Transfer Of Learnt Knowledge With Card Games

by

James R. Livingston, BSc-BComp

A dissertation submitted to the
School of Computing
in partial fulfilment of the requirements for the degree of
Bachelor of Computing with Honours

The University of Tasmania

November 2, 2005

I, James Livingston, assert that this thesis, submitted in partial fulfilment of the requirements of the degree of Bachelor of Computing with Honours, contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution, and that to my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Abstract

Reinforcement learning algorithms are an important machine learning technique, which can be applied to the process of learning many tasks. Much of the existing work on improving these algorithms, and analysis into the usefulness, only considers agents which have to perform one task. Many real-world applications of reinforcement learning algorithms require that an agent can cope with small variations in their given task, and the application of their learnt knowledge to those tasks.

I consider the application of reinforcement learning algorithms to several card games, and the process of transferring learnt knowledge between these card games. The two card games used, Cut-Throat Euchre and Sergent-Major, are similar in their rules and the strategies that are used to play the game. The differences between the two games are used to measure the effectiveness of transferring knowledge between them, using a common-state approach.

These simulations of playing card-games indicate that the tasks of playing these two games are similar enough that knowledge can effectively be shared between the two. An improvement in the ability of an agent to play one of the games, results in a significant improvement in the ability of the agent to play the other game.

I would like to take the opportunity to thank the following people:

- Ian Lewis, for offering the project which has proven to be very interesting and thought-provoking, furthuring my understanding of the field.
- Other members of the School of Computing, who have provided advice on how best to set about doing my research.
- My friends, who have provided entertainment and reminded me to enjoy life and spend time doing things unrelated to the University.
- My parents, who have offered their continual support throughout my studies, free from pressure to do well.
- My fellow honours students, who listened to me ramble on about my research and gave their ideas and thoughts.

All of these people and more played an important part in enabling me to complete the Honours year.

Contents

List of Figures

Chapter 1

Introduction

Much of the work done on reinforcement learning systems is aimed at improving how well an agent can learn a task, increasing the speed at which it learns, and investigating which tasks it can learn well. This is because these are solid goals of finding better techniques and methods that can be applied to various situations. Other work focuses on forming generalisations within a single task, such as forcing an agent to learn on a small (but widely spread) points in the state space, and then determining how well it can interpolate between those points to form good solutions. Little research has been done into forming generalisation across different tasks, and investigating how agents trained at one task perform at different tasks.

The aim of my research is to determine how effectively generalisations can be formed across similar but different tasks, withing the context of card games. This goal is accomplished through the use of artificial neural networks to approximate knowledge learnt in one trump card game, and then directly applying it to the task of playing a related trump card game. This will show how well knowledge can be shared between the two games, and how effective neural networks are at encouraging the transfer of knowledge.

Chapter 2

Background

2.1 Reinforcement Learning

2.1.1 Introduction

Reinforcement learning is the area of artificial intelligence which deals with *agents* immersed in an *environment*, which attempt to learn a *policy* to determine the optimal *actions* for the current state of said environment. Techniques developed in this area can be applied to many diverse diverse areas including: controlling robots, optimisation problems and game playing.

Reinforcement learning specifically deals with how agents can learn successful decision policies by experimenting with their environment. Generally feedback of this success is quantised into a numerical *reward*, which indicates the desirability of the resulting state. (Mitchell, 1997) states that the goal of an agent is to learn from this indirect (possibly delayed) reward to choose a sequence of actions that produce the greatest cumulative reward. For many systems of reinforcement learning an agent has no prior knowledge of the environment or the likely results of any actions; this 'clean slate' lets an agent learn solutions to a task, without any bias given by what humans expect.

In many situations an agent will not be aware of what is goal actually is, and all the agent can determine is what effect it's actions have on the environment, and whether

it's actions result in a reward. An agent may have many goals, some complimentary, some conflicting, and they may be made up of 'sub-goals' which could lead to a larger one.

Agents also face a trade-off between *exploration* of the state space by performing untried actions, and *exploration* of it's learnt policy to attempt actions that are likely to yield a high reward. This is made more complicated by *partially observable states* for which the agent only has partial knowledge of the environment, in which case it may have to rely on previous observations.

(Keerthi and Ravindran, 1995) provides a good overview of reinforcement learning, and surveys the techniques available in the field. They also discuss some of the mathematical foundation of reinforcement learning, and some of the potential problems that newcomers to the area are likely to face.

2.1.2 Formal definitions and mathematical foundation

As with many other areas of artificial intelligence, reinforcement learning has a solid mathematical foundation. The field is based on such mathematical concepts as *Markov Decision Processes* (MDPs) and *Learning Theory* .

Formally an agent is a decision making entity that exists with an environment described by the set of possible states S. The agent and the environment form a feedback look, as shown in 2.1, with the agent performing actions which result in a change of the environment's state. The set of possible actions that can be performed by the agent while in state s is given by A_s. For each time interval t, when the state of the environment is s_t, the agent chooses an action $a_t \in A_{s_t}$ which will cause a transition from state s_t to the resulting state s_{t+1}.

A reinforcement learning agent also a reward $r_t \in \mathbb{R}$ for each state transition, which is indicates the desirability of the resulting state s_{t+1}. This will produce a sequence of states s_i, actions a_i and rewards r_i, as shown in 2.2.

The goal of the agent is to learn a decision policy $\pi : S \rightarrow A$, that maximises the expected sum of the all rewards, with future rewards possibly discounted exponentially according to their delay. The cumulative reward for all future time is given

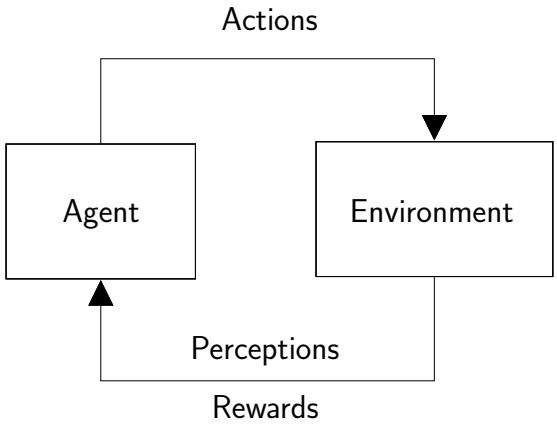

Figure 2.1: Feedback loop formed by an agent and it's environment

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \cdots$$

Figure 2.2: Sequence of state transitions of the agent-environment pair

by:

$$\text{cumulative reward} = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{i=0}^{\infty} \gamma^i r_i, \text{where } 0 \leqslant \gamma < 1$$

2.1.3 Rewards

The concept of a reward is an central part of reinforcement learning, and different ways of dealing with rewards leads to varying approaches to the topic. A positive rewards indicates a resulting state that is 'good' and a negative reward indicates one that is 'bad', with the magnitude of the reward relative to how good or bad the state is.

As an example consider a scheduling problems, such as for a pizza delivery company trying to determine which cars to send on various jobs. A negative reward could be applied for each minute that a client has to wait for their pizza, another negative reward for each litre of fuel used by a car, and a large positive reward for each pizza delivered. The relative sizes of the rewards show the company's policies on the trade-offs that must be made.

This is a form of *actor-critic* learning, as described in (Sutton and Barto, 1998), in which there are two entities embedded in the environment. The first is the actor

actor, which is the entity which is attempting to learn a policy. The second is the critic which fulfils the tasks of deciding how favourable the actions taken by the actor were, and assign rewards to the actor. The relationship between the environment, the actor and the critic is shown in Figure 2.3.

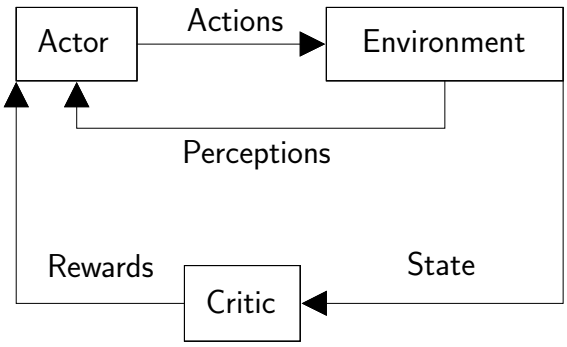

Figure 2.3: Feedback loop formed by the environment, the actor and the critic

One of the most important respects of why reinforcement learning is different from other areas of artificial intelligence is that of *delayed rewards*. The task of an agent is to learn a policy π that maps from the current state s to an optimal action $a = \pi(s)$. In reinforcement learning, an agent is not (directly) aware of the results of actions, as it is only given a sequence of immediate rewards, and not a mapping from action to reward.

As such this leads to the problem of *temporal credit assignment* which occurs because an agent cannot know which of all the taken actions lead to the reward being given. In many circumstances there was no single action that lead to the result, and it occurred due to the combination of actions taken. A reward may be the result of an action performed many time-steps ago (a delayed reward) or even the result of a combination of action that the agent has taken. An agent must have a method for assigning credit to actions when a reward is received.

2.1.4 Markov Decision Processes

A *Decision Process* (DP) is a formal structure describing the response of the environment to an agent's action giving the reward $r_t = r(s_t, a_t)$ and the succeeding state $s_{t+1} = \delta(s_t, a_t)$. A *Markov Decision Process*(MDP) is a Decision Process in which the function $r(s_t, a_t)$ and $\delta(s_t, a_t)$ are subject to the constraint that they only

depend on the current state, not any previous state. This is sometimes referred to 'Independence of path' for reaching a state, because the equation results of any future actions will not depend on the 'path' of action that an agent took in reaching that current state. Although $r(s_t, a_t)$ and $\delta(s_t, a_t)$ only depend on the current state s_t they are not necessarily deterministic, and in the context of card games they are usually non-deterministic.

Given that the task of an agent is to learn a policy, $\pi \colon S \rightarrow A$, the question is how to specify the policy π that the agent should learn? One obvious approach is to choose the policy that will produce the greatest cumulative reward; for which we define the *value function* (future cumulative reward) $V^{\pi}(s_t)$, that will be achieved by an arbitrary policy π from an initial state s_t (currently in time-step t) by the equation

$$V^{\pi}(s_t) \equiv r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where r_{t+i} are the future rewards generated by starting at state s_t and iteratively using policy π to select actions for every time t, and with the discount-factor $0 \leqslant \gamma < 1$.

The leads to the definition of an agent's task to be the learning of an *optimal policy* π^* which will maximise $V^{\pi}(s)$ for all possible states $s \in S$; i.e.

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

A reinforcement learning agent cannot learn the optimal policy $\pi^* \colon S \rightarrow A$ directly, as it does not have data of the form $\langle s, a \rangle$ but in the form $r(s_i, a_i)$. There are a number of methods for learning from such data, two of the more common being known as *Q Learning* and *SARSA* which are based on *Temporal Difference* learning.

Q-learning is described in (Watkins, 1989), as an on-line reinforcement learning algorithm that does not need a model of it's environment. Q-learning, is a simple incremental algorithm that was developed from the theory of dynamic programming (), for delayed-reward reinforcement learning. In Q-learning, policies and the

value function are represented by a two-dimensional lookup table mapping states to actions.

2.1.5 After-states

(Sutton and Barto, 1998) describes the concept of *after-states* which occur when some of the dynamics of the task are known. In particular an after-state is a state which is known to be an intermediate state of the transition from s_t to s_{t+1}. These commonly occur in games when an agent knows what the results of their chosen action will be, but not the response of the environment (including other players). Taking after-states into account can produce a more efficient learning method, because they can determine if two actions will have the same result, and hence the same value function.

After-states are not often used in analytical reinforcement learning projects, because they do not provide anything than cannot be done by a system that does not use after-states. In reinforcement learning systems that are to be used in a practical situations, after-states can prove a elegant method for improving the time-efficiency of learning algorithms.

2.1.6 Function approximation

Much of the literature on reinforcement learning techniques has been developed to be applied to discrete finite-state Markovian Decision Processes. These techniques store the learnt knowledge in tabular form, which proves to be unwieldy for tasks involving high-dimension state spaces, because of the extremely large amount of information that must be kept. Effectively dealing with such high-dimension state spaces requires a form of approximation, so that the amount of information to be stored can be reduced, resulting in a tractable problem.

Conceptually one of the easiest techniques is to model the information using linear functions, which has the advantage of being easier to analyse mathematically than most other forms of function approximation. (Schoknecht, 2003) discusses many of the issues surrounding using linear function approximation in conjunction with

reinforcement learning, and provides some insightful background. Given certain limitations and conditions, it can be proven that a reinforcement learning system using linear function approximation will converge to an optimal policy.

Using artificial neural networks is another popular method of providing function approximation to deal with the problems of high-dimensionality state spaces. The complex nature of neural networks makes it infeasible to formally prove many of the theorems that apply to reinforcement learning systems which store data in a tabular (exact) form. (Rummery and Niranjan, 1994) discusses how various forms of neural networks lean using an "on-line" policy, to solve a robot navigation problem.

2.2 Artificial Neural Networks

Artificial Neural Networks are a technique for machine learning based upon the concept of a network of *neurons*, modelled on how physical brains works. The roots of artificial neural networks extend back over 60 years to models of neurons created by McCulloch and Pitts (McCulloch and Pitts, 1943) and the psychological ideas from Hebbian Learning (Hebb, 1949).

Figure 2.4 shows how a typical articial netural network is laid out, with a number of inputs, a number of outputs and several *layers* of neurons.

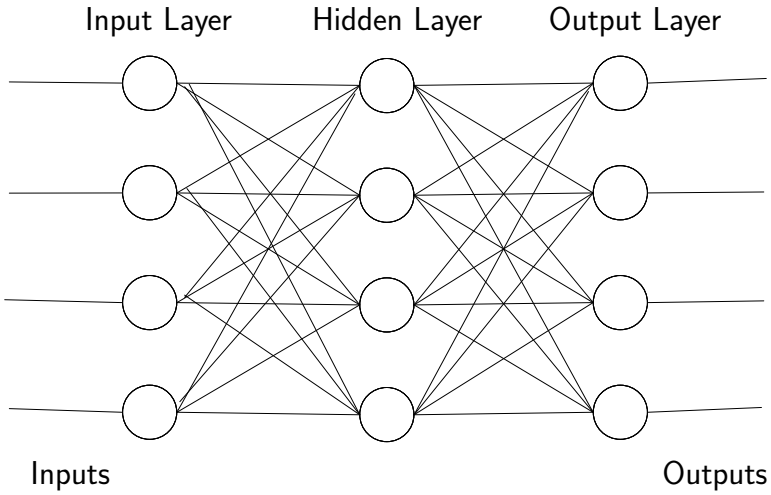

Figure 2.4: Layout of an artificial neural network

Each neuron within a neural network acts upon it's own inputs to produce an output, using a formula dependand on what type of neuron it is. The most commonly used type of neuron is called the *percepton*, which uses a simple linear formula to determine it's output of either true or false. The output of a perceptron P is given by: $P = \begin{cases} 1 & \text{if } \sum_i W_i x_i > 0 \\ 0 & \text{if } \sum_i W_i x_i \leqslant 0 \end{cases}$ where x_i is the the value from the i^{th} input, and W_i is the *weight* given to the i^{th} input.

The exact properties of a neural network depends greatly on it's configuration. A common configuration (the one shown in figure 2.4) is known as a *multi-layer perceptron* (MLP) network. In a MLP network perceptrons are connected together in such a way that they form layers, with perceptrons in the i^{th} layer having their inputs being the output from the neurons in the $i - 1^{th}$ layer. There are a variety of other network configurations that have been researched, including Cascade-Correlation

Networks (Fahlman and Lebiere, 1990) and Radial Basis Function Networks (discussed in (Poggio and Girosi, 1989)). These networks contain *hidden nodes* which are those nodes that are not directly connected to the input values of the output.

Learning with neural networks

Neural networks learn by adjusting their weightings in response to being told what their output should have been. (Pearlmutter, 1995) provides an overview of several methods for adjusting the weights of *recurrent networks*, which include the multilayer perceptron networks with hidden nodes. These methods usually involve taking the *error* in the output, which is the difference between the actual and desired output, and adjusting the weights so as to reduce the error.

There are a number of different techniques for adjusting the weightings, but the most widely known is the *backpropagation* method. Backpropagation in neural networks involves taking the error and adjusting the weights of the output layer, then propagating the error through to the second-last layer, and so back to the input layer.

2.2.1 Using neural networks with reinforcement learning

Artificial Neural Networks have been widely used in reinforcement learning, and have been the subject of a large amount of research. The two essential aspects of applying artificial neural networks to reinforcement learning are that of action selection, and learning from the training episodes.

The process of action selection when using neural networks in conjunction with reinforcement learning is exactly the same as when using any other function approximation method. The current state is encoded and set as inputs to the neural network, and the potential actions are evaluated by encoding them as inputs and iteration over the set of possible actions calculating the output value for each action. The selection of action from the output values is based on which action selection algorithm is used. Greedy selection chooses the action with the highest value, ϵ-greedy selection has probability ϵ of choosing a random action, and probability $(1 - \epsilon)$ of

picking the action with the highest output value.

The issue of training a neural network used for reinforcement learning is more complex, because rewards may be given after the action(s) that lead to the reward. Various techniques such as *eligibility traces* as used to assign the reward value to actions. Once these values has been assigned training can be performed by the use of backpropagation (or other technique) with the reward value for an action as the 'correct' output value. This causes connections that contribute to choosing good decisions to have their weights increased, and connections that contribute to bad decisions to have their weights reduces.

Care must be taken to ensure that the assignment of rewards to actions is fair, which is a difficult task. In general assigning a large part of the reward to recently performed actions is used, because they are more likely to be involved in the good decision than actions performed earlier. However in many situations, especially games early decisions often have a very large affect on the outcomes later. An example is that in chess, losing your queen early in the game is likely to lead to loss later. Similar situations arise in card games, playing the wrong card early may lead to a poor selection of cards to play later in the hand.

(Olson, 1993) investigates the use of artificial neural networks with reinforcement learning (using the temporal difference method), as applied to mapping mazes, playing blackjack and laying tic-tac-toe. He finds that for these tasks neural networks form reasonable methods of function approximation to apply to reinforcement learning with these tasks.

The combination of artificial neural networks and reinforcement learning is also used in (Lin, 1992) and (Rummery and Niranjan, 1994), where they are used to learn policies for controlling robots. Those two papers approach the application from a different point-of-view, but they both find that neural networks can be an effective method of function approximation.

2.3 Card Games

2.3.1 Introduction to Trump Games

There are an exceptionally large number of card games that exist in the world, these can be divided into many categories, and sub-categories. For the purposes of research into *transfer of knowledge* several games that are closely related need to be chosen. These games also needs to be simple enough that reinforcement learning can be used in conjunction with these games, yet difficult enough that the advantage given by having learnt a similar game will be discernible.

One group of card games that fits the above criteria is generally referred to as the *Trump Group*, which forms part of the category *Trick-Taking Games*. The Trump Group has various subgroups in which rules and game-play vary, but they all have certain properties in common. (McLeod,) provides an overview of the Trump Group and Trick-Taking Games, and the more common games that belong to it; (Gibson, 1974) also provides a good description of many of those games.

In Trick-Taking games every player has an equal number of cards, and a hand is split into *tricks*. During each trick every player plays one card, and the player who played the highest card[1] wins that trick. At the end of every hand each player is given a score depending on the number of tricks they won, and other criteria which vary from game-to-game.

In games belonging to the Trump Group, there is usually a suit that is named as *trumps*. For each trick the player who won the previous trick may *lead* any card they wish. Every other player must play a card of that suit if they have one, or play any card if they do not. If there is a suit named as trumps and a card of that suit was played, whichever player played the highest card of the trump suit wins the trick; if no trump was played, the player who played the highest card of the suit that was lead wins the trick.

In trump games the cards of a suit are ranked in the order A-K-$\cdots$-3-2, with the exception of a trump suit which is ranked in the order J-A-K-Q-10-$\cdots$-3-2. Many

[1] what the highest card is defined to be depends on the rules of the game, and in may also depends on earlier events in the game

trump games also use *bowers*: the joker (also called the *best bower*) is the highest card in the game, the jack of the trump suit (*right bower*) is the second highest card, and the the jack of the other suit of the same colour as the trump suit (*left bower*) is the third highest card. All bowers are considered as part of the trump suit, including the left bower (which would normally be in a different suit).

2.3.2 Games chosen for research

Two closely related card games *Cut-Throat Euchre* and *Sergent—Major—General* will be used for the majority of research, with *Five Hundred* (also in the Euchre Group) and *Oh Shit!* (in the *Exact-Tricks Group*) providing possibilities for additional research. These have been chosen because all four games are fairly similar (being in the Euchre Group), with Cut-Throat Euchre and Sergent—Major—General as the two used for major study because neither has an important form of bidding which would affect the ability of an agent to learn. Oh Shit! as has one significant game-play difference in that it requires a bid of the *exact* number of tricks that the agent expects to win, rather than a bid of the minimum; and Five Hundred has partnerships, which would significantly complicate the process of reinforcement learning.

2.3.3 Cut-Throat Euchre)

Euchre is a card game belonging to the *Euchre Group* (a sub-group of the Trump Group). For Cut-Throat Euchre, a three player variant, a 24 card desk consisting of the A-K-Q-J-10-9 of each suit, is used. Each player is dealt five cards ant then the players take turns to bid. A bid consists of the number of tricks they believe they can win (from one to five) or a pass, with each bid being higher than previous bids. If two players pass in succession, the remaining player has can name the suit that will be trumps (or declares there will be no trumps).

The tricks are played as a regular Trump Game. Each trick that a player wins scores them one point, with that exception that the player who won the bid must win that number of tricks or lose 5 points. The first player to reach 15 points win the game;

if two people pass 15 in one hand the one with the higher score wins.

2.3.4 Sergent—Major—General (aka 8–5–3)

Sergent—Major—General (SMG) is a three player card game belonging to the *Whist Group* (a sub-group of the Trump Group). A standard 52 card deck is used and 16 cards are dealt to each player. The dealer than names any suit as trumps, discards any four cards from their hand, and takes the four remaining cards (the *kitty*) into their hand. The tricks are played as a regular Trump Game, with each player trying to win as many as possible; the Dealer having a target of 8, the next player 5, and the last player 3. If a player takes 12 tricks the game ends with them having won.

If a player win more than their target they are *up* that number of extra tricks, alternatively they are *down* a number of tricks if they win less. In the second and subsequent hands each player who was up on the previous hand gives away one unwanted card per over-trick to a player who was down, and that player must return the highest card(s) held of the same suit(s). Then the dealer names trumps, discards four cards and takes the four un-dealt cards. The exact procedure for card exchange is: (McLeod,)

- If just one player was up, that player gives each of the other players as many unwanted cards face down as they had under-tricks. These cards are all given simultaneously. The other players add these cards to their hands, and for each card received, they give back face down their highest card of that suit. A player who has no other cards of the suit received will of course have to give back the same card.
- If two players were up, the player with the higher target for the hand about to be played trades first. This player gives (face down) as many cards as he or she had over-tricks to the player who was down, and that player gives back face down the highest card(s) held in the same suit(s). After that, the other player who was up gives a card per over-trick to the player who was down, and receives in exchange that player's highest card(s) in the same suit(s).

2.4 Applying reinforcement learning to card games

There have been a number of attempts to use reinforcement learning to create agents capable of playing card games. In unsupervised learning of card games is a difficult process, although there have been some definite successes.

Evolving Backjack strategies using reinforcement learning is discussed in (Kendall and Smith, 2003), which evaluates how well a reinforcement learning agent can learn to play with only a small amount of training (1000 games). In that paper they find that the highly random nature of Blackjack leads to some bad decisions based on short-term gain, however they also note that in longer learning runs this may prove to be an advantage.

(Kendall and Shaw, 2003) describes an adaptive cribbage player that learns using evolutionary strategies. In their conclusion they note

> It is gratifying to see that a player that evolves it's discard strategy is able to compete with a commercial application and it demonstrates that players that have no strategy programmed into them are able to evolve strong playing styles that are able to compete with players that have been explicitly programmed with game strategy.

This is one example of an agent that successfully learnt to play a relatively complex card game via unsupervised learning. Although these examples do not involve trump card games they do give some indication of what to expect when applying reinforcement learning to trump card games. The conclusions drawn in those papers lead to the expectation that reinforcement learning should be able to learn trump games, and that the highly random nature will be both boon and bane, as it will help exploration but slow the learning process.

2.5 Transfer of knowledge

2.5.1 Introduction

A number of methods exist in reinforcement learning for transferring knowledge between tasks, but there has not been extensive work done on evaluating these methods. I propose to investigate one (or more) of these methods with the framework of the card games described above, determining whether the method(s) can reduce the amount of training required to play multiple different games. Valuable insight could be obtained from various situations including, but not limited to:

- Taking an agent trained on Game A, and making it player game B without any further training. The success rate of this could be compared to an agent trained on game B.
- Taking an agent trained on Game A, and giving it a small amount of training at game B before playing game B. This could be compared to an agent trained on game B, to determine whether the agent trained on A can learn game B more quickly.
- Taking an agent and alternately training it on game A and then game B. This could be compared against an agent trained only one game (for A and B) to determine whether the agent can get to an equal level with less training on that particular game.

2.5.2 Common State

The first method of transferring knowledge between tasks would be via the use of *common state*. This approach requires that as much of the environmental state as possible of each task is identical, so learning about the meaning of that state can be shared. This approach works by sharing learning about elements of the common state between tasks.

To apply this method the state of each task needs to be divided into two sections: the common part, and the state-specific part. When transferring between tasks,

the agent incorporates the learning related to the common state into the combined learning for each task. As an example of an agent that is learning about two tasks, A and B, there are three sections of learning that the agent must store, A-specific learning, B-specific learning and the common learning. Once the agent has learnt about task A it will divide it's learning into the A-specific parts and common parts, and store these separately. When attempting to learn about task B, it will retrieve the common learning and use that as a basis for learning about B; and once complete it will divide in new learning into B-specific and common parts.

2.5.3 Generalisation in reinforcement learning

One important factor in transferring knowledge from one task to another with common state will be that of the generalisation of information. Generalisation is performed in reinforcement learning by using some non-exact function to approximate the value function $V^{\pi}(s)$. One good example is that of using a neural network to provide an approximation of $V^{\pi}(s)$ – which was successfully used to play games in the system TD-Gammon described in (Tesauro, 1995). If a neural network has (almost) identical input and output it should be trivial to attempt to use the common-state method to transfer learning between tasks.

2.5.4 Decomposition into Elemental Tasks

Another method for transferring knowledge between reinforcement learning tasks is to decompose them into *elemental sub-tasks*, as described in (Singh, 1992). Decomposition involves breaking each composite task into a number of elemental sub-tasks, where elemental sub-tasks are common between multiple composite tasks.

For this method each elemental sub-task much be a Markovian Decision task (MDT) and the composite tasks a ordered sequence of MDTs. The algorithm for learning composite tasks can be simplified to the following, which will lead to knowledge of how to complete each elemental sub-task is shared across all composite tasks.

- Learn how to perform each elemental task, usually via reinforcement learning.

- Treat the learning of composite tasks, as a process of developing a strategy for decomposing a composite into a sequence of elemental tasks. A method for doing this effectively, and taking advantage of processes developed by reinforcement learning is given in (Singh, 1992), which builds that method from earlier work and discusses how effective it is in learning composite tasks.

2.5.5 Transfer of Knowledge between Card Games

For the purposes of research into card games the method of common state would be best, as no obvious method for breaking games into elemental sub-tasks can be seen. The chosen games can obviously be decomposed into tricks but the only two different sub-tasks are simply play when leading versus playing when not leading; and both games would decompose identically, in which case decomposition would not provide any benefit.

In addition each game contains a pre-game task which is not shared by the other, for which a separate learning task can be used. Euchre contains the pre-game task of bidding for a number of tricks that a player wants to win and Sergent—Major—General contains the picking of the trump suit.

In both games the results of the pre-game task to not change the strategy, which is to win as many tricks as possible, but simply change the value of a players hand. In both games using a random choice for the pre-game task will an acceptable (although obviously not optimal for attempting to win) strategy. The random choice may also provide stochastic noise which could improve the agent's exploration of possible strategies.

Chapter 3

Methodology

3.1 Learning Architecture

This section describes the architecture of the framework that is used to evaluate a reinforcement learning system's ability to transfer knowledge between card games. At the highest level it can to be considered as the canonical agent-environment pair used in reinforcement learning, in which the agent takes an action the affects the environment and the agent then perceives some change in the environment.

When considering card games. the environment consists of the two opposing players, and the rules of the game which trigger responses to players' actions.

3.2 Learning Algorithms

3.2.1 Q learning

There are a large variety of reinforcement learning algorithms that could be applied to learning from card game, such as TD learning, TD(λ), Q learning and SARSA.

(Olson, 1993) uses TD learning (in the forms of TD(0), TD(1) and TDX) to learn to play Blackjack, however the application of reinforcement learning to Blackjack is

very different from trump games, because of a much smaller state space and simpler strategies. As such a very different method of performing the learning could be used, including table-based look-up and "thermometer encoding" which could be considered a degenerate form of neural network. One of the conclusions drawn in that paper is that the highly random nature of card games can lead to difficulties in learning a good strategy, even in simple games such as Blackjack.

3.2.2 Function Approximation

As a card game has a very large state space, it would be unwieldy to represent in tabular form. For Cut-Throat Euchre there are $4 \times^5 C_{32} \times^5 C_{27} \times^5 C_{22} \approx 1.7 \times 10^{15}$ possible initial conditions [1], and the number of possible states during the game is much greater than this. As such, a form of function approximation must be used, so that the problem become tractable.

There are a number of function approximation methods available, including Linear Approximation, Artificial Neural Networks and XXX. Artificial Neural Networks were chosen because they form generalisations well, which will be useful when attempting to apply learnt knowledge to a different game.

Neural Network considerations

Neural networks have many parameters, which play a large part in determining how well they will learn a particular task. As XXX describes, choosing parameters that will lead to learning that is both effective and efficient is more an art than a science. There is not a large body of work available which discusses the choice of parameters for card games, so a small survey had to be performed to establish the parameters that would be used for further study.

(Olson, 1993) says

> One of the problems with back-propagation is determining the correct topology. With small problems, trial and error can suffice; with com-

[1] 4 possible trump suits, and three hands of five cards each

> plex mappings (larger state space, bizarre non-linearities), trial and error leads to despair. Larger state spaces typically require more units which tends to paralyse the vanilla back-propagation learning rule.

While this is still true, the large improvements in computers over the last 12 years will help to mitigate some of the issues – by being able to throw more computational power at the problem, and hence have larger networks. This does not help with the fundamental problem of using vanilla back-propagation to learn complex mappings such as games.

Using different methods, such as Cascade-Correlation (described in (Fahlman and Lebiere, 1990)) and other adaptive training systems, would be likely to produce improved results. However as the application of these systems to game-learning is an active area of research, that is not well covered in literature, and so using these methods would be outside the scope of this project.

Neural Network parameters

A single layer of hidden neurons was used because increasing the number of layers would very quickly increase the dimensionality of the problem of determining good parameters. (O'Connor, 2000) shows that using multiple layers of hidden neurons work well in card games, however tuning such as network is computationally expensive and will take considerable time.

The evaluation of the system was performed with a small number of parameter variations, including 30, 40, 50 and 60 hidden nodes in a single layers, and a varying learning rate. The parameters that performed the best, and hence were chosen for use in further study were a 40 hidden nodes in a single layer and a learning rate of $\alpha_0 = 0.05$, using the epsilon-greedy strategy with $\epsilon = 0.1$ and a reward decay rate of $\gamma = 0.9$. It is likely that there is a combination of parameters that would increase the effectiveness and efficiency of learning, however establishing the exact parameters would take considerable time that could otherwise be spent on evaluating the transfer of learning.

3.3 Game Implementation

The implementation of the learning system, and the exact variation of the rules will affect how effective learning is. In order to put the result of learning in context, there needs to be a short description of the actual implementation

3.3.1 Game Rules

Several modification of the rules of Cut-Throat Euchre and Sergent-Major need to be made, so that reinforcement learning can work effectively. Some of these modification make the games moderately unlike the original versions, but that is not important for the purposes of analysing the transfer of learning

The main game feature that would make it difficult for a reinforcement learning system to learn to play Cut-Throat Euchre is the bidding phase. As the process of bidding is unrelated to the process of card-playing during the game, they must be controlled by separate decision processes. Attempting to concurrently learn good policy in separate decision processes, where a favourable result depends on good choices from all decision processes, is considerably less efficient than learning good policy for a single decision process. As learning to play a trump game is a already a difficult process, attempting to learn multiple orthogonal policies concurrently may lead the system to fail to learn any good policy.

As having the game contain a bidding phase would lead to probable failure, it was removed from the rules. The following rule variations are used to compensate for the loss of bidding

- The trump suit is chosen randomly
- The title of being the player that leads in the first trick of each hand is rotated amongst all players, with the mantle being given to a random player for the first hand of each game.
- Scoring is changed so that it does not depend on who won the bidding, or

which player is the dealer[2]. The score received by each player is a simple function of the number of tricks that player won.

Sergent-Major was modified to use a much simpler scoring system, and limited to five hands per game. The scoring system used is simple keeping a cumulative total of the number of tricks that each player is over or under by. At the end of the five hands, the player(s) who have the highest cumulative total are declared winners, and the other players declared losers.

3.3.2 Design

The implementation used for testing the hypotheses set forth is based on an event dispatch mechanism. Each game is contained as a small section of code that sends event relating to what has occurred and pick a player to ask to make their move. A generic trump game would run by sending "card dealt" events to each player, asking the leading player to choose a card to play, send out "card played" events to every player, and then ask the second player to choose a card et cetera.

The computer-controlled players respond to these events by updating their internal data to represent their understanding of the state of the game. When asked to make a move, the computer-controlled players will use their specified algorithm to determine which card to play.

This system works well to support reinforcement learning agents, because they can translate certain events into rewards, and choosing a move to make is a simple matter of applying the currently-learnt policy to the agents understanding of the game state.

[2]in a computer simulation it is obvious that no player is *actually* dealing, it is a figurative title that can be passed amongst the players

3.4 State of the Environment

The representation of the environment is an important part of any reinforcement learning system, and this of particular importance for a system that is attempting to use the common state approach to transfer knowledge between situations. A bad choice of representation of state may lead to a complete failure to learn good policy. When using neural networks, the most important part of the representation of state is the *State encoding*, which is how the state and actions are represented numerically for input into the neural network.

3.4.1 State encoding

For this research a minimalistic approach to state was used, providing raw data where possible, and not providing higher-level features that may help an agent to learn more quickly. This approach was chosen because it is portable across many similar games, and including higher-level features in the state may influence how the agent learns. The state representation chosen does not include the entire playing history of a hand, as this would make the encoding much larger and slow down the rate of learning.

The state variables used to represent the games consists of

- the set of card in the player's hand,
- the set of cards that have already been played, and
- the set of cards on the table
- which suit is trumps
- the position that the player is in

A set of cards is represented as a vector of the form $\langle C_1, C_2, \cdots, C_N \rangle$ where N is the number of cards in the deck, and $C_k = \begin{cases} 1 & \text{if the } k^{th} \text{ card is in set} \\ 0 & \text{if the } k^{th} \text{ card is not in the set} \end{cases}$

The trump suit is represented by the vector $\langle S_1, S_2, \cdots, S_N \rangle$ where N is the number of suits in the deck, and $S_k = \begin{cases} 1 & \text{if the } k^{th} \text{ suit is trumps} \\ 0 & \text{if the } k^{th} \text{ suit is not trumps} \end{cases}$

The position of a player is represented as a single number P, which indication what position the player hold with respect to the player that is leading that round. P is 0 if they are leading the round, and $P > 0$ indicates that they are the P^{th} player after the player who is leading that round.

Chapter 4

Results

4.1 Introduction

There are a large number of possible tests that could be run, to test the transfer of knowledge between the games of Cut-Throat Euchre and Sergent-Major. Four different tests have been run, to determine how effective the transfer of knowledge between those game is, including

- Initial assessment of an agents ability to learn Cut-Throat Euchre and Sergent-Major
- Learning while playing Cut-Throat Euchre, and then playing Sergent-Major with that knowledge
- Learning while playing Sergent-Major, and then playing Cut-Throat Euchre with that knowledge
- Learning while alternating between playing Cut-Throat Euchre and Sergent Major

4.2 Initial assessment of an agent's ability to learn

Before an agent's ability to transfer knowledge between games can be assessed, there needs to be an assessment of how well an agent can learn those games. This initial assessment was performed by having an agent attempt to learn to play the games, via the canonical self-playing method. The agents were trained using a neural network with a single hidden layer of 40 neurons, and the following parameters:

$\alpha_0 = 0.05 \quad \epsilon = 0.1$

$\gamma = 0.9$

4.2.1 Learning to play Cut-Throat Euchre

The agent's ability to learn Cut-Throat Euchre was assessed by having it play games against itself repeatedly, and learn from the results of those games. Due to the influences of the initial random state of a neural network, the learning process needs to be performed several times. Statistically an agent that has not learnt anything should, on average, be able to win one third of games played against a random player. The rate of winning against an agent using the *basic strategy* should be less than this, but the exact rate will depend on the details of implementation.

Five runs of learning to play Cut-Throat Euchre were performed[1], each 200 000 games long, and the agent was tested every 50 games. The averaged results of these runs are shown in Figure 4.1. The rate of winning was calculated by having the agent play 1000 games against two other players (acting randomly, or using the *basic strategy*),and smoothed by plotting the 1000 game moving average.

As can be seen, the agent learnt to play the game of Cut-Throat Euchre better, and increased it's rate of winning. Due to the random influences of the initial random values in the neural network and the non-deterministic process of learning, the resulting rate of winning has large fluctuations - however an increase in winning rate of approximately 3% occurs over the first 200 000 games. This increase in the rate of winning will provide a basis for comparison, when agents learn in different ways.

[1] increasing the number of runs would increase the statistical significance of the results, however five runs is sufficient to analyse the results and draw conclusions from

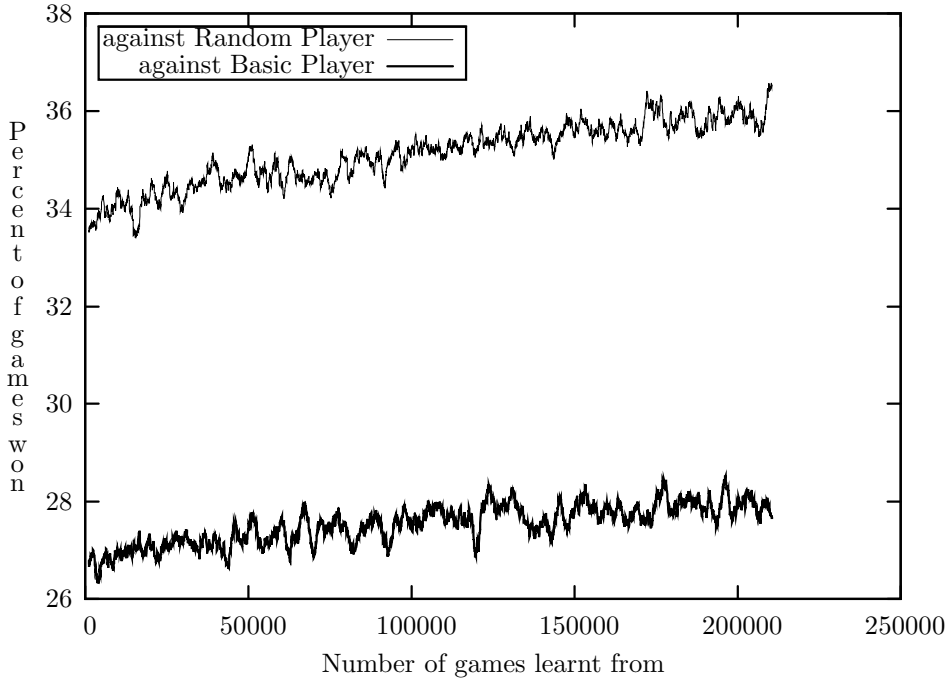

Figure 4.1: Results of learning to play Cut-Throat Euchre

As seen in Figure 4.1 the initial winning rate against the randomly-acting player is 33.6%, which is above the expected rate of 33.3% because it only contains the combined results of three learning runs. If a larger number of runs were analysed, the initial winning rate should converge to the expected value. The initial winning rate against the playing using the basic strategy is 26.7%, which is indicative of the fact that the basic strategy is more successful than playing randomly. Comparing the number of games won by the randomly-acting and basic strategy players leads to the evaluation that the basic strategy wins approximately 25% more games than acting randomly. This is consistent with results obtained when the basic strategy following and randomly acting players were competing with each other, in games containing two players with one strategy and a single player with the other strategy.

The improvement over time can be seen more clearly in Figure 4.2.

4.2.2 Learning to play Sergent-Major

As with the initial learning of Cut-Throat Euchre, the agents learning Sergent-Major had a small increase in the rate of winning over time. The rate of winning was also

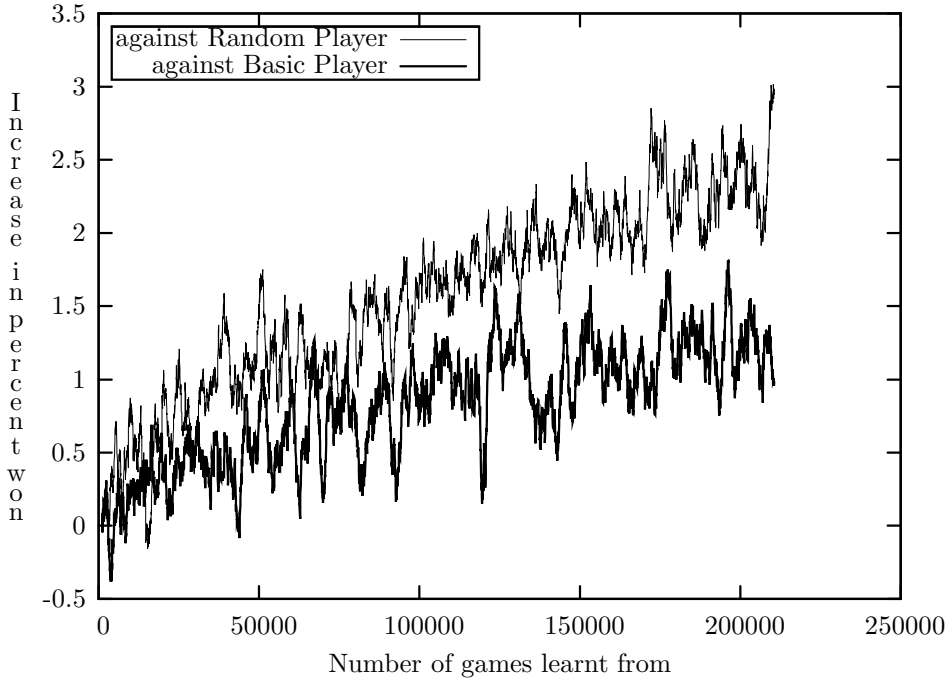

Figure 4.2: Improvement when learning from Cut-Throat Euchre

subject to the same large variations due to the random nature of the game. Figure 4.3 shows the rate of winning versus number of games learnt from, averaged across five learning runs as with the Euchre-learning agents. The improvement over time is similar to that of the agent learning to play Cut-Throat Euchre, as can be seen in 4.4.

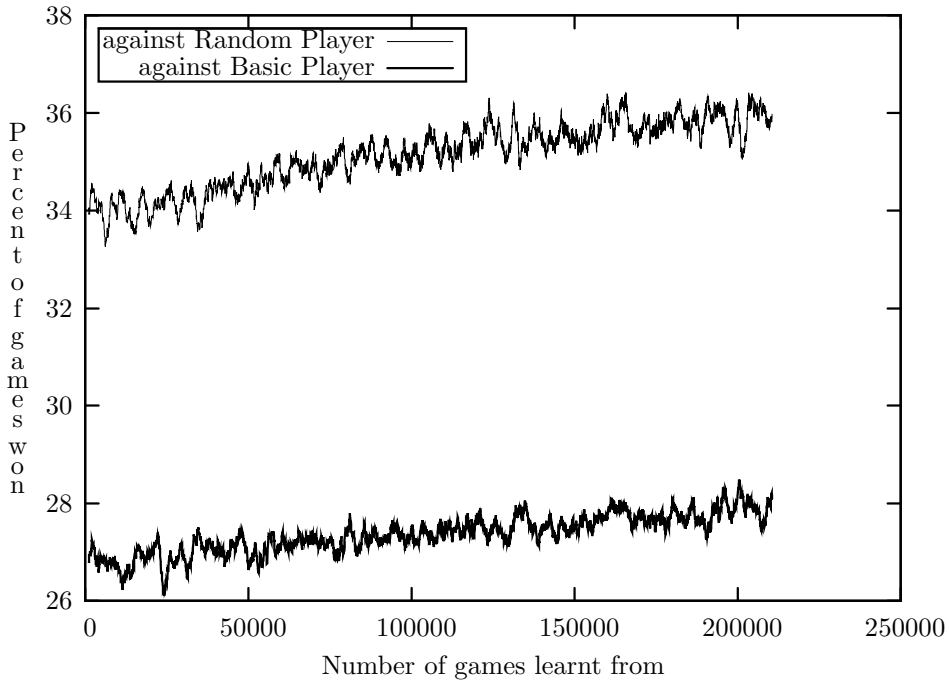

Figure 4.3: Results of learning to play Sergent-Major

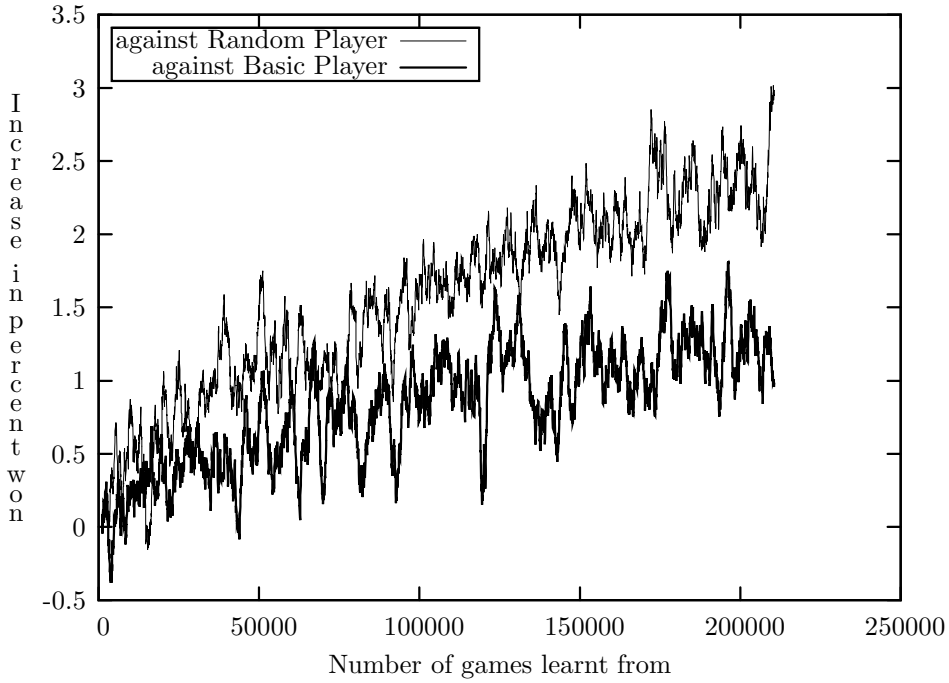

Figure 4.4: Improvement when learning from Sergent-Major

4.3 Transfer from Cut-Throat Euchre to Sergent-Major

The first test of how effectively learning could be transferred between two games was performed by assigning an agent that had been learning to player Cut-Throat Euchre to play games of Sergent-Major. As can be seen in Figure 4.5 (which is smoothed with a 1000 game moving average, as for the others graphs), the agent had an increase in the rate of winning at Sergent-Major as the number of games of Cut-Throat Euchre learnt from increased.

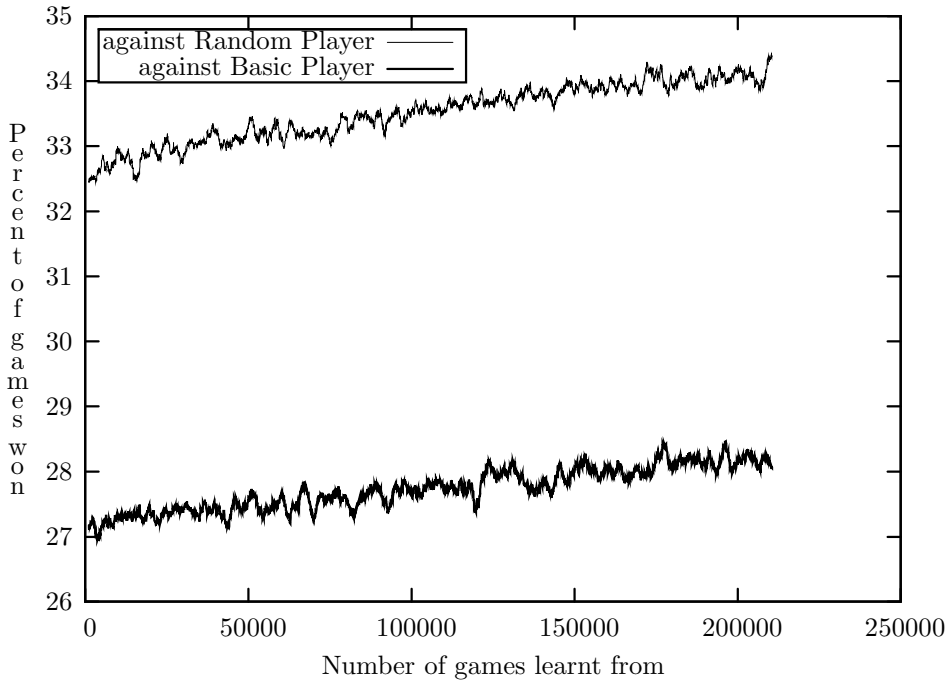

Figure 4.5: Results of playing Sergent Major, after learning Cut-Throat Euchre

Comparing the increase in the rate of winning (shown in Figure 4.6) indicates that knowledge is being transferred between games. The increase while playing Cut-Throat Euchre is larger than the one when playing Sergent-Major which, as is to be expected, the process is not perfect. As can be seen, the increase in the rate of winning at the two games is closely related – which is likely due to the high degree of similarity in the two games.

We will denote the fraction of games won while playing game X, after learning from playing game Y, as $W_{X \rightarrow Y}$. This leads us to define the efficiency of the knowledge

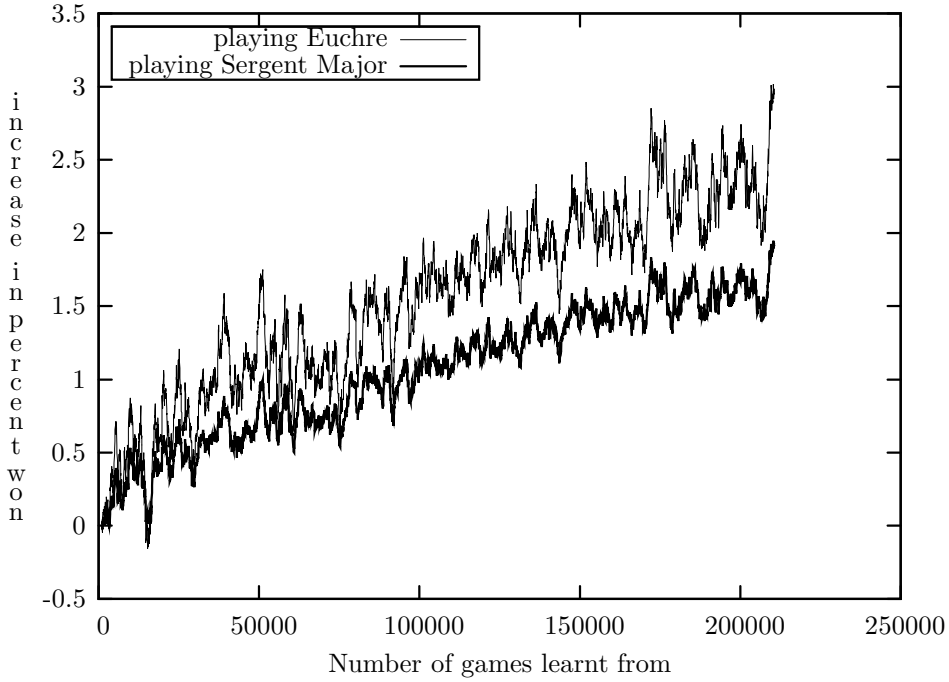

Figure 4.6: Comparison of playing Cut-Throat Euchre and Sergent-Major after learning from Cut-Throat Euchre

transfer from game A to game B as $\epsilon_{A\to B} = \frac{W_{A\to A}}{W_{A\to B}}$

The average efficiency of the transfer of knowledge from Cut-Throat Euchre to Sergent-Major is $\overline{\epsilon}_{E\to SM} \approx 0.45$, which indicated that a reasonable amount of knowledge gained from learning to play Cut-Throat Euchre is useful when playing Sergent-Major. Although the rate of winning at Sergent-Major is coupled to that of winning at Cut-Throat Euchre it does show substantial variation in places, which is like to be caused by the agent learning a strategy that works well for winning at Cut-Throat Euchre which does not work as well for winning at Sergent-Major, or vice versa.

4.4 Transfer from Sergent-Major to Cut-Throat Euchre

The results for transferring knowledge from Sergent-Major to Cut-Throat Euchre are similar to those of transferring knowledge from Cut-Throat Euchre to Sergent-Major. Figure 4.7 shows the rate of winning against the random and basic strategy players, when playing Sergent-Major after learning by playing Cut-Throat Euchre.

If the statistical noise in the results for Sergent-Major to Cut-Throat Euchre and Cut-Throat Euchre to Sergent-Major knowledge transfer is disregarded, the rate of improvement is close and well within the error margin caused by performing the learning run a small number of times. It would be likely that if a much larger set of learning runs were performed, and the statistical noise minimised, the rates of learning would be very closely related.

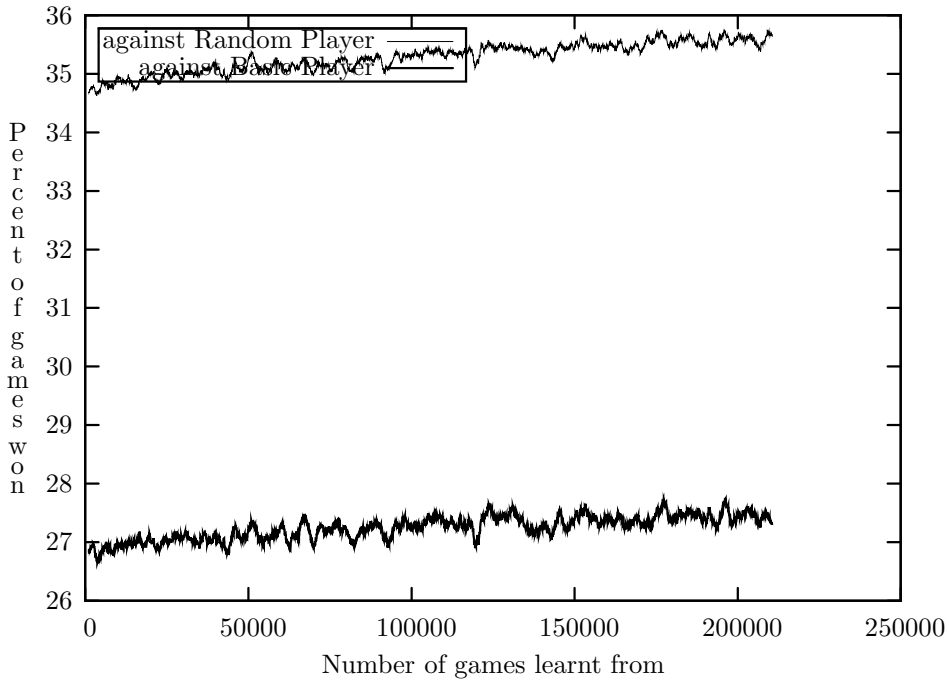

Figure 4.7: Results of playing Cut-Throat Euchre, after learning Sergent Major

Shown in Figure 4.8 is the increases in winning rate when playing the two games after learning from playing Sergent-Major. As for the Cut-Throat Euchre to Sergent-Major case, the average efficiency of the knowledge transfer can be calculated, giving an approximate value of $\overline{\epsilon}_{SM \to E} \approx 0.4$ – which is slightly lower than the value for

the former knowledge transfer.

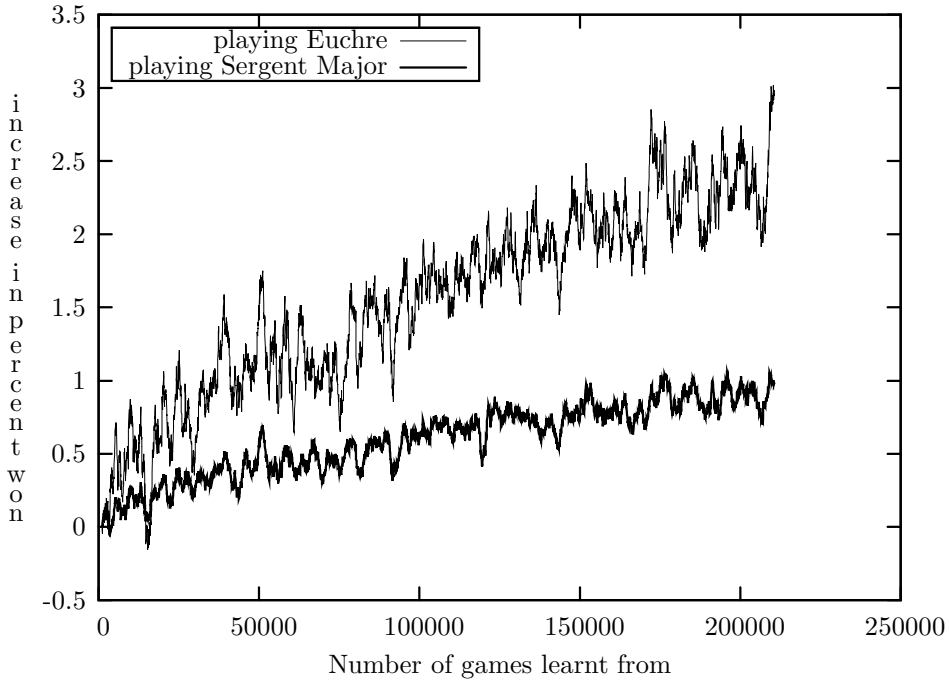

Figure 4.8: Comparison of playing Cut-Throat Euchre and Sergent-Major after learning from Sergent-Major

4.5 Discussion of results

4.5.1 Effectiveness of learning

The plots of Figure 4.1 and Figure 4.3 show how the agent using reinforcement learning is improving it's ability to play the two games. That there is a noticeable increase is significant in and of itself, because as mentioned in (Kendall and Shaw, 2003) card games are a difficult task to learn. Although the absolute increase in the rate of winning (against the random player) looks small when compared to the overall rate of winning, it results in the agent winning 10% more games than the randomly acting players. The large amount of random noise in the results indicates that a more detailed study, analysing a larger number of results, to gain a more statistically significant result needs to be done. However several conclusions can be drawn from a comparison of the rates at which the RL agent is learning to play Cut-Throat Euchre and Sergent Major.

A quantitative analysis is impossible with the data set, but it can be seen in Figure 4.2 and 4.4 that the rate of increase in winning over the first 250 000 games stays within the same order of magnitude, subject to random variations. The fact that the rate of learning has not slowed down indicates that the agent is still in the process of learning a good strategy.

Looking at the differences between the processes of learning Cut-Throat Euchre and Sergent-Major, it is obvious that the agent is learning at a very similar rate, which suggests that in both cases it is learning a strategies for general play of trump-games. This is consistent with the experiences of players of these games, as the early strategies learnt in trump games generally works as a simple strategy for other trump games.

4.5.2 Transfer of learning

As well as improving in their ability to play the game they were trained on, the agents have also improved in their ability to play the other game. Figure 4.6 and Figure 4.8 shows the relative increases in the rate of winning, and it is apparent

that the rates of increase are coupled. Generally an increase in the rate of winning at one game is accompanied by an increase in the rate of winning at the other, and the decreases are similarly affected, although there are some spikes and troughs in the winning rate that are not accompanied by a change in the rate of winning the other game, and these need to be explained.

The tightly coupled nature of the rates of winning gives substance to the hypothesis that learnt knowledge can be transferred between two similar tasks, using reinforcement learning. The 40% to 45% efficiency of the knowledge transfer shows that although knowledge can be transferred between the two games, it is perfect. The efficiency of the transfer could probably be improved with careful tuning of the state encoding, so that the apparent differences between the game are minimised, allowing the network to form better generalisations.

One of the possible reasons for the occasional disparities between the winning rate of the two is that it is just a statistical anomaly caused by the randomness of the games. To test this hypothesis, a more detailed evaluation of the winning rates around one of the disparities was done by running the testing with a larger number of games (10 000). The results of the more extensive test were extremely similar to that of the test with 1000 games, which offers some proof that the disparity is not due to statistical noise.

A second possible reason is that the agent has learnt some piece of strategy that is not useful (or as useful) in the other game. Although the games are similar they are not identical, which can lead to differences in how effective certain strategies are. That these differences in strategy lead to less efficient transfer of knowledge is to be expected, as machine learning techniques of this type do not usually lend themselves to sharing knowledge between tasks.

The second reason is highly likely to be the cause of the differences in the rate of winning, and would be consistent with the expectation gained from other studies into the area. As learning progressed further, the strategies learnt for the two games would probably become more dissimilar, as the general trump-game strategies had been learnt and the agents began to learn more games-specific strategies. This would lead to a reduction in the effectiveness of transferring knowledge as it became better at playing the games.

4.5.3 Effectiveness against players with non-random strategy

The comparisons drawn so far have been against a randomly-acting player. This is because the reinforcement learning agent is effectively acting randomly before it has learnt any strategy, and so forms a good base for comparison.

As can be seen in Figures 4.1, 4.3, 4.2 and 4.4 the reinforcement learning agent fares much worse when playing the basic-strategy using player than when playing the randomly-acting player. It is obvious that should be the case, because the basic-strategy using playing is better than the randomly acting player, but of more interest is how it plays as it improves.

As the agent is trained by playing games, there is a noticeable improvement in it's ability to win against the basic-strategy player, albeit less of an improvement than against the randomly-acting player. Looking at the fine structured variations in the rate of winning, it can be seen that the variations are matched almost identically. This indicates that the strategy being learnt is also effective against the basic-strategy player, and that the effectiveness improves at a related but lower rate.

Chapter 5

Conclusion and Future Work

This paper has presented a simple method for transferring learnt knowledge between similar card games. This method has been demonstrated to be effective at sharing knowledge between Cut-Throat Euchre and Sergent–Major, with an efficiency of over 40%.

The reason for this high level of sharing efficiency is likely to be related to how similar the two games are. Further study could be done into how well knowledge can be transferred between more dissimilar games, or games belonging to groups other than trump-games. Of course these results are very far from conclusive because of the relatively small number of learning-runs that were performed, and deeper understanding could be gained by performing the analysis over a larger set.

To be considered is that this ability to share knowledge between the games may not apply further on in the learning process, as the study was only done with 250 000 games to learn from. As a larger number of training games are used, the agents may begin to learn strategies that apply only to one of the games – which would reduce the efficiency of the transfer of learning. Additional work could be done into determining how well reinforcement learning can be used at later stages of learning to play trump card games, and how well this later learning could be transferred.

The transfers of knowledge attempted have only been unidirectional, from one game to the other. Investigation of bidirectional transfer would be of interest, and there are a large number of methods that could be used for the bidirectional transfer. Strong

generalisations about trump games could be established by training the agent of both games, alternating after a small number (possible even one) game.

Bibliography

Fahlman, S. E. and Lebiere, C.: 1990, in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems*, Vol. 2, pp 524–532, Morgan Kaufmann, San Mateo, Denver 1989

Gibson, W. B.: 1974, *Hoyle's Modern Encyclopaedia of Card Games*

Hebb, D. O.: 1949, *The organization of behavior: A neuropsychological theory*, Wiley, New York, USA

Keerthi, S. and Ravindran, B.: 1995, *A tutorial survey of reinforcement learning*

Kendall, G. and Shaw, S.: 2003, *Investigation of an Adaptive Cribbage Player*

Kendall, G. and Smith, C.: 2003, *The Evolution of Blackjack Strategies*

Lin, L.-J.: 1992, *Reinforcement learning for robots using neural networks*

McCulloch, W. S. and Pitts, W. H.: 1943, *A logical calculus of the ideas immanent in nervous activity*

McLeod, J., *Pagat's Classified Index of Card Games*, http://www.pagat.com/class/trick.html

Mitchell, T. M.: 1997, *Machine Learning*, McGraw-Hill, New York

O'Connor, R.: 2000, *Temporal difference reinforcement learning applied to Cribbage*

Olson, D. K.: 1993, *Learning to Play Games From Experience: An Application of Artificial Neural Networks and Temporal Difference Learning*

Pearlmutter, B. A.: 1995, *IEEE Transactions on Neural Networks* **6(5)**, 1212

Poggio, T. and Girosi, F.: 1989, *A Theory of Networks for Approximation and Learning*, Technical Report AIM-1140

Rummery, G. and Niranjan, M.: 1994, *On-line q-learning using connectionist systems*

Schoknecht, R.: 2003, *Optimality of reinforcement learning algorithms with linear function approximation*

Singh, S. P.: 1992, *Transfer of Learning by Composing Solutions of Elemental*

Sequential Tasks

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

Tesauro, G.: 1995, *Temporal difference learning and TD-Gammon*

Watkins, C.: 1989, *Learning from Delayed Rewards*