies can be dangerous, especially when the tasks being averaged are not very similar. For example, if one policy is to go right and another policy is to go left, the average of the two may be neither right nor left, and may generate a policy that is not good for either task. Policy averaging is especially problematic the less similar the source tasks are to each other and to the target task.

Another method of forming this social welfare function is to take only the invariants among the source tasks, and transfer those invariants as in direct transfer, while initializing the rest of the state space normally. Theoretically this will allow an agent to learn the task normally except in regions where all his previous sources agree. As before, this technique will be more effective the more related the source tasks are to each other and to the target task. The more you can narrow down the number of source tasks, the more invariants there will be to transfer.
7.4 Comparison of Single Source Non-Sub-Transfer Techniques

7.4.1 Simple Decision World Results

This task was mostly used for analysis purposes as discussed above, and we will not discuss its performance in detail except to say that memory-guided exploration performed much better than direct transfer as discussed by Peterson et. al. [25]. It should be noted that direct transfer performed especially poorly in this world due to the fact that all transitions that did not terminate in a goal state converged to a Q-value of 0. This is not the case in the other worlds we tested, where the ability to backtrack insured that Q-values were high in areas near the goal even along paths that did not directly lead to the goal. This property exacerbated the unlearning problem in the simple decision world. This is an important observation. This suggests that worlds, such as chess, which are like the simple decision world in that moves cannot be undone, will especially suffer from the unlearning problem.

7.4.2 Grid World Results

Figure 7.8 demonstrates the value of direct transfer, showing that a task can be learned much faster using transfer from a related task while demonstrating the potential damage that can be produced if transfer is indiscriminately used from an unrelated task. These results were generated in the deterministic version of the simple grid world. For our related task we moved the goal one step from its initial position, creating two tasks that were as similar as possible without being identical. Interestingly, because the source task had been thoroughly explored, its application in transfer prevented the convergence to a sub-optimal policy that is so common when
learning from scratch with a “choose best” exploration policy. For our unrelated tasks we swapped the start and the goal, generating two tasks that were as dissimilar as possible. Notice that the situation is actually worse than it at first appears since the scale of this graph is logarithmic. In unrelated transfer, more time was spent in the first trial while unlearning the incorrect portion of the task than in all the rest of the trials combined. This result motivated the exploration of other transfer techniques and various similarity measures.

To see how the various techniques compare as the similarity slowly deteriorates, we used the deterministic grid world without any obstacles. The agent was trained with the goal in the upper left. This was used for the source task, and target tasks were generated by moving the goal different amounts to the right. Larger distances generated a target task that was less similar to the source task.

Local values of W were used with memory-guided exploration, and for soft transfer we chose a value .5 for W. All three methods of task transfer were superior to learning the task from scratch until the tasks became sufficiently dissimilar (see Figure 7.9). This is in contrast to the results in the decision world, where direct transfer was worse even when the tasks were similar. In the grid world, with the increasing dissimilarity of the tasks, direct transfer performed the worst, memory-guided exploration performed better, and soft transfer performed best.

7.4.3 Nomad II Robot Results

In the Nomad II simulator we found that memory-guided exploration performed better than learning the task from scratch (see Figure 7.10). Interestingly, soft transfer and direct transfer were worse than learning the task from scratch in this environment. These results can vary somewhat from run to run. We believe that this was
due to differences in the wall avoidance policy learned in the previous task. There
are many effective policies for wall avoidance that could be learned. One policy for
wall avoidance might transfer to wall following better than another, and different
mechanisms for task transfer may transfer one policy better than another.

7.4.4 Analysis

Memory guided exploration performed better than tabula rasa learning in almost
every situation tested. Memory guided exploration was also superior to the direct
transfer of Q-values. More research is needed to determine when soft transfer performs
better.

7.5 Comparison of Sub-transfer Techniques

To visualize how these transfer methods function, we chose a variant of the simple
grid world from Section 4.2 with three rooms, as first introduced by Sebastian Thrun
[38] and used by Bowling and Veloso [2][3] (see Figure 7.11). This world has multiple
positions for the start and the goal. We first learned the task with the start and
the goal in the bottom positions and then moved the start and the goal to their
counterpart positions toward the top.

The policies learned when the start and the goal are moved are nearly identical
in the first room. In the second room, the two policies are similar, and in the third
room, the policies are different. This synthetic environment models what happens
in transfer when a portion of a policy changes while another portion of the policy
remains the same, the exact situation where the sub-transfer techniques are useful.
7.5.1 Direct Transfer

Direct transfer is not a sub-transfer technique, however it provides a baseline against which the other techniques should be judged. Under some conditions direct transfer can perform worse than learning from scratch. Bowling and Veloso reported that in the simple three room grid world, direct transfer did indeed perform worse than learning from scratch \[2\]. We felt that direct transfer should perform better than learning from scratch in the three room simple decision task because the difference between the two policies was not great. We found that direct transfer did indeed outperform learning from scratch (see Figure 7.12) and that Boling and Veloso’s experiments in this case were not reproducible.

7.5.2 Fixed Sub-transfer

We compared our results for learning from scratch and direct transfer to the results obtained from fixed sub-transfer by fixing the policy in the left room only, and then by fixing the left two rooms (See Figure 7.13). These results were similar to those obtained by Bowling and Veloso who performed the same experiment \[2\]. Notice that when the first room is fixed there is a substantial improvement in the learning rate. When the second room is fixed, this improvement is even greater but the policy learned is sub-optimal.

In fixed sub-transfer the majority of the speedup in the learning rate comes from the initialization. Direct transfer of Q-values is effectively performed on the portions of the policy that are similar, while the rest of the Q-space is initialized to a fixed \(I\). This removes the difficulty inherent in the unlearning problem discussed in Section 7.2.1. The fact that the similar portions of the state space are fixed has less to do with the speedup, as information loss is not as large a problem as unlearning. These
observations lead us to dynamic sub-transfer.

### 7.5.3 Dynamic Sub-transfer

We tested dynamic sub-transfer and found that it had a learning rate comparable to those obtained through the fixed sub-transfer strategy in the three-room problem (See Figure 7.14). As can be seen from the graph, dynamic sub-transfer only lagged a few time steps behind fixed sub-transfer through the information loss problem. Thus the information loss problem was not a significant problem in this case as we hypothesized in Section 7.2.5. However, the agent achieved an optimal policy over time. Thus, in the three room problem, at least, dynamic sub-transfer is preferable.

### 7.6 Multiple Task Transfer Results

We used the simple grid world (see Section 4.2) in order to rapidly create a set of tasks that could be easily engineered to produce various levels of similarity. We produced a suite of thoroughly learned tasks in this world. Moving the goal various amounts, removing the obstacle in the center, or swapping the start and the goal produced various levels of similarity. There were four main types of tasks: tasks with the goal within four squares of the upper right, tasks with the goal within four squares of the bottom left, and tasks with and without the central obstacle.

Because policy averaging is known to have problems when the tasks averaged do not share sufficient similarity, we first applied the clustering techniques from Chapter 6. The hope was that the clustering algorithm would group these similar tasks together. Various clusters were produced, which were then treated as secondary indexes into the list of learned tasks. Multiple cluster trees were produced using multiple distance measures. One task was withheld from the clustering algorithm to use as the
We used the $D_Q$ and $D_R$ distance measures, and generated the cluster trees represented by Figures 7.15 and 7.16 which we have already seen from Chapter 6.

Once these clusters were produced, we tested transfer using both invariants and average Q-values from various clusters. The averaging technique is basically a modified form of direct transfer, where the policy directly transferred is an average of a cluster of tasks. The invariant transfer technique is basically dynamic-sub transfer, where the portion that was transferred was the portion deemed “invariant” in a cluster of tasks.

We found that the invariants from the top of the cluster tree built from the Q-values didn’t generate any useful information because there were no invariants at this level. There were invariants in the cluster tree based upon the R-values, even at the top of the cluster tree where all the tasks were included. This was because the bounding wall was invariant among all tasks, and the R-values easily captured this feature, while the Q-values did not. When invariant information was transferred from related branches of the Q-value cluster tree a substantial speedup in learning rate was detected (see Figure 7.17). Transferring information from the R-values substantially increased the agent’s fault tolerance, reducing the number of negative rewards encountered during training. If the proper sub-branch of the R-value cluster tree was used, the number of negative rewards received by the agent was reduced to 0.

In addition to invariant transfer, we also tested the use of the average Q-value from the appropriate branch of the tree built from the Q-values for the initialization of a target task. This gave a substantial increase in learning rate [shown in 7.17]. We also attempted transfer from the top of the Q-value tree (effectively using all tasks as if they had not been clustered). As expected, it was determined that the average
Q-values of all tasks generated a nonsensical initial policy which was detrimental to learning.

7.7 Task Transfer Conclusions

We have shown that task transfer, the most central element of our task library system, can be performed. There are several techniques for task transfer. We have shown that in general, related transfer can be extremely helpful, while unrelated transfer can be detrimental. We have shown that direct transfer is extremely sensitive to the level of similarity between the source and target tasks. We have analyzed the reasons for this sensitivity and have introduced several novel task transfer techniques, including memory-guided exploration, soft transfer, and dynamic sub-transfer. We have empirically shown that these techniques can increase the tolerance for dis-similarity in task transfer.

In cases where the designer knows which portions of the policy are similar beforehand, the sub-transfer methods are preferable to direct transfer, soft transfer, or memory guided exploration, because the agent can use the information from the user to know which parts of the state space to keep (which are similar) and which parts to discard (which are dis-similar). The effectiveness of this will depend on the effectiveness of the information given by the human user. The non-sub-transfer methods are preferable when such design intervention needs to be avoided.

Of the sub-transfer techniques, our results indicate that dynamic sub-transfer is empirically preferable to fixed sub-transfer in many situations, because it learns almost as quickly, but doesn’t converge to a sub-optimal policy, and it avoids the convergence issues found in fixed sub-transfer.

Chapter 6 showed that reinforcement learning tasks can be effectively clustered.
This chapter showed that the transfer of invariants or Q-value averages in such a cluster can be beneficial if the source cluster is sufficiently related to the target task. Thus we have shown that multi-task transfer is possible, however more work remains to be done in this domain.

This research has potential in the production of a truly autonomous agent that can learn many tasks over its lifetime. Such an agent could store these tasks in a task library, and work on clustering these tasks offline. Then when a new task must be learned, the agent could attempt to use information from these past tasks online in order to learn the new task faster.

### 7.8 Task Transfer Future Work

This chapter has focused on model free methods. Model based methods should work well with task transfer and should be explored further. Additional methods for soft transfer and memory-guided exploration could be explored such as initializing $W$ such that the values of $W$ closer to the old goal are low, while values of $W$ closer to the start are high. Also, it may not be necessary to keep a unique value for $W$ in each state, a local function approximator may be sufficient. Memory-guided exploration could also serve as a method for creating an advice taking agent, where the agent’s initial exploration is handled through direct control. Methods that use multiple past tasks should be explored, and memory-guided exploration should be expanded to handle multiple past tasks.

In dynamic sub-transfer, the amount of time wasted by information loss was small for the problems that we tested. However, this time could possibly have been recovered by temporarily fixing the similar portion, and then releasing it to allow convergence to an optimal policy. This technique should eventually be tested.
Another area of research that should be explored is piecewise transfer, where one source task may be used in a portion of the target task, while another source task may be used in another portion of the target task [38]. Related to this idea is the notion that tasks may have one thing in common with one cluster of tasks, while having something else in common with another cluster, perhaps even with another cluster generated using a different distance metric. It may be possible to combine this information from different source clusters in order to rapidly learn a new task.

Methods for automatically determining sub-problem similarity need to be developed, and hybrids of the above methods should be explored. In worlds where information loss is a serious issue, portions of the policy should be fixed until the agent has learned the rest of the state space to avoid information loss. These portions could then be released to allow the agent to adjust, thereby avoiding sub-optimality while retaining a fast learning rate. This process could perhaps be automated by watching the change in average reward, and releasing the fixed portions when it levels out.

A better method for dealing with the divergence issues involved with fixed sub-transfer needs to be developed. This is important because the current methods only function in select, carefully controlled situations.

Once an adequate feel for the various strengths and weaknesses of these methods has been reached and their effects have been quantified, we believe that the future course of this research should be to automate the entire transfer mechanism. When an agent encounters a new situation it should automatically determine that a new task is necessary. Then the agent should automatically choose the best transfer mechanism for the current situation, and adapt that mechanism as more information becomes available. Any such agent should incorporate the model-based approaches to task transfer as well as the model-free approaches.

We eventually hope to automate the process of selecting an appropriate cluster
from which to transfer. Determining the relatedness between a target task and a
cluster of source tasks should be reducible to a simple localization problem. This
problem should be easy to solve in the case of the R-values, while solving this problem
with Q-values may be more difficult because the Q-values will not approach their
correct values until the entire task has been learned. This problem should be a major
focus of our future research. By combining such a tool with the research described in
this paper the entire task library process could perhaps be automated. If this is the
case it could have extremely significant ramifications for such techniques as shaping.
Figure 7.5: This simple situation illustrates the convergence problem encountered when fixing a portion of the state space, even when that portion is correct. The dark arrow illustrates a fixed transition. The first arc is fixed, and the others are initialized to .5. Then the agent begins to learn. By step 4 of the learning process the agent is caught in an infinite loop. If randomness is introduced into the agent’s exploration, it can break out of this infinite loop; but if the goal is further away than in this simple example, the agent can create a false attraction to the boundary of the fixed portion of the state space that can be too large to break out of with any exploration strategy other than with a purely random strategy.
Figure 7.6: This shows how fixing the policy in the first two rooms can generate a suboptimal solution when the goal is moved from $G$ to $G^2$.

Figure 7.7: The actions of the target agent can be thought of as a social welfare function compiled from the advice of many source tasks and the target’s own experience in the world.
Figure 7.8: Shows the advantage of transfer if the two tasks are related, and the danger in transfer if the two tasks are unrelated. The left axis represents the average number of steps to the goal, while the bottom axis represents each epoch.
Figure 7.9: The average of 150 runs. The y-axis represents the steps to find goal 150 times, and the x-axis represents the distance that the goal was moved for all three transfer versions and learning from scratch in the deterministic version of the grid world.
Figure 7.10: Average of 3 runs each, for learning from scratch, direct transfer, soft transfer and memory guided exploration on the Nomad II simulator transferring wall avoidance to wall following.
CHAPTER 7. TASK TRANSFER

Figure 7.11: Simple three room stochastic environment.

Figure 7.12: Example run comparing direct transfer with learning from scratch.
7.8. TASK TRANSFER FUTURE WORK

Figure 7.13: Example run of fixed Sub-transfer when fixing the first two rooms and fixing the first room only, as compared to learning from scratch.
Figure 7.14: Example run of fixed sub-transfer when fixing the first room, and first two rooms vs. dynamic sub-transfer and learning from scratch.
Figure 7.15: Cluster tree 1, based on the mean squared error of the Q-values.

Figure 7.16: Cluster tree 3, based on the mean squared error of the R-values.
Figure 7.17: Steps to the goal vs. iterations. Transferring information from the clustered tasks rather than from all tasks avoided the disadvantages found in unrelated transfer, while allowing most of the speedup from related transfer.
Chapter 8

Task Library Conclusions and Future Work

Task Localization, Task Similarity Discovery, Task Clustering, and Task Transfer are the first steps toward the creation of a reinforcement learning task library system. Because this thesis presented several distinct algorithms, most information on conclusions and future work has been placed in the chapter in which the algorithm was discussed. However some things remain to be said concerning the library system as a whole.

8.1 Conclusions

We have shown that task localization can be accomplished through our Bayesian Task Localization Algorithm. This allows an agent to determine whether a current situation matches a past situation in its library. We have shown that various task similarity measures exist, and that there is no best task similarity measure. We have shown that different measures capture different types of similarity, and compared the
CHAPTER 8. TASK LIBRARY CONCLUSIONS AND FUTURE WORK

types of similarity captured by our four proposed similarity measures. Finally we have shown that transfer from similar tasks can be helpful, while transfer from tasks that are not similar can be detrimental. We have compared several techniques for task transfer, specifically we have focused on techniques that perform better as similarity degrades.

Although we have shown how each of these elements can be accomplished, it remains to put the entire task library system together. Although this is beyond the scope of this thesis, the entire process should eventually be tested. A task library should be compiled, localization should be performed, and, if localization fails, similarity discovery and clustering should be performed, followed by the automatic selection of the appropriate transfer technique to use. There are still many obstacles that must be overcome before any such complete system could be created. This leads us to our future work.

8.2 Future Work

Currently there is no way to integrate exploration during localization with the exploration necessary during transfer. Furthermore, although our localization technique determines whether a target task matches a task in the library, a completed system might use a modified version of our task localization technique to determine whether a target task belongs in a particular cluster, thus combining task localization and task clustering. Then a multiple task transfer technique could be employed to move information from the most likely cluster to guide the agent’s exploration while task localization was being performed.

BTLT should be expanded to handle non-MGMDP cases. Although we have shown that multiple task transfer could be performed through a modified form of
direct transfer and dynamic sub-transfer, there is currently no method for performing multiple soft transfer, or multiple memory-guided exploration. Such a technique should be explored.

Experiments should be performed to determine whether multiple source transfer from the cluster of most likely past tasks is more useful than standard single source transfer from the most likely past task. The library system could possibly be combined with the SKILLS algorithm to determine structure in the past tasks that would allow the agent to automatically create theoretical compositions of tasks using the sub-transfer techniques.

Value of information [26] could be integrated with the exploration of the agent. This is difficult because the value of performing a policy when in a task that doesn’t match that policy is unknown. However several approximations to this value could be proposed.

Putting the entire process of localization, similarity discovery, and transfer together is still a daunting prospect. It is unclear how each of these techniques should be integrated. However, our research showing that each sub-piece can be done, is an important step toward the creation of such a system.
Bibliography


