Multi-Agent Reinforcement Learning in Markov Games

John Wilbur Sheppard

A dissertation submitted to The Johns Hopkins University in conformity with the requirement for the degree of Doctor of Philosophy.

Baltimore, Maryland

1997

Abstract

Game playing has been a popular problem area for research in artificial intelligence and machine learning for many years. In almost every study of game playing and machine learning, the focus has been on games with a finite set of states and a finite set of actions. Further, most of this research has focused on a single player or team learning how to play against another player or team that is applying a fixed strategy for playing the game.

In this dissertation, we provide and evaluate algorithms for learning strategies in two player games with large state and action spaces. First, we focus on the class of differential games in which the state space and the action space are both continuous. We model these games as discrete Markov games and provide methods for representing the state and actions spaces at varying levels of resolution. Second, we explore multi-agent learning and develop algorithms for "co-learning" in which all players attempt to learn their optimal strategies simultaneously.

Specifically, in this dissertation we compare several algorithms for a single player to learn an optimal strategy against a fixed opponent. Next we combine the results of one algorithm—a genetic algorithm—with a second algorithm—a memory-based learning algorithm—to yield performance exceeding the capabilities of either algorithm alone. Then we explore two approaches to co-learning in which both players learn simultaneously. We

demonstrate strong performance by a memory-based reinforcement learner and comparable but faster performance with a tree-based reinforcement learner. In addition to the experimental results, we also provide an overview of machine learning and game playing as well as an overview of differential and Markov games.

Dedicated to

my wife, Justina, and my son, Jesse Carl

Acknowledgments

A work such as a Ph.D. dissertation and the research it describes is not done in a vacuum. There are many individuals who have been a tremendous help and inspiration in completing this work and to whom I will be eternally indebted. I begin by thanking my family since they are the ones who had to put up with me every day for the six-plus years it took to finish. I thank my wife, Justina, who suffered with me through classes I thought I would fail, through qualifying exams and oral exams, through seminars and conference presentations, and ultimately through the thesis itself. I know I drove her crazy and love her more than ever for keeping me sane. My son, Jesse, was born at the start of my third year and has only known me to be a student who has a job that makes me travel too much. Still, he is a delight and has been a source of energy and motivation to me throughout his four years of life.

A great inspiration to me, I will never forget the words of my father, Harry Reid Sheppard III, every time we spoke—"You've got to get that degree!" Well, it looks like I got it. Thanks for the pushing and the constant reminder of priorities. Of course, I am also eternally grateful for the unending love, faith, and support from my mother, Mary Jane

Sheppard. Whenever I was unsure of myself with some difficult task ahead, she was always there to remind me that I could do it and would be fine.

Steven Salzberg, my advisor, seemed to experience my trials all along the way. I feel he learned with me as we embarked on this journey together. Steven showed me how to do solid experimental research and filled me with a great love and appreciation for machine learning and all of its complexities. Simon Kasif served as first reader on my thesis committee. Simon led me to my first publication in machine learning before I even knew what I wanted to study. He expressed an idea, helped me flesh it out, and then encouraged my work to completion and publication. His support then continued through my exams, research, and thesis. Finally, special thanks go to Fernando Pineda for stepping in at the last minute to serve as a reader on my committee. His comments and advice added significantly to improving the quality of the thesis.

I first studied game theory with Alan Goldman. I thought I was in over my head, but Alan's enthusiasm for the subject was contagious. I knew I had to do something with games. Alan's enthusiasm and excitement stayed with me through completion of this work.

Then there is Randy Simpson. Randy hired me at ARINC ten years ago and introduced me to research and doing the work of science. While we no longer work for the same company, we still work together closely, and I consider him to be one of my closest friends. He hired me, knowing that a Ph.D. was my number-one priority. We both expected me to leave within a year. When I decided to stay and study at Hopkins, he took me under his wing, sharing his experiences as a Ph.D. student, as a test pilot, and as an air combat

expert. While I do not regret staying away from aircraft carriers, I am grateful for the fascination in military gaming and air combat instilled in me by my friend, Randy.

When I look back on how my fascination with computers began, I see a high school teacher that introduced me to punch cards, batch processing, and FORTRAN. Bernie Uzelac, a computer science teacher when computer science in high school was almost unheard of, led me into the fascinating world of computing and showed me that I could be a part of making the future reality. I joke that it is "his fault" I ended up studying computers. But looking back, I wouldn't have it any other way. Thanks Bernie.

A special thank you goes to Jim Pierce and Bill Kolb at ARINC. Jim is the Chairman and CEO of the ARINC Companies and fully supported my studies, both financially and administratively. Bill was my boss for many years and helped orchestrate the support program ARINC put together. He helped me work my way through the red tape and supported me all the way.

Of course, who can forget all of the friends and acquaintances I found during this great, six-year adventure. To Sreerama Murthy, David Heath, David Aha, Hal Balaban, Brian Pickerall, Tom Maid, Jerry Graham, Brian Kelley, Elizabeth Reed, Lewis Stiller, Scott Weiss, Nancy Scheeler, Tensie Wenderoth, Les Orlidge, Arnie Greenspan, and Rona Greenspan, thank you and may God bless!

Finally, I stop and remember the words of my favorite composer–Johan Sebastian Bach–who annotated all of his music with the phrase, "*Soli Deo gloria.*" This can be roughly translated as, "To God alone the glory." While I have dedicated this work to my family, it

is my Father in Heaven to whom belongs any glory arising from this work. And to Him, I give my eternal thanks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Games and Artificial Intelligence

Since the genesis of the study of artificial intelligence (AI), AI researchers have found game playing to be a fertile area for exploring and expanding the capabilities of machines in problem solving. Games offer the human many challenges and opportunities for exploring his or her own abilities in finding strategies for personal advance—generally at the expense of the opponent. Attempting to instill game playing abilities in the computer has opened new avenues for studying approaches to efficient search, pattern recognition, classification, and knowledge representation.

Initially, research in computer game playing was limited to constructing fixed strategies for the computer to apply against a human opponent. The worth of the strategy was determined on the basis of how well the computer fared against the human. How many times did the computer win? How long did the game last? Was the computer, at least, an

interesting opponent to play?

Until Arthur Samuel developed his checkers player [297, 298], the thought of constructing a machine that could "learn" to play a game capable of competing with a human was just a dream. With Samuel's checkers player, artificial intelligence took a step forward, demonstrating that a mere machine could not only be programmed to solve complex problems but could actual *learn* how to solve these problems by applying knowledge gained from previous experience. Since Samuel built his learning checkers player, the field of machine learning and the study of game playing have come together to yield several significant advances.

1.2 Machine Learning

Research in machine learning has focused on developing approaches for machines to automatically develop strategies and algorithms for solving many types of problems. Some of the types of problems explored include data classification, data mining, automatic programming, control theory, and planning.

Classification systems identify a concept class from a set of available classes to which a particular example belongs [3, 10, 61, 91, 119, 182]. Data classification usually proceeds from a set of available attributes and associated values. A training set is used to present examples of concept classes, and the classifier constructed is designed to be consistent with that training set. The classifier can classify new examples based on the knowledge gained from the training set.

The emphasis of data mining is on analyzing a large set of data to determine relationships among the data and to derive previously unknown concept classes. Data mining stems from work in statistics and unsupervised learning and has drawn from ideas taken from automatic discovery [2, 66, 79, 124, 185]. Many approaches to data mining use techniques such as clustering, rule induction, and classification to analyze the large set of data. From this set of data, data mining systems attempt to induce general "laws" and classification procedures for characterizing the data.

Automatic programming had lost popularity until recently with the advent of genetic programming, evolutionary programming, and relational learning. The task of automatic programming is to derive a procedure for a program using a set of examples or formal specifications of that behavior [198, 206, 208, 219]. Genetic programming approaches the problem by representing programs as parse trees of functions and variables and applying a genetic algorithm to populations of parse trees to evolve programs satisfying requirements and maximizing performance on a set of test cases.

Control theory focuses on developing optimal procedures for maintaining equilibrium or for achieving some performance objective by determining values of several "control variables" in a dynamic system [1, 26, 40, 45, 71, 100, 101, 156, 237, 252, 253]. Machine learning systems attempt to determine optimal values for these control variables from experience rather than explicitly solving a set of differential equations. For example, the classic "inverted pendulum" involves deriving appropriate durations of constant thrust on either side of a cart to keep a pole balanced upright on the cart using experience from previous

attempts to balance the pole.

Finally, planning attempts to determine optimal strategies for an agent to apply in performing some task [74, 106, 107, 131, 288]. These strategies usually consist of sequences of steps to perform, and generally, the objective is to reach some terminal state (as opposed to maintaining an equilibrium as in control problems). Classic planning problems in artificial intelligence have included stacking blocks and navigating mazes. A key distinction between control systems and planning systems is the focus on achieving a specific goal; although, frequently the distinction between them becomes blurred [252].

1.3 Learning and Intelligent Agents

Recently, the machine learning community has paid increasing attention to problems of delayed reinforcement learning [37, 121, 179, 225, 228, 318, 368]. These problems usually involve an agent that has to make a sequence of decisions, or actions, in an environment that provides feedback about those decisions. The feedback about those decisions might be considerably delayed, and this delay makes learning much more difficult. In this case, the problem is called a *delayed* reinforcement problem [109, 167, 168, 220, 221, 317, 343, 363]. The basic loop followed in sequential decision making tasks such as these includes evaluating the current state, taking an action, and computing the new state. This loop is repeated until the system either reaches a goal state or recognizes that it will never terminate.

To date, research in multiple agent planning and control has been limited largely to the area of distributed artificial intelligence [138, 283, 329, 330, 331, 341, 356] and artificial

life [83, 116, 117, 172, 299, 325]. In distributed AI (DAI), several agents cooperate to achieve some goal or accomplish some task. The task is usually one of sufficient complexity that no single agent can accomplish the task alone. Because the agents cooperate, research in distributed AI has focused primarily on developing efficient procedures for communicating between the agents to enable the agents to develop the cooperative plans.

Although artificial life research does explore issues related to both cooperation and competition, its primary focus is on the emergence of intelligent behavior in a population of agents. For example, one area of application that has received considerable attention is the evolution of foraging behavior among artificial organisms (e.g., artificial ants) in the presence of predators. Also, migration patterns of artificial birds have been evolved. In none of these cases has behavior of individual agents been the focus of the research.

Recently, work has begun to appear that focuses on learning in multi-agent systems [56, 65, 150, 303, 328, 327, 329, 341]. Stone and Veloso provide a taxonomy of multi-agent systems by focusing on attributes such as agent homogeneity, communication, deliberative versus reactive control, and number of agents [329]. Problems in multi-agent systems are distinct from problems in DAI and distributed computing, from which the field was derived, in that DAI and distributed computing focus on information processing and multi-agent systems focus on behavior development and behavior management. In addition, problems in multi-agent systems are distinct from problems in artificial life in that multi-agent systems still focus on individual behaviors and artificial life focuses on population dynamics. So far, most work in learning and multi-agent systems has focused on multiple agents' learning

complementary behaviors in a coordinated environment to accomplish some task, such as team game playing [330, 337, 338, 341], combinatorial optimization [116, 117], and obstacle avoidance [150].

The research discussed in this dissertation combines work in control and planning in the context of competitive multi-agent systems. In particular, we focus on exploring methods for the on-line learning of optimal strategies for playing differential games. Differential games are related to control problems in that the objective is to determine values of a set of control variables that optimize some objective function (namely, payoff) and satisfy the constraints of the game. The games are related to planning problems in that each player, independently, attempts to make a decision to force the state of the game into the best state for that player. We note that very little current research in reinforcement learning has considered situations where multiple, individual agents are competing and learning simultaneously how to accomplish their respective tasks.

In the case of pure competition where cooperation is assumed not to be possible, communication between the agents is not a concern because communication assumes the prospect of cooperation. Further, because each agent has an objective in conflict with the other agents, population behavior is not of interest either. Thus, the focus of the research reported in this dissertation is in the area of learning optimal strategies among multiple competing individual agents. As will be described in more detail, we focus on problems from the area of differential game theory known as pursuit games and apply reinforcement learning as the primary learning paradigm.

1.4 Assumptions

In this dissertation, we develop and evaluate several approaches to learning how to play games. We do not focus on board games (e.g., chess, othello, checkers), games of chance (e.g., craps roulette), card games (e.g., poker bridge), or any other of the normally studied games in AI. Rather, we study a branch of game theory that until now has not been studied extensively in either AI or machine learning—differential game theory.

As with any other branch of game theory, differential game theory is quite complex and offers many different avenues of research. Differential game theory focuses on games in which the moves are simultaneous, and state transitions are modeled by differential equations. Although solving differential games is difficult, differential game theory is valuable when applied directly to solving problems such as those in economics, adaptive control, and national defense. For example, several ships navigating to different docks in a harbor may have competing objectives if their courses cross. Further, missile or torpedo evasion can me modeled directly as differential games. Further, results from differential game theory can be applied to more general game theory by converting the differential equations to difference equations or some other representation of state transition. To manage the complexity of this research and to limit its scope, we make several assumptions about the games we study.

- *Two-player games.* We restrict the number of players in the games we study (with one exception) to two; however, we anticipate extension to three or more players as straightforward. The primary reason for this restriction is the availability of a large

literature base in game theory; we will be able to select several interesting games that have been analyzed that will be useful for demonstrating the capabilities of the algorithms we study.

- *Zero-sum games.* We restrict the games to zero-sum games where payoff received by one player is at the equivalent expense to that player's opponent. This restriction, generally, is not necessary. We decided to limit the games to zero-sum games to simplify analysis while still providing several interesting games. In particular, analysis is simplified because we need only consider a single payoff function rather than modeling separate payoff functions for each player.

- *Multi-stage games.* We assume that all of the games require multiple steps for each player. This is referred to as a multi-stage game and is typified by games such as chess, checkers, card games, and most sports.

- *Markov decision processes.* Because a decision will need to be made at each stage of play, we impose a Markovian restriction: The optimal decision at each stage of play depends on the current state and time only. Thus, historical information is irrelevant in determining the best course of action, and the optimal decision is a greedy one that optimizes the immediate or long-term expected reward that is estimated from the current state. The concept of a Markov decision process and a Markov game is formalized in Chapter 2.

- *Infinite horizon games.* For the games we study, we apply a finite horizon and a

terminal payoff only. However, we apply a discounted reward function as if the horizon where infinite. Many games have the ability to continue indefinitely (e.g., economic games) or require so many moves that it is impractical to consider play to the end (e.g., chess). These games usually are modeled as infinite horizon games. Infinite horizon games can be modeled as finite horizon games simply by specifying a fixed horizon for evaluation. This is only significant when assigning payoff. If the game applies a running payoff (i.e., payoff is assigned at each step), then infinite horizon games may require discounting to determining a measure of expected payoff. On the other hand, finite horizon games can compute an expected payoff directly on the basis of the length of the horizon. In the event of terminal payoff games, with a finitte but unknown horizon, discounting is still appropriate when estimating the expected payoff of the game.

1.5 Contributions

This dissertation concentrates on the problem of learning in differential games and develops approaches for learning approximate optimal solutions to discrete Markov versions of these games. Further, because differential games assume simultaneous actions by the players, these games are more general than traditional games in extensive form. Consequently, the results of this dissertation can be applied to the general class of two-player games in extensive form. The major contributions provided by this dissertation include the following:

- A direct comparison of three distinct learning approaches is given where the algorithms are applied to two difficult pursuit games. Strengths and weakness of these algorithms on tasks with large state and action spaces as well as adaptations of the algorithms to these tasks are discussed.

- A method for generating good examples for a nearest-neighbor approach to game playing is provided using a second learning algorithm. Experimental results demonstrate the utility of a combination of learning approaches in which the resulting behavior surpasses the behavior of either algorithm when functioning alone. Improvements in convergence time, successful agent performance, and memory requirements are shown.

- A novel algorithm for co-learning is provided in which all players use a shared memory base to derive optimal strategies for several differential games. This algorithm demonstrates the ability to learn approximate optimal strategies simultaneously.

- A second novel algorithm for learning is provided in which all players use a common decision tree to derive approximate optimal strategies for several differential games. This algorithm demonstrates the potential for improvement in learning time and memory requirements.

- An extensive review of machine learning and game playing is provided with a focus on reinforcement learning, Markov games, and co-learning. Several reviews of reinforcement learning and World Wide Web pages bringing together resources on learning and games exist, but this is the first review that focuses on the application of reinforcement

learning to co-learning in games.

1.6 Organization of this Dissertation

This dissertation is organized in eight chapters. In this, the first chapter, we introduce the topic of reinforcement learning and differential games by stating the problem to be solved and the applicable domains. We discuss the assumptions and motivation for the dissertation, outline the major contributions of the research, and provide a breakdown for the remainder of the dissertation.

Because the focus of this dissertation is on learning strategies for differential games, Chapter 2 provides a brief overview of differential game theory. The chapter is not intended to provide solution approaches for these games but provides definitions, objectives, and examples from differential game theory. The chapter further motivates the need for learning in differential games by discussing the complexity associated with solving these kinds of games.

Chapter 3 provides a review of past research in machine learning and game playing. This chapter surveys other work related to learning and game playing. We present the problem of game learning in the context of Markov decision processes and discuss several approaches to solving MDPs. We then focus, specifically, on reinforcement learning applied to games and end with a review of the work most relevant to this dissertation—co-learning and differential games.

Chapter 4 describes our initial results on the subject of learning and differential

games. This chapter provides the formulation of a differential game of pursuit as a reinforcement learning problem. It then focuses on one game in particular—the evasive maneuvers game—and provides a comparison of three learning strategies on two forms of the game. The three strategies are 1-nearest neighbor, genetic algorithms, and Q-learning. The two forms of the game are evasion of a single pursuer and evasion of two pursuers.

Proceeding from the results in Chapter 4, Chapter 5 describes an approach for improving the algorithm with the lowest performance by applying a novel bootstrapping method from the highest performance strategy (i.e., genetic algorithms). This chapter focuses on the one evader, two pursuer game and demonstrates that the teaching approach yields better performance than either learning method by itself. The chapter also covers potential benefits of editing the memory base and provides results of an experiment demonstrating these benefits.

Considerable prior work exists on learning strategies for one player in a game, but little work is available where all players learn their strategies simultaneously. The field of multi-agent learning is gaining interest and popularity, but to date work has focused on developing cooperating agents, not competing agents. Chapters 6 and 7 provide the results of experiments in an attempt to focus on this void in multi-agent competitive co-learning.

In Chapter 6, we present a novel memory-based algorithm for co-learning and discuss experiments on four games: 1) a simple game of force, 2) a simple game of pursuit and evasion, 3) pursuit and evasion in the half plane, and 4) pursuit and evasion with limited player mobility. Closed form solutions are provided and compared to the results of

learning for the first three games, and experiments comparing a heuristic solution to the results of learning are provided for the fourth.

Chapter 7 extends the results of Chapter 6 by providing another novel approach to co-learning using a tree partitioning the state space. The need for another algorithm is motivated by the large memory and computational requirements of the memory-based approach. The algorithm described in Chapter 7 was developed specifically to reduce these requirements while attempting to maintain good learning performance.

The algorithm described in Chapter 7 constructs a tree, partitioning the state space and associating game matrices with each partition. The algorithm grows the tree dynamically while learning strategies for both players within each partition. Chapter 7 also provides a discussion about experimental results, including player performance and learning time, directly comparing the results with the memory-based approach.

Finally, Chapter 8 provides a summary of the work performed and reviews the major contributions of the research. In this chapter, we point out several areas for future research and discuss implications derived from the current work.

Some of the work reported in this dissertation has been reported in preliminary form in papers that are cited here. Results in Chapter 4 will appear in "A Teaching Method for Memory-Based Control" by J. Sheppard and S. Salzberg in the journal, *Artificial Intelligence Review* [307] and have appeared in "Memory-Based Learning of Pursuit Games" by J. Sheppard and S. Salzberg as Johns Hopkins technical report, JHU-94-02 [304]. The results from Chapter 5 appeared in "Bootstrapping Memory-Based Learning with Genetic

Algorithms" by J. Sheppard and S. Salzberg presented at the 1994 AAAI Workshop on Case-Based Reasoning [305], "Combining Memory Based Reasoning with Genetic Algorithms" by J. Sheppard and S. Salzberg in *Proceedings of the Sixth International Conference on Genetic Algorithms* [306], and will appear in the previously referenced article in *Artificial Intelligence Review*. This research has been supported, in part, by the National Science Foundation under Grant Nos. IRI-9116843 and IRI-9223591 and through the educational assistance program at ARINC Incorporated.

Chapter 2

An Overview of Differential Game Theory

2.1 Elements of Game Theory

Game theory is the formal study of conflict and competition. The purpose of studying games is to determine rational decisions by competitors intending to maximize some payoff or minimize some penalty. Games have been studied along a wide variety of dimensions, such as the following:

- Two-player or n-player games.
- Cooperative or competitive games (or some combination).
- Constant sum or non-constant sum games.
- Single stage or multi-stage games.
- Perfect information or imperfect information games.

2.1.1 Properties of a Game

In general, a game can have $n > 1$ players. When $n = 1$, a game is considered to be an optimal control problem in which the player attempts to maximize some payoff relative to a set of fixed control laws or a fixed strategy. Much research in game theory has focused on two-player games. These games are mathematically interesting yet relatively simple to describe and visualize. In this dissertation, we focus on two-player games with one minor exception. In Chapters 4 and 5, we present a game with three players, except the game is played as a two-player game where one "player" is a team of two agents.

Cooperative games provide a means of communication between players, thus opening the possibility of the players establishing alliances or cooperating in some way. Cooperation may occur when the expected payoffs to all players are maximized but each player still has strictly selfish interests. Frequently, cooperative games limit cooperation to pre-play discussion and establishing binding agreements. In non-cooperative games, no communication is permitted between the players. Thus, pre-play discussion is prohibited and agreements cannot be made. Instead, play is determined by examining the current state and the expected payoffs.

One method of representing a game (especially a two-person game) is by using a payoff matrix. The payoff matrix has the form given in Table 2.1. Each row, i, corresponds to strategy (or action) a_1^i for player $P1$, and each columen, j, corresponds to strategy (or action) a_2^j for player $P2$. Each cell of the matrix represents the payoff received by each player given the associated actions are played. Thus $\rho_1^{ij} = f_1(a_1^i, a_2^j)$ is the payoff received

Table 2.1: Payoff matrix for two-player non-zero-sum game in normal form.

$P1/P2$	a_2^1	a_2^2	$\cdots$	a_2^n
a_1^1	ρ_1^{11}/ρ_2^{11}	ρ_1^{12}/ρ_2^{12}	$\cdots$	ρ_1^{1n}/ρ_2^{1n}
a_1^2	ρ_1^{21}/ρ_2^{21}	ρ_1^{22}/ρ_2^{22}	$\cdots$	ρ_1^{2n}/ρ_2^{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_1^m	ρ_1^{m1}/ρ_2^{m1}	ρ_1^{m2}/ρ_2^{m2}	$\cdots$	ρ_1^{mn}/ρ_2^{mn}

by player $P1$ when $P1$ plays action a_1^i and $P2$ plays action a_2^j. Further, $\rho_2^{ij} = f_2(a_1^i, a_2^j)$ is the payoff received by player $P2$ when $P1$ plays action a_1^i and $P2$ plays action a_2^j.

The matrix in Table 2.1 is actually two matrices, and games using this form are called *bimatrix games*. If $\forall a_1, a_2, f_1(a_1, a_2) + f_2(a_1, a_2) = k$, where k is some constant, the corresponding game is called a *constant sum game*. If one of the payoff functions is modified to subtract k (e.g., $f_2'(a_1, a_2) = f_2(a_1, a_2) - k$), then we find that $\forall a_1, a_2, f_1(a_1, a_2) + f_2'(a_1, a_2) = 0$, and the resulting game is called a *zero-sum game*. Zero-sum games (therefore, constant sum games) can be represented with a single matrix consisting of the values of just $f_1(a_1, a_2)$, because $f_1(a_1, a_2) = -f_2'(a_1, a_2)$.

When representing a game as a matrix, the game is said to be in *normal form*. In this case, all decisions to be made in playing the game are made "up front." In a sense, the game has been collapsed to a single "stage" where the decision is made. If decisions are made throughout the play of the game (i.e., in multiple stages), the game is said to be in *extensive form*. Games in extensive form can be represented as a tree in which decisions are made at the internal nodes of the tree that cause transitions to a new state in the game.

Finally, we can draw a distinctions between *perfect* and *imperfect* information

games. If at a given point of a game, each player knows or can deduce the entire history leading up to that point in the game (or the history does not matter), then the game is a perfect information game. We define an *information set* to be the set of vertices in a game tree, V_p^i, such that if player p is currently at vertex $v \in V_p^i$, then p does not know which vertex in V_p^i is the current vertex. A game of perfect information is one for which all information sets contain exactly one vertex. A game of imperfect information is one where there exists an information set V_p^i such that $|V_p^i| > 1$.

2.1.2 Solutions to a Game

Given the characteristics of a game, one of the principal goals of game theory is to understand the properties of optimal play. To gain such an understanding, one must define what constitutes optimal play. The most widely accepted definition of optimal play is the *Nash equilibrium point.* (Because extensive form games can be represented in normal form, for the moment, we limit the representation of the game to normal form and consider only non-cooperative games.) We define a Nash equilibrium point as follows. Given a game Γ, let $\mathcal{P}$ be the set of players and $\mathcal{A}_p$ be the set of actions (or strategies) for players $p \in \mathcal{P}$.

Definition 2.1 *Let* $\mathcal{A} = \{\langle a_1, \ldots, a_p \rangle | a_1 \in \mathcal{A}_1 \wedge \cdots \wedge a_p \in \mathcal{A}_p\}$ *be the set of joint strategies.*

Definition 2.2 *For any* $p \in \mathcal{P}$, $a'_p \in \mathcal{A}_p$, *and* $a \in \mathcal{A}\}$, *let* $a||a'_p$ *denote the member of* $\mathcal{A}$ *obtained by replacing* a_p *by* a'_p. *This is a unilateral defection from joint strategy* a *by player* p.

Definition 2.3 *Let* $f_p(a)$ *denote the payoff to player* p *given joint strategy* $a \in \mathcal{A}$.

Definition 2.4 *If, for all $a'_p \in \mathcal{A}_p$, we have $f_p(a||a'_p) \leq f_p(a)$, then $a \in \mathcal{A}$ is admissible. In other words, player p has no profitable unilateral defection from a.*

Definition 2.5 *Joint strategy $a \in \mathcal{A}$ is a Nash equilibrium point for game Γ if and only if it is admissible for all players $p \in \mathcal{P}$.*

Unfortunately, this definition is not satisfactory; many games do not have a Nash equilibrium point when considering only *pure strategies* (i.e., specific strategies prescribed at the start of play). In this case, we consider the set of *mixed strategies*. The set of mixed strategies for player p, Σ_p, is defined to be the set of probability distributions over the strategy space $\mathcal{A}_p$. From this, we can determine a Nash equilibrium point for any normal-form game.

Theorem 2.1 (Nash) *Given any n-player non-cooperative game, there exists at least one Nash equilibrium point consisting of mixed strategies.*

Proof: See [250, 254, 360].

If we reconsider the concept of a normal form game and limit our discussion to two-person zero-sum games, it has been shown that Nash equilibrium points with mixed strategies can be found by solving the following linear program. Let f^π be the expected value of the game Γ given in normal form.

$$\text{minimize } f^\pi \text{ subject to}$$

$$\sum_{a_1 \in \mathcal{A}_1} p(a_1) = 1, \text{ where } p(a_1) \geq 0$$

$$\forall a_1 \in \mathcal{A}_1, \sum_{a_2 \in \mathcal{A}_2} p(a_1) f(a_1, a_2) - f^{\pi} \leq 0$$

This will determine an optimal mixed strategy for player $P1$. For player $P2$, the dual linear program must be solved.

$$\text{maximize } f^{\pi} \text{ subject to}$$

$$\sum_{a_2 \in \mathcal{A}_2} p(a_2) = 1, \text{ where } p(a_2) \geq 0$$

$$\forall a_2 \in \mathcal{A}_2, \sum_{a_1 \in \mathcal{A}_1} p(a_2) f(a_1, a_2) - f^{\pi} \geq 0$$

This will find an optimal mixed strategy for player $P2$. The exchangeability property of Nash equilibrium points ensures that pairing any optimal mixed strategy for player $P1$ with any optimal mixed strategy for player $P2$ will maintain optimal play for both players.

If we return to the notion of an imperfect information game in extensive form, then we need to consider the strategies available to a player in an information set. Similar to defining the set of mixed strategies to each player in a game in normal form, if we define the set of "mixed strategies" for each player for each information set in a game of imperfect information, then we have defined *behavioral strategies* for each player [217].

Theorem 2.2 (Kuhn) *Given a game with perfect recall with behavioral strategies* $(\beta_1, \beta_2, \ldots, \beta_p)$ *induced by a set of mixed strategies* $(\sigma_1, \sigma_2, \ldots, \sigma_p)$*, then for each player* i*,* $f_i(\beta_1, \beta_2, \ldots, \beta_p) = f_i(\sigma_1, \sigma_2, \ldots, \sigma_p)$.

Proof: See [199].

Given the concept of a behavioral strategy, we can modify (slightly) our solution concept by applying the theorem by Nash and the theorem by Kuhn. Specifically, we

determine a payoff matrix at each stage of play and solve it as if it is a normal-form game. The resulting mixed strategies define the behavioral strategies at the stage of the game. The concept of behavioral strategies is important in Chapters 6 and 7. Further, we extend the linear programming approach to behavioral strategies in Section 2.3.

2.2 Markov Decision Processes

For purposes of this dissertation, we restrict our attention to two-person zero-sum games of imperfect information. Further, we limit the scope to positional games. A *positional game* is a game in which the sequence of moves leading up to the current state is irrelevant in deciding the optimal strategy to apply. Thus it is also a game of perfect recall because history does not matter. This property is called the *Markov* property and is derived from the study of Markov decision processes.

A Markov decision process (MDP) is defined by a set of states, $\mathcal{S}$, a set of actions, $\mathcal{A}$, a set of transitions between states, $\mathcal{T}$, associated with a particular action, and a set of discrete probability distributions, $\mathcal{P}$, over the set $\mathcal{S}$. Thus $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}$. Associated with each action while in a given state is a cost (or reward), $c(s, a)$. Given a Markov decision process, the goal is to determine a policy, $\pi(s)$, (i.e., a set of actions to be applied from a given state) to minimize total expected discounted cost. Figure 2.1 provides a graphical view of a part of a Markov decision process.

Let $f^{\pi}(s_i)$ represent the total expected discounted infinite horizon cost under policy π from state s_i. Let γ $(0 \leq \gamma < 1)$ be a discount factor which has the effect of controlling

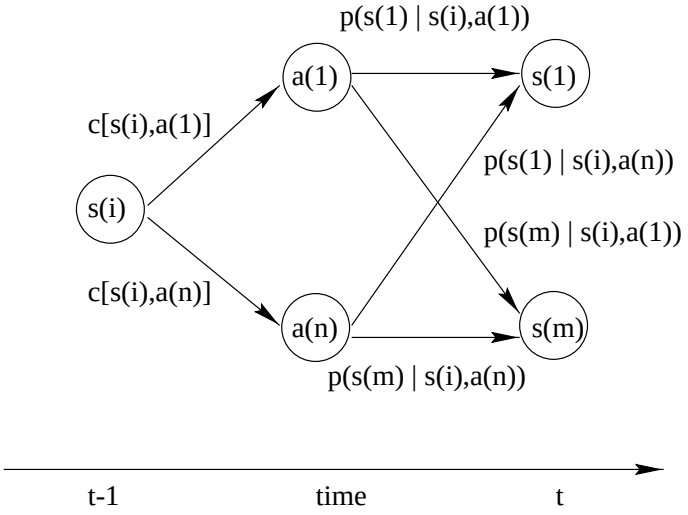


Figure 2.1: Graphical view of a Markov decision process.

the influence of future cost on π. Then,

$$f^{\pi}(s_i) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, \pi(s_t)) | s_0 = s_i\right]$$

where $E_{\pi}[\cdot]$ is the expectation given policy π. Note we can estimate $f^{\pi}(s_i)$ for some $\pi(s_i) = a$ as follows [35]:

$$f^{\pi}(s_i) \approx Q^f(s_i, a) = c(s_i, a) + \gamma \sum_{s_j \in \mathcal{S}} p(s_j | s_i, a) f(s_j)$$

From this, we are able to establish a policy, π based on the current estimate Q^f; namely, select $\pi(s_i) = a$ such that,

$$Q^{f_i}(s_i, \pi(s_i)) = \min_{a \in \mathcal{A}} Q^f(s_i, a)$$

This equation is in the form of the *Bellman optimality equation* which can be solved for all $f(s_j)$ using several techniques such as dynamic programming [41].

Two approaches to solving MDPs are value iteration and policy iteration [35, 211,

269]. The object of solving an MDP is to find a policy that maximizes the reward or minimizes the cost from the current state.

2.2.1 Value Iteration

Value iteration is an approach to solving MDPs in which the expected values for acting optimally from each state are updated based on the expected optimal values from subsequent states. The approach falls in the class of iterative relaxation algorithms [316]. Iterative relaxation algorithms take the general form

$$\begin{aligned} V_t(s) &= (1-\eta)V_{t-1}(s) + \eta\hat{V}(s) \\ &= V_{t-1}(s) + \eta(\hat{V}(s) - V_{t-1}(s)) \end{aligned}$$

where $V_t(s)$ is the value approximation at time t for state s, $\hat{V}(s)$ is a new value estimate for state s, and η is an update rate, $0 \leq \eta \leq 1$. Thus, the new value approximation is derived from the previous value approximation combined with an immediate estimate of the value for that state. Generally, this immediate estimate is derived from an immediate cost and an approximation of subsequent state values.

Initially, $\forall s \in \mathcal{S}, V_0(s) = 0$. Then one step in the value iteration algorithm consists of evaluating the following.

$$\forall s \in \mathcal{S}, a \in \mathcal{A}, Q_t^V(s,a) = c(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)V_{t-1}(s')$$

$$V_t(s) = Q_t^V(s, \pi_t(s))$$

Each iteration is called a *sweep*, and the space is swept until such time that

$$\max_{s \in \mathcal{S}} |V_t(s) - V_{t-1}(s)| < \epsilon$$

for some pre-defined error, ϵ.

Value iteration is a form of dynamic programming and can be shown to be a special case of iterative relaxation, in particular, when $\eta = 1$ and $\hat{V}(s) = Q^V(s, \pi(s))$. A more generally used form of value iteration in machine learning is Q-learning [363] which has the form

$$Q_t(s,a) = (1-\eta)Q_{t-1}(s,a) + \eta[c(s,a) + \gamma Q_{t-1}(s', \pi(s'))]$$

where s' is the state transitioned to when applying action a in state s and $c(s,a)$ is the cost of applying action a in state s. In this update equation, the role of $V_t(s)$ is filled by $Q_t(s,a)$ and the role of $\hat{V}(s)$ is filled by $c(s,a) + \gamma Q_{t-1}(s', \pi(s'))$. Q-learning is discussed further in Chapters 3 and 4.

2.2.2 Policy Iteration

Policy iteration is similar to value iteration in that a sequence of value functions is maintained. In value iteration, each value function $V_t(s)$ is the approximation of the expected cost of applying an optimal policy, π, from state s. In policy iteration, each value function, $V_t(s)$, is the approximation of the expected cost of applying a *greedy* policy, $\hat{\pi}$, from state s based on the previous value function, $V_{t-1}(s)$.

As in value iteration, policy iteration begins by initializing the value function,

$V_0(s) = 0$, for all states. Then a greedy policy is determined such that

$$\forall s \in \mathcal{S}, \hat{\pi}_t(s) = \arg\min_{a \in \mathcal{A}} Q_t(s, a)$$

where $Q_t(s, a)$ has been updated using

$$Q_t(s, a) = c(s, a) + \gamma \sum_{s_i \in \mathcal{S}} p(s_i|s, a) V_{t-1}(s_i)$$

Given the new policy, the new value function, $V_t(s)$, is determined by solving the system of simultaneous equations given by

$$\forall s \in \mathcal{S}, V_t(s) = \ c(s, \hat{\pi}_t(s)) + \gamma \sum_{s_i \in \mathcal{S}} p(s_i|s, \hat{\pi}_t(s)) V_t(s_i)$$

These equations can be solved by any of a number of algorithms for solving systems of simultaneous equations (e.g., Gaussian elimination). The policy iteration is halted when there is no change in the value estimate, $V_t(s)$ for all s.

2.3 Markov Games

The game theory literature refers to MDPs applied to games as *Markov games* [355]. A Markov game is an extension of the MDP in which decisions by multiple players must be considered, and these decisions generally conflict. Under the restriction of two-person zero-sum games, we define $\mathcal{S}$ to be the set of states, $\mathcal{A}_1$ to be the set of actions for player 1, $\mathcal{A}_2$ to be the set of actions for player 2, and $\mathcal{T}$ to be the set of transitions. We also define a set of discrete probability distributions, $\mathcal{P}$, over the set $\mathcal{S}$. Thus $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \rightarrow \mathcal{P}$. Associated with actions for each player is a cost (or reward), $c_1(s, a_1)$ and $c_2(s, a_2)$. Given a

Markov game, the goal is for each player to determine a policy, $\pi_1(s)$ and $\pi_2(s)$, to minimize total expected discounted cost given that the opponent is attempting to do the same and the goals are conflicting.

Let $f^{\pi_1}(s_i)$ be the total expected discounted cost under policy π_1 from state s_i for player 1. Let $f^{\pi_2}(s_i)$ be the total expected discounted cost under policy π_2 from state s_i for player 2. Let γ $(0 \leq \gamma < 1)$ be a discount factor. Then,

$$\begin{aligned} f^{\pi_1}(s_i) &= E_{\pi_1}\left[\sum_{t=0}^{\infty} \gamma^t c_1(s_t, \pi_1(s_t), \pi_2(s_t)) | s_0 = s_i\right] \\ f^{\pi_2}(s_i) &= E_{\pi_2}\left[\sum_{t=0}^{\infty} \gamma^t c_2(s_t, \pi_1(s_t), \pi_2(s_t)) | s_0 = s_i\right] \end{aligned}$$

Given we are restricting analysis to zero-sum games, these two equations can be combined because $c_1(s_i, \pi_1(s_i), \pi_2(s_i)) = -c_2(s_i, \pi_1(s_i), \pi_2(s_i))$. We consider $\pi(s)$ to be the combined policy of $\pi_1(s)$ and $\pi_2(s)$. Then,

$$f^{\pi}(s_i) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, \pi(s_t)) | s_0 = s_i\right]$$

as before. We can estimate $f^{\pi}(s_i)$ for some $\pi(s_i) = a$ as follows [211]:

$$f^{\pi}(s_i) \approx Q^f(s_i, a_1, a_2) = c(s_i, a_1, a_2) + \gamma \sum_{s_j \in \mathcal{S}} p(s_j | s_1, a_1, a_2) f(s_j)$$

From this, we are able to establish a combined policy, π based on the current estimate Q^f; namely, select $\pi(s_i) = \langle a_1, a_2 \rangle$ such that,

$$Q^{f_i}(s_i, \pi_1(s_i), \pi_2(s_i)) = \sum_{a_1 \in \mathcal{A}_1} \sum_{a_2 \in \mathcal{A}_2} p(a_1 | s_i) p(a_2 | s_i) Q^f(s_i, a_1, a_2)$$

where at each step, we must solve the linear program for player 1,

$$\text{minimize } f^{\pi}(s_i) \text{ subject to}$$

$$\sum_{a_1 \in \mathcal{A}_1} p(a_1|s_i) = 1, \text{ where } p(a_1|s_i) \geq 0$$

$$\forall a_1 \in \mathcal{A}_1, \sum_{a_2 \in \mathcal{A}_2} p(a_1|s_i) Q^f(s_i, a_1, a_2) - f^\pi(s_i) \leq 0$$

and its dual for player 2,

$$\text{maximize } f^\pi(s_i) \text{ subject to}$$

$$\sum_{a_2 \in \mathcal{A}_2} p(a_2|s_i) = 1, \text{ where } p(a_2|s_i) \geq 0$$

$$\forall a_2 \in \mathcal{A}_2, \sum_{a_1 \in \mathcal{A}_1} p(a_2|s_i) Q^f(s_i, a_1, a_2) - f^\pi(s_i) \geq 0$$

Selecting actions $\pi(s_i) = \langle a_1, a_2 \rangle$ involves randomly selecting the actions according to the probability distributions determined by the linear programs. These probability distributions correspond to behavioral strategies for both players. In the event the game can be solved with pure strategies, we can determine the optimal policy with,

$$Q^{f_i}(s_i, \pi_1(s_i), \pi_2(s_i)) = \min_{a_1 \in \mathcal{A}_1} \max_{a_2 \in \mathcal{A}_2} Q^f(s_i, a_1, a_2)$$

which would also be found through linear programming.

Note the optimal policies for a Markov game differ significantly from the optimal play of a Markov decision process in two ways. First, because the goals are conflicting, a "rational" player will play so as to maximize its return in a worst case scenario. This leads to the minimax formulation. Second, optimal policies in MDPs are stationary in that the policy does not change as a function of time, and the policies are deterministic in that the same action is always chosen whenever the agent is in a particular state s, for all $s \in \mathcal{S}$. For Markov games, on the other hand, at least one policy exists that is stationary, but the

optimal policy need not be deterministic. This corresponds to the notion of a behavioral strategy in which an action is chosen according to a discrete probability distribution. The $p_a(s_j)$ term reflects this probability distribution.

2.4 Differential Games

The class of problems we are studying falls in the discipline of *differential game theory.* Differential game theory is an extension of traditional game theory where the game follows a sequence of actions through a continuous state space to achieve some payoff [176, 177]. This sequence can be modeled with a set of differential equations (called kinematic equations) which are analyzed to determine optimal play by the players. We can also interpret differential games to be an extension of optimal control theory in which players' positions develop continuously in time, and we wish to optimize competing control laws for the players [126]. We restrict a differential game to be a two-person zero-sum game in which both players are required to make a lengthy sequence of decisions to maximize payoff throughout the game. Further, we formulate the game with a set of state variables and a set of control variables. At a given instant in the game, the values of the state variables completely characterize that state. The actions that are possible for each player are completely determined by the control variables.

Finding a solution to a differential game consists of computing the "value" of the game (based on the game's payoff) and determining the optimal strategies for the players that yield this value. Differential games are difficult to solve yet are important to the

military and entertainment industries. More recently, systems for intelligent highways, air traffic control, railroad monitoring, and ship routing are benefiting from differential game theory in determining how to optimize vehicle control in the presence of competing goals.

Differential game theory originated in the early 1960s [176] in response to the need for a more formal analysis of war games. In a differential game, the dynamics of the game (i.e., the behaviors of the players) are modeled with a system of first order differential equations of the form

$$\frac{dk_j^t}{dt} = h_j^t(k^t, a^t), j = 1, ..., n$$

where $a^t = (a_1^t, ..., a_p^t)$ is the set of actions taken by p players at time t, $k^t = (k_1^t, ..., k_n^t)$ is a vector in real Euclidean n-space denoting a position in play (i.e., the state) for the game, and h_j^t is the history of the game for the jth dimension of the state space. In other words, the differential equations model how actions taken by the players in the game change the state of the game over time. In these games, the initial state of the game k^0 is given. Note these differential equations are frequently represented as $\dot{k_j^t} = h_j^t(k^t, a^t)$.

The object of analyzing a differential game is to determine the optimal strategies for each player of the game and to determine the value of the game assuming all players follow the optimal strategies. Payoff is represented as a set of payoff functions of the form

$$\zeta_i = \int_0^T g_i^t(k^t, a^t)dt + v_i^T(k^T), i = 1, ..., p$$

where g_i^t is the payoff awarded at time t to player i based on the current state k^t and the action taken a^t, and v_i^T is the payoff awarded at the end of the game (i.e., time T). This

final payoff is dependent only on the terminal state k^T. Usually, differential games consider either integral payoff (referred to as a *running cost game*) or terminal payoff (referred to as a *terminal cost game*), but not both.

Frequently, differential game theorists limit the scope of their analysis to zero-sum two person games (thus $\zeta_1 + \zeta_2 = 0$) and impose a Markovian restriction such that strategies depend only on time and the current state. The differential equations define a path or trajectory through the state space of the game. The solution to the game is then computed by solving the system of differential equations such that payoff to player 1 (denoted P for "pursuer")[1] is minimized and payoff to player 2 (denoted E for "evader") is maximized, yielding minimax strategies for each player, i.e.,

$$V^t(k) = \min_{\sigma_p \in \Sigma_p} \max_{\sigma_e \in \Sigma_e} \left\{ \int_0^T g^s(\sigma_p, \sigma_e) ds + v^T(\sigma_p, \sigma_e) \right\}$$

where Σ_i represents the set of strategies for player i, g^s is the running cost (i.e., integral payoff) applied under strategies σ_p and σ_e, and v^T is the terminal payoff applied under strategies σ_p and σ_e.

Solutions to two-person zero-sum differential games are characterized as an n-dimensional surface providing a "barrier" between terminal conditions of the game. This surface is referred to as the "boundary of the usable part" (BUP) of the state space. If the current state of the game is on the BUP, and each player can play such that subsequent states remain on the BUP, then the game is in a state of equilibrium. Accordingly, the regions to

[1] The designations based on "pursuer" and "evader" have their roots in games of pursuit and evasion—the first games analyzed using differential game theory. Even differential games that do not correspond to pursuit games tend to use this notation. We also use this notation for all of the games discussed in this dissertation.

either side of this boundary are referred to as the "usable part" (UP) and the "nonusable part" (NUP) where the UP corresponds to the region where P is able to guarantee a win and the NUP corresponds to the region where E can guarantee a win.

Mathematically, let $\mathbf{x}$ be a set of state vectors for the game, and $\mathcal{T}$ be the target set of the game (i.e., the set of states in the game such that transitioning to a terminal state can be forced). Then,

$$\begin{aligned}
\text{UP} &\equiv \left\{ \mathbf{x} \mid \mathbf{x} \in \mathcal{T}, \exists \sigma_p^* \in \Sigma_p, \max_{\sigma_e \in \Sigma_e} (\zeta(\mathbf{x}, \sigma_p^*, \sigma_e) < 0 \right\} \\
\text{NUP} &\equiv \left\{ \mathbf{x} \mid \mathbf{x} \in \mathcal{T}, \exists \sigma_e^* \in \Sigma_e, \min_{\sigma_p \in \Sigma_p} (\zeta(\mathbf{x}, \sigma_p, \sigma_e^*) > 0 \right\} \\
\text{BUP} &\equiv \left\{ \mathbf{x} \mid \mathbf{x} \in \mathcal{T}, \exists \sigma_p^* \in \Sigma_p, \exists \sigma_e^* \in \Sigma_e, \max_{\sigma_e \in \Sigma_e} (\zeta(\mathbf{x}, \sigma_p^*, \sigma_e) = 0 = \min_{\sigma_p \in \Sigma_p} (\zeta(\mathbf{x}, \sigma_p, \sigma_e^*) \right\}
\end{aligned}$$

Under these definitions, σ_p^* and σ_e^* would be the optimal strategies for each player to play. The UP represents the region adjacent to the BUP reached at some previous point when E deviated from optimal play and now cannot return to equilibrium. In the same way, the NUP represents the region adjacent to the BUP reached at some previous point when P deviated from optimal play and now cannot return to equilibrium.

2.5 Pursuit Games

One class of differential games that has been studied extensively is the pursuit game. In the pursuit game, there are two players, a pursuer (denoted P) and an evader (denoted E). In the basic game, E is attempting to achieve some objective (e.g., E is a bomber attempting to reach a bomb site or a pedestrian attempting to cross a parking lot) and P

is attempting to prevent E from achieving that objective (e.g., P is a missile attempting to shoot down E or a mad driver of a car attempting to hit E.) The game terminates when P successfully captures E or when E guarantees that capture is impossible (perhaps by achieving its objective). Under this definition, the UP is a region surrounded by a lethal envelope of P, the NUP is a region around E such that capture is impossible, and the BUP is the surface of points separating these two regions.

One classic pursuit game studied in differential game theory is the *Homicidal Chauffeur* game. In this game, we can think of the playing field being an open parking lot with a single pedestrian crossing the parking lot and a single car. The driver of the car tries to run down the pedestrian and proceeds to pursue the pedestrian through the parking lot. Some of the constraints include that the game is played on a two dimensional playing field, the car is faster than the pedestrian, but the pedestrian is more maneuverable than the car. Generally, there are additional assumptions that both the car and the pedestrian are traveling at a fixed speed, the car has a fixed minimum radius of curvature, and the pedestrian is able to make arbitrarily sharp turns [39]. In the following, we present a summary of an analysis of this game by Isaacs [177] and Basar and Olsder [39].

In analyzing this game, it turns out that the solution is relatively simple and depends on four parameters—the speed of the car, the speed of the pedestrian, the radius of curvature of the car and the "lethal envelope" of the car (i.e., the distance between the car and the pedestrian that is considered to be "close enough" to hit the pedestrian). The

kinematic equations characterizing this game are given by

$$\begin{aligned}
\dot{x}_P &= v_P \sin\theta_P \\
\dot{y}_P &= v_E \cos\theta_P \\
\dot{x}_E &= v_E \sin\theta_E \\
\dot{y}_E &= v_E \cos\theta_E \\
\dot{\theta}_P &= \frac{v_P}{\rho_P}\phi_P = \omega_P\phi_P \\
\dot{\theta}_E &= \frac{v_E}{\rho_E}\phi_E = \omega_E\phi_E
\end{aligned}$$

where v_P is P's speed, v_E is E's speed, θ_P is the turn angle of P, θ_E is the turn angle of E, ρ_P is P's minimum radius of curvature, ρ_E is E's minimum radius of curvature, ω_P is P's angular velocity, and ω_E is E's angular velocity. Thus the game is controlled by ϕ_P and ϕ_E for P and E respectively.[2] Note, $|\phi_P| \leq 1$ and $|\phi_E| \leq 1$. Thus for either player, if $\phi_i = 0$, player i moves in a straight line. If $\phi_i = \pm 1$, player i makes its sharpest possible turn in the positive or negative direction.

To simplify analysis, we generally select a player as a point of reference and track relative positions. For the following, we consider play relative to P's position. In addition, Isaacs speaks of reducing the state space by collapsing the kinematic equations into this common frame of reference. If we center play on P's position, then we need only track E's position through the game. In this relative frame of reference, we have,

$$x = (x_E - x_P)\cos\theta_P - (y_E - y_P)\sin\theta_P$$

[2] The variables ϕ_P and ϕ_E are used to *parameterize* the game which in turn permits the reduced form of the game described later in the section.

$$y = (x_E - x_P)\sin\theta_P - (y_E - y_P)\cos\theta_P$$

$$\theta = \theta_E - \theta_P$$

Also, the equations of motion become,

$$\dot{x} = -\omega_P \phi_P y + v_E \sin\phi_E$$

$$\dot{y} = \omega_P \phi_P x - v_P + v_E \cos\phi_E$$

Note this makes the game separable, meaning the impact of the two players on the state of the game can be determined separately.

If we normalize the problem by making $v_P = 1$ and $\omega_P = 1$, then the kinematic equations become,

$$\dot{x} = v_E \sin\phi_E - y\phi_P$$

$$\dot{y} = v_E \cos\phi_E - 1 + x\phi_P$$

with new control variables, ϕ_P for P and ϕ_E for E.

Isaacs shows that, assuming optimal play by both players, if γ, the speed ratio $\frac{v_e}{v_p}$, is less than 1 (i.e., the car is faster than the pedestrian), and if

$$\frac{l}{\rho} > \sqrt{1-\gamma^2} + \sin^{-1}\gamma - 1$$

then capture *will* occur, where l is the lethal range. Otherwise, escape *can* occur. This is illustrated in Figure 2.2. To read this figure, note that the position of E does not appear explicitly. The coordinates are relative to P's position, and it is assumed P is facing to the

right. If E lies within the region designated by the UP, then capture is imminent. If E lies in the region designated by the NUP, the escape is guaranteed. The BUP indicated the boundary between these two states and need not be a simple line as shown in the figure.

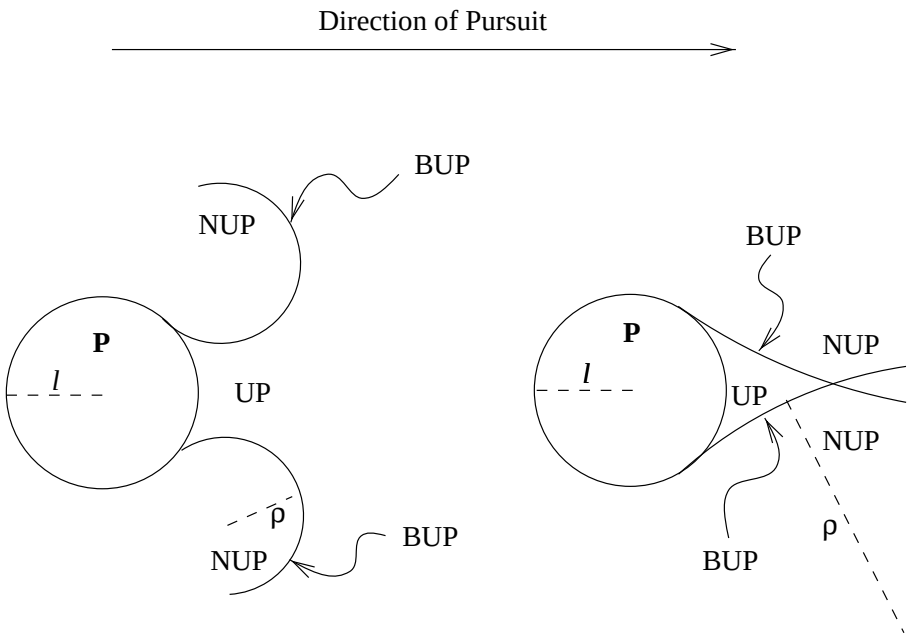
In this figure, if the above inequality holds, the UP has an infinite area directly in front of P, and P's greater speed guarantees P can force E into the UP. On the other hand, Figure 2.2 also shows the small, finite size of the UP which permits escape to occur where the inequality does not hold. Isaacs shows that the UP is defined by the equations

$$\begin{aligned} x(\tau) &= (l - v_E\tau)\cos(s + \omega_P\tau) + \rho \sin \omega_P\tau \\ y(\tau) &= (l - v_E\tau)\sin(s + \omega_P\tau) + \rho(1 - \cos \omega_P\tau) \end{aligned}$$

where τ represents time $T - t$ with the restriction that $0 \leq \tau \leq \frac{l}{v_E}$, and s is a point in the UP which is where capture occurs. What this means is if the ratio of the lethal range to the radius of curvature exceeds the maneuverability of the pedestrian at the designated speeds, then the pedestrian will be hit no matter what.

From this analysis, Basar and Olsder [39] determine optimal strategies for the players to ensure the state trajectories follow the BUP. They define a switch function $A_c(x, y) = xV_y(x, y) - yV_x(x, y)$ where $V_x(x, y)$ and $V_y(x, y)$ are x and y components of ∇V, a vector normal to the BUP, and V is defined as in Section 2.4. From this, we can denote (V_x, V_y) as a vector emanating from P's position. Then the optimal control for E is to select ϕ_E^* such that $(\cos\phi_E^*, \sin\phi_E^*)$ is co-linear with (V_x, V_y). So,

$$\cos\phi_E^* = \frac{V_x}{\sqrt{V_x^2 + V_y^2}}$$



Small radius of curvature Large radius of curvature

Figure 2.2: Solution configurations for the Homicidal Chauffeur game.

$$\sin \phi_E^* = \frac{V_y}{\sqrt{V_x^2 + V_y^2}}$$

Basar and Olsder point out further that normalizing ∇V reduces determining V_x and V_y to determining the direction of ∇V. Thus, $V_x = \sqrt{1 - v_E^2}$ and $V_y = v_E$. So, $\cos \phi_E^* = \sqrt{1 - v_E^2}$ and $\sin \phi_E^* = v_E$, yielding $\phi_E^* = \cos^{-1} \sqrt{1 - v_E^2}$. The optimal control for P is to choose ϕ_P^* such that

$$\phi_P^* = \begin{cases} \text{sgn}(A_c(x, y)), & A_c \neq 0 \\ \text{any admissible } \phi_P, & A_c = 0 \end{cases}$$

where $\text{sgn}(x) = \frac{|x|}{x}$.

2.6 Examples

In addition to the classical Homicidal Chauffeur game, a wide variety of differential games have been studied. Many pursuit games focus on fixed control parameters in the dynamics, such as speed, and only permit changes in turn angle or angular velocity. Prasad and Rajan considered the case where, in addition to controlling turn angle, the players had control over speed as well [267]. This complicates play by permitting multi-dimensional action spaces.

Many researchers limit studies to two dimensions, arguing that the complexities in two dimensions are, in themselves, worthy of study. Unfortunately, these cases are frequently insufficient to cover real-world problems which take place in three dimensions, or at least in two and a half dimensions. Ardema and Rajan examine the three-dimensional aircraft pursuit problem and focus on real-time characteristics of any feasible control law [17]. Such cases are closer to $2\frac{1}{2}$ dimensions given normal orientations of pilots in gravity.

Merz considered a true three-dimensional pursuit game by considering stochastic guidance of satellites in orbit, engaged in a pursuit-evasion conflict [230].

As an interesting discrete problem, similar to the Lady in the Lake [205], Bernhard *et al.* consider the Rabbit and Hunter game [47] under conditions of stationary and non-stationary dynamics. In both the Lady in the Lake and the Rabbit and Hunter games, one player is constrained to follow a boundary while the other player has free movement. In the Lady in the Lake, a lady is swimming in a lake and is being pursued by a man running around the perimeter of the lake. In the Rabbit and Hunter, a rabbit is trapped against a wall and makes random jumps back-and-forth while the hunter attempts to shoot the rabbit.

Other work in differential game theory is extending the number of players. Yavin describes a three-player game in which two agile players attempt to evade a single pursuer [376]. At the same time, the evaders want to shoot down the pursuer. This is also an example of the two-target game where a player has two competing objectives [141, 322]. Imado and Ishihara also consider the case where two missiles attempt to shoot down an airplane [175]. Lai and Tanaka consider general n-person games where each player attempts to force opponents into a terminal location in the playing field [200].

Related to work in partially observable Markov decision processes [70, 71, 180], Galperin and Skowronski are studying games in which noise is introduced into game dynamics [134]. Corless *et al.* consider the case where state information is uncertain [88]. Chan and Ng consider partial observability in linear-quadratic games [72], and Yavin considers

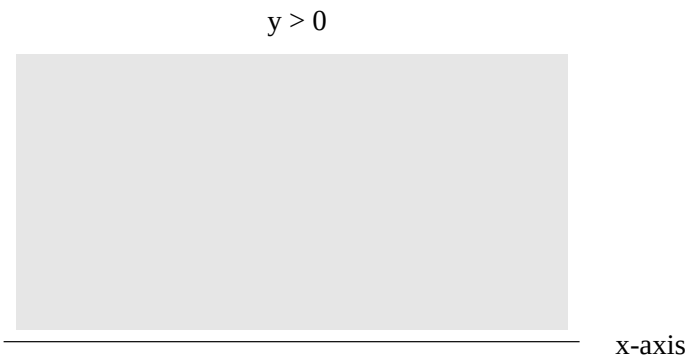


Figure 2.3: Playing field for a simple differential game.

the case where the process of observing the state is deceptive or is somehow interrupted [375].

2.7 A Simple Differential Game

To illustrate the problems associated with solving (and ultimately learning) differential games, we provide a simple example of players attempting to move an object in the plane. This game is described in detail in [205]. The playing field is shown in Figure 2.3. In this game, there are two players, P and E. An object is placed in the plane somewhere above the x-axis (i.e., $y > 0$) at a random location and begins to drop toward the x-axis. The two players exert opposing forces on the object in an attempt to control where the object crosses the x-axis. In this game, P is attempting to minimize x while E is attempting to maximize x. The dynamics of the game force the object to fall at all times, i.e., the object cannot be suspended indefinitely above the x-axis.

To solve this game, we begin by examining the equations of motion (i.e., the

kinematic equations) which are given as,

$$\begin{aligned} \dot{x} &= Av + B\sin u \\ \dot{y} &= -1 + B\cos u \\ |v| &\leq 1 \\ u &\in [0, 2\pi) \end{aligned}$$

where A and B are parameters defining the dynamics of the game, and u and v are the controls set by P and E respectively. We assume the game has a terminal payoff equal to the value of x at the point the object crosses the x-axis.

To understand the dynamics of this game, we note that the kinematic equations can be rewritten as a vector equation, separating the impact of the two players on the outcome. When such separation is possible, the differential game is said to be a *separable game*. Specifically, we see

$$\dot{\mathbf{x}} = ((B\sin u)\hat{\mathbf{e}}_x + (B\cos u)\hat{\mathbf{e}}_y) + (Av\hat{\mathbf{e}}_x - \hat{\mathbf{e}}_y)$$

where $\hat{\mathbf{e}}_x$ and $\hat{\mathbf{e}}_y$ are unit vectors in the directions of the x- and y-axes respectively. Notice that player P dominates the first term of the equation (i.e., the value of the first term is fully determined by the action taken by P), and player E dominates the second term (i.e., the value of the second term is fully determined by the action taken by E).

From this vector equation, we can construct *vectograms* describing how each player affects the movement of the object. These vectograms are given in Figure 2.4. In this figure, the circle on the left is P's vectogram and indicates that P is able to determine the direction

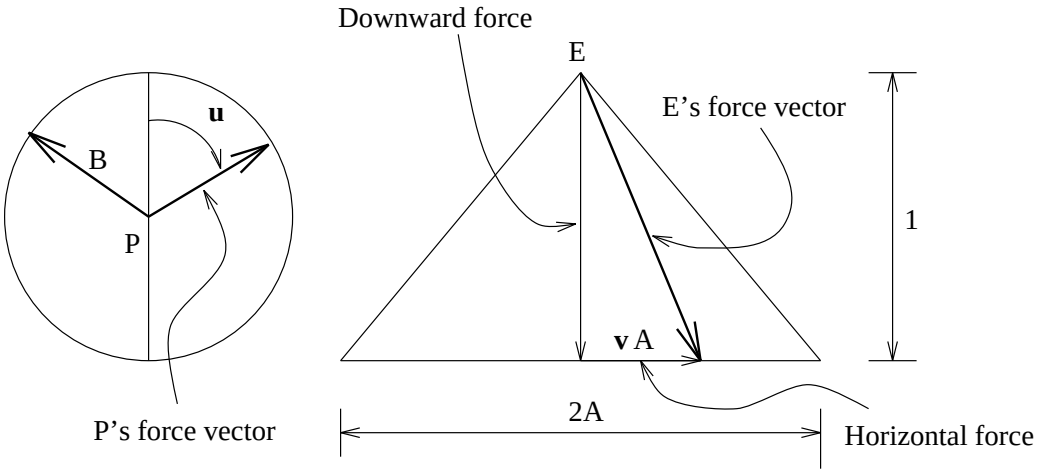


Figure 2.4: Vectograms for play in the simple vector game.

of the force vector to be applied against the object, but not the magnitude. This direction is indicated by $\mathbf{u}$, and the magnitude is given by B. The triangle on the right is E's vectogram and indicates that E is able to determine the magnitude of the force vector to be applied against the object, but not the direction. The magnitude varies from -1 to 1, but the direction is determined by combining a downward force of fixed magnitude and a horizontal force determined by the control variable $\mathbf{v}$.

We are now in a position to "solve" the game. Given E wants to maximize x, it should be clear from the vectograms that E would select $v^* = 1$. This has the effect of applying the greatest force along the x-axis in the positive direction. The force in the y direction is fixed and negative, so we know the game will ultimately terminate by crossing the x-axis.

Determining the optimal strategy for P is less clear because P wants to minimize x

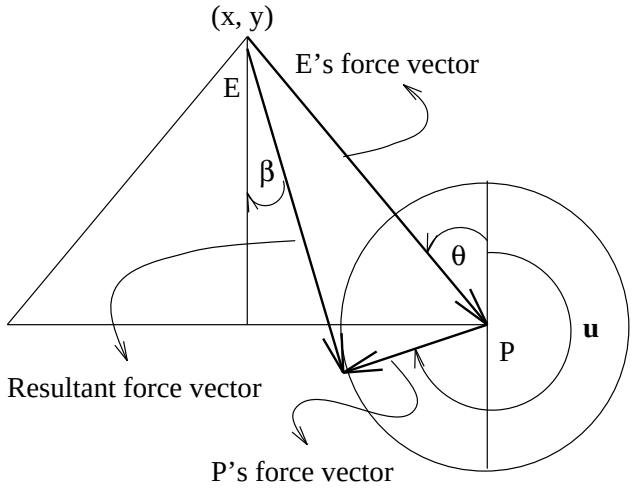


Figure 2.5: Combined vectogram in the simple vector game.

and can only provide a "direction" for the force to be applied; magnitude is predetermined by the kinematic equation and is independent of direction. Because this is a separable game (and can therefore be represented with vectograms), let us combine the vectograms to determine the optimal play for P (Figure 2.5). For this problem, we want to find u^* such that we minimize the inclination β of the trajectory with respect to the y-axis.

Let $\hat{\mathbf{f}}_E$ be E's force vector, and let $\hat{\mathbf{f}}_P$ be P's force vector. Let $\hat{\mathbf{f}}$ represent the resultant force vector such that $\hat{\mathbf{f}} = \hat{\mathbf{f}}_E + \hat{\mathbf{f}}_P$. We can see that the magnitude of $\hat{\mathbf{f}}_E$ is $\sqrt{1 + A^2}$. Since $\hat{\mathbf{f}}$ is at a tangent to P's control circle (which is required in order to minimize β), we know that the angle between $\hat{\mathbf{f}}$ and $\hat{\mathbf{f}}_P$ is $\frac{\pi}{2}$. So the angle between $\hat{\mathbf{f}}_E$ and $\hat{\mathbf{f}}_P$ is $\cos^{-1} \frac{B}{\sqrt{1+A^2}}$. To find u^*, we note that we must subtract the angle between $\hat{\mathbf{f}}_E$ and $\hat{\mathbf{f}}_P$ from 2π. Then we must subtract θ from this result to get u^*. Note $\theta = \frac{\pi}{2} - \cos^{-1} \frac{A}{\sqrt{1+A^2}}$. So,

$$\begin{aligned} u^* &= 2\pi - \cos^{-1}\frac{B}{\sqrt{1+A^2}} - \left(\frac{\pi}{2} - \cos^{-1}\frac{A}{\sqrt{1+A^2}}\right) \\ &= \frac{3\pi}{2} + \cos^{-1}\frac{A}{\sqrt{1+A^2}} - \cos^{-1}\frac{B}{\sqrt{1+A^2}} \end{aligned}$$

Given the optimal strategies u^* and v^*, we can also compute the value of the game by following the resulting trajectory. For some initial point of play (x_0, y_0), we can compute the payoff to be,

$$\begin{aligned} V(x_0, y_0) &= x_0 + \frac{y_0(A + B\sin u^*)}{1 - B\cos u^*} \\ &= x_0 + \frac{y_0\left(A - B\cos\left[\cos^{-1}\frac{A}{\sqrt{1+A^2}} - \cos^{-1}\frac{B}{\sqrt{1+A^2}}\right]\right)}{1 - B\sin\left[\cos^{-1}\frac{A}{\sqrt{1+A^2}} - \cos^{-1}\frac{B}{\sqrt{1+A^2}}\right]} \end{aligned}$$

2.8 A Simple Pursuit Game

Now consider the following simple pursuit game. Assume we have two players, P and E, where P is trying to capture E. The game area (i.e., the state space) is limited to the plane, $\Re^2$. Each player can choose their instantaneous heading of ϕ_P and ϕ_E respectively. Assume the speeds of P and E (v_P and v_E respectively) are constant and that the speed of P is greater than the speed of E. From this description, we can write the equations of motion as

$$\begin{aligned} \dot{x}_P &= -v_P \sin\phi_P \\ \dot{y}_P &= v_P \cos\phi_P \end{aligned}$$

$$\dot{x}_E = -v_E \sin \phi_E$$

$$\dot{y}_E = v_E \cos \phi_E$$

where $\langle x_P, y_P \rangle$ and $\langle x_E, y_E \rangle$ are instantaneous positions of P and E respectively. Finally, we assume that P has a lethal envelope, $l > 0$, such that P captures E if

$$\sqrt{(x_P - x_E)^2 + (y_P - y_E)^2} \leq l$$

As described previously, it is convenient to consider relative positions of P and E. Without loss of generality, we consider the state of the game relative to P, yielding

$$y = y_E - y_P$$

$$x = x_P - x_E$$

We can also consider the speed ratio,

$$\alpha = \frac{v_P}{v_E}$$

The new kinematic equations then become,

$$\dot{y} = \cos \phi_E - \alpha \cos \phi_P$$

$$\dot{x} = \sin \phi_E - \alpha \sin \phi_P$$

The kinematic structure of this game (without relative coordinates) is shown in Figure 2.6.

Although shown in the first quadrant in Euclidean space, either player can appear in any quadrant. After converting to relative coordinates, the kinematic structure becomes that shown in Figure 2.7.

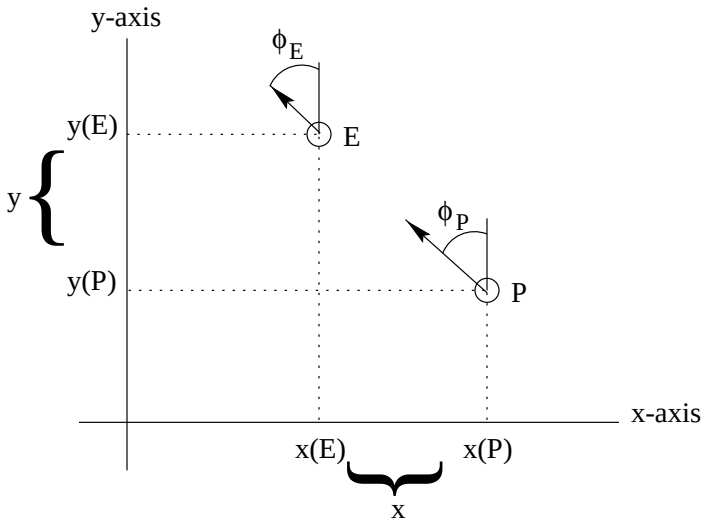


Figure 2.6: Kinematic structure of simple pursuit game.

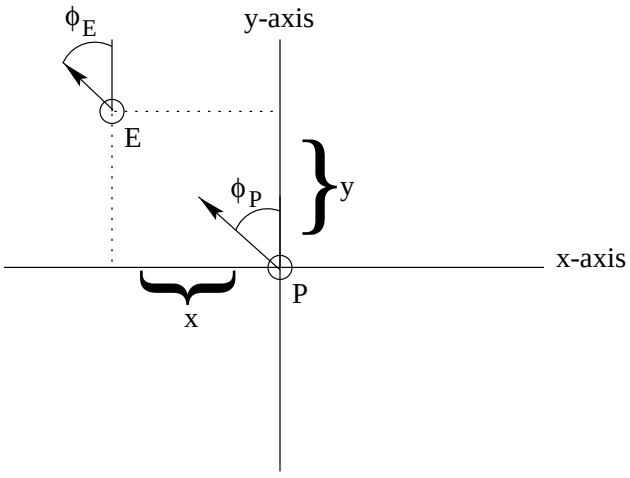


Figure 2.7: Kinematic structure of simple pursuit game in relative coordinates.

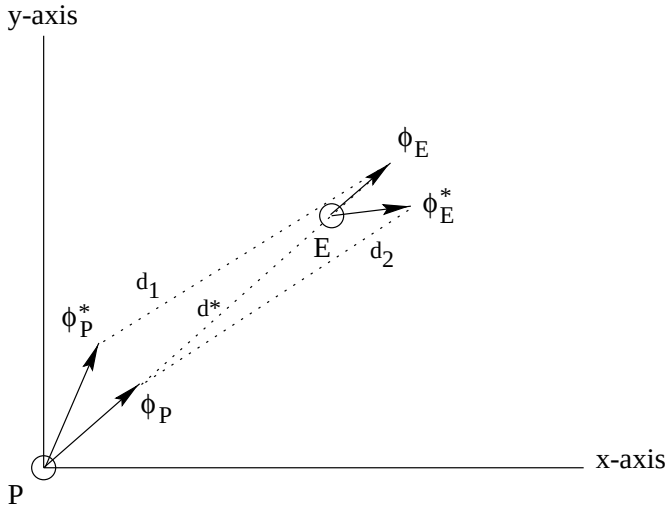


Figure 2.8: Effects of deviating from optimal strategy in simple pursuit game.

In this game, P is always able to capture E, and the target set for the game is defined as $\{(x, y)|x^2 + y^2 \leq l^2\}$. For this game, the optimal strategies for P and E are identical. In particular,

$$\phi_P = \phi_E \ \arctan \frac{x}{y}$$

To prove these strategies are optimal, we need to consider another strategy for each player. This would correspond to considering unilateral defections as described in Section 2.1. The strategies for P and E select ϕ_P^* and ϕ_E^* such that P and E travel along the line connecting them. Therefore, intuitively, we can see the effect of deviating from this line (Figure 2.8).

When considering Figure 2.8, we can see that while P may continue to gain on E, if P deviates from traveling directly toward E, P fails to close as quickly. In other words, $d^* < d_1$, where d^* is the distance between E and P if playing optimally. Further, if E deviates from traveling directly away from P, then P will close more quickly. In other words, $d^* > d_2$.

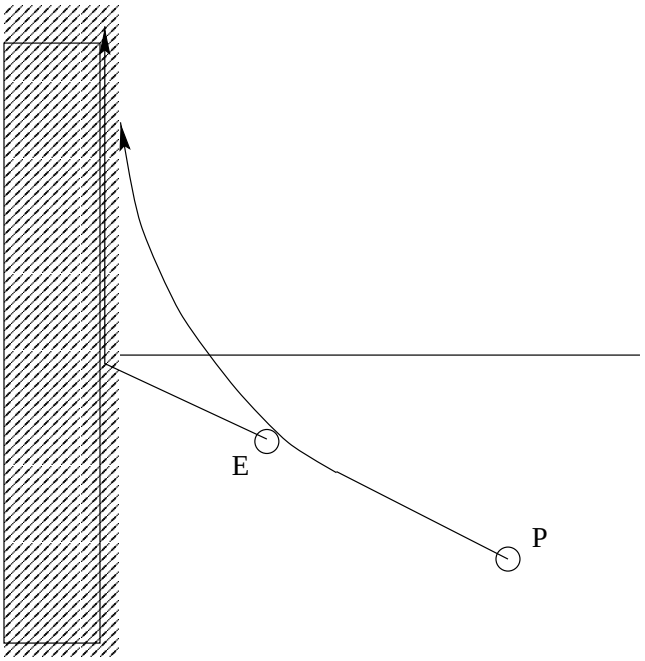


Figure 2.9: Optimal strategy for simple pursuit in the half plane.

If we extend the game such that there is a boundary along $x = 0$ (i.e., the game occurs in the half plane), then the optimal strategy for P and E is the same until E and P approach the boundary. When this occurs, the previous strategy is applied until E reaches the boundary. At this point, E immediately turns to follow the boundary away from P. P continues to aim directly at E; no sudden turn to follow the boundary will be required since P will smoothly transition to following the boundary while following E (Figure 2.9).

Throughout this dissertation, we consider several games, including the simple game of force, the simple pursuit game, and the simple pursuit game in the half plane. In addition, three more complex games are considered including a variant of the Homicidal Chauffeur game where P has a fixed, complex strategy, a variant of the Homicidal Chauffeur with two pursuers, and an extension of the Homicidal Chauffeur where both P and E have restricted mobility.

Chapter 3

Machine Learning and Games: A Review

3.1 Learning Game Strategies: The Problem

Although games have been a popular topic for study in artificial intelligence and machine learning, little research has been done in multi-agent learning and games. In fact, only limited work has been performed in multi-agent learning and control; however, this is becoming a topic of great interest in the reinforcement learning community. Until recently, the majority of the work was performed within the context of distributed artificial intelligence and artificial life.

As with other problems in reinforcement learning, learning strategies for game playing can be posed as a control problem. The object is for an agent (or player) to learn the "best" or "optimal" strategy to use against its opponent. In the context of two-player games where one player is applying a fixed strategy, the second player optimizes its strategy

to yield maximum payoff in the game. When both players are attempting to learn, each must be sensitive to the fact the opponent's strategy is not fixed. Thus, what may be "optimal" in one context (i.e., with a particular strategy applied) will not necessarily be optimal in another context. Of course, if the expected payoffs are known for the various joint strategies, then "learning" reduces to solving the game.

Most researchers exploring machine learning and games are not interested in finding new methods for solving games in game theory. Instead, they focus on problems such as the following:

- *Expected payoffs are not known.* In a normal form game, this is equivalent to filling out the entries in the payoff matrix. In an extensive form game, this could involve learning the heuristic evaluation function to be applied at the interior nodes of the game tree [123, 140, 155, 156, 157, 204, 258, 259, 260, 261, 286].

- *Opponent capabilities are not known.* In most studies in game theory, each player knows the permissible actions for itself and its opponent. In problems such as differential games, this may not be true. For example, a pursuer may have a long range and be able to pursue the evader for a long period of time, but the evader may assume (given the nature of the encounter) that the range is more limited. This problem is addressed through techniques such as *opponent modeling* [68, 67, 69, 243].

- *The game environment is not fully described.* This differs from games of imperfect information in that the environment can be learned from experience. Thus, the in-

formation sets associated with various states in the game can vary over time as the players learn the environment. This is analogous to exploring the environment in navigation problems [208, 209, 219, 238, 239].

- *The game dynamics are uncertain* [134]. This is analogous to the problem of learning an opponent's strategy except that the dynamics, usually, are fixed. Learning game dynamics when the dynamics are not fixed is also of interest [152, 165].

In this chapter, we review recent work in learning game strategies. Although each of the above issues are touched upon, the focus of this chapter is on reviewing learning algorithms designed to learn strategies in two person games. We begin with the problem of a player learning a strategy to apply against a fixed opponent and then consider the research addressing problems where both players are learning. Finally, we discuss research in related areas within artificial intelligence such as distributed artificial intelligence, artificial life, and multi-agent systems.

3.2 Learning Methods and Markov Decision Processes

In the previous chapter, we introduce the problem of solving Markov decision processes (MDPs). We provided two standard approaches to solving MDPs—value iteration and policy iteration. These methods require (in their basic form) full sweeps of the state-action space to find optimal policies. For problems with large state or action spaces, these approaches become impractical.

Research in learning and MDPs has focused on developing approaches to finding optimal policies in MDPs when the state or action spaces becomes large. Many of the algorithms are derivatives of the value and policy iteration algorithms but focus on sweeping only parts of the space. Other algorithms apply ideas from reinforcement learning to function approximators and genetic algorithms to learn the policies. In the following sections, we describe several of these algorithms.

3.2.1 Q-Learning and Dynamic Programming

The basic algorithm for value iteration was given in Chapter 2 and consists of repeated application of the following

$$\forall s \in \mathcal{S}, a \in \mathcal{A}, V_t(s) = Q_t^V(s, \pi_t(s))$$

where

$$Q_t^V(s, \pi(s)) = \min_{a \in \mathcal{A}} Q_t^V(s, a)$$

$$Q_t^V(s, a) = c(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) Q_t^V(s', \pi(s'))$$

until

$$\max_{s \in \mathcal{S}} |V_t(s) - V_{t-1}(s)| < \epsilon$$

Applying the procedure directly is a synchronous approach to dynamic programming where all state-action paris are updated at each time step.

Barto *et al.* [35, 37, 34] describe two alternative approaches to solving an MDP that appear to be less computationally expensive and are both forms of *asynchronous* dy-

namic programming. The first approach, which they refer to as "asynchronous dynamic programming," is a derivative of Gauss-Seidel dynamic programming [48] in which, for each state s and each time step $t = 0, 1, \ldots$, update $V_t(s)$ as follows:

$$\begin{aligned} V_t(s) &= \min_{a \in \mathcal{A}} \left[c(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s') \right] \\ &= \min_{a \in \mathcal{A}} Q_t^V(s, \pi(s)) \end{aligned}$$

where

$$V(s') = \begin{cases} V_t(s'), & \text{if } s' \prec s \\ V_{t-1}(s'), & \text{otherwise} \end{cases}$$

and "$\prec$" indicates an ordering relation on the states.

As in Gauss-Seidel dynamic programming, asynchronous dynamic programming does not update all states simultaneously. However, where Gauss-Seidel dynamic programming still performs a systematic "sweep" of all states, asynchronous dynamic programming allows states to be updated at arbitrary points in time. When a state is updated, it uses the current values of successive states.

The second approach is called real-time dynamic programming (RTDP) because it provides an on-line learning strategy rather than the traditional off-line strategies of other dynamic programming algorithms. RTDP applies a greedy strategy with respect to the current estimate of $V(s)$, $\hat{V}(s)$, to define the policy for the controller. Whenever an action is taken, the cost/payoff of that action is applied immediately to update $\hat{V}(s)$. To ensure convergence of RTDP, it is necessary to visit all states "infinitely" often. One approach used to ensure this is to require that all states be selected infinitely often as initial states.

One of the problems with methods such as asynchronous dynamic programming and RTDP is that these methods require a complete understanding of the transition probabilities, $p(s'|s,a)$, underlying the MDP [35]. They also require knowledge of the immediate costs, $c(s,a)$. The requirement to know the cost holds, in particular, in the off-line case but can be relaxed when learning on line as in RTDP. In many control tasks, such as the differential games studies in this dissertation, such knowledge may not be available.

A conceptually simple approach to solving MDPs with incomplete knowledge was proposed by Watkins in 1989 [363, 364] called Q-learning. As with traditional value iteration methods, Q-learning can be performed both off line and on line. In Q-learning, the controller maintains estimates of the optimal Q values for each admissible state-action pair. These Q values are estimated based on experience applying admissible actions in each state, rather than based on an evaluation function that includes the state-transition probabilities. In off-line Q-learning, as in off-line dynamic programming, a successor function is defined that provides a new state, s', with probability $p(s'|s,a)$. Obviously, this successor function must have knowledge of the underlying probabilities. In on-line Q-learning, however, an explicit successor function is not required since the actual system provides the successor states.

During control, the controller keeps track of the succession of states visited, the actions taken in each state, and the costs incurred as a result of taking the actions in each state. Either during control or upon termination, the Q values are updated as follows. Let $Q_t(s,a)$ be the Q value at time t when action a is performed in state s. Then this Q value

is updated by computing

$$Q_{t+1}(s,a) = [1 - \alpha_t(s,a)]Q_t(s,a) + \alpha_t(s,a)[c(s,a) + \gamma Q_t(s', \pi(s'))]$$

where $\alpha_t(s,a)$ is the value of the learning rate for state-action pair $\langle s,a \rangle$ at time t, γ is the discount rate, $Q_t(s', \pi(s')) = \min_{a \in \mathcal{A}} Q_t(s',a)$, and s' is the successor state.

Note that a learning rate is associated with each state-action pair rather than providing a single learning rate for the entire update process. Note further that the learning rate is not constant. Typically, $0 \leq \alpha_t(s,a) < 1$ and $\alpha_t(s,a)$ decreases over time. Specifically, $\alpha_t(s,a)$ is changed only when action a is applied in state s. For convergence, the schedule for changing $\alpha_t(s,a)$ must conform to the following requirements [363, 364]. For all $s \in \mathcal{S}$ and $a \in \mathcal{A}$,

$$\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t(s,a)^2 < \infty$$

One schedule that satisfies these requirements, proposed by Barto *et al.* [35], is

$$\alpha_t(s,a) = \frac{\alpha_0 \tau}{\tau + n_t(s,a)}$$

where α_0 is the initial learning rate (applied to all state-action pairs), τ is a user-defined parameter, and $n_t(s,a)$ is the number of times $Q(s,a)$ has been updated at time t.

Given the algorithm presented above, with the associated requirements on the learning rate, Watkins proved that, assuming each state-action pair is visited (and updated) infinitely often, $Q(s,a)$ will converge to the optimum Q values with probability one. To

ensure that each action is considered at each state, a method must be applied where actions are selected other than according to the Greedy policy. One commonly applied approach is to assign a probability distribution to the admissible actions in the current state. Frequently, the Boltzmann distribution is used.

$$p(a|s) = \frac{e^{-Q_t(s,a)/T}}{\sum_{a \in \mathcal{A}} e^{-Q_t(s,a)/T}}$$

where T is a parameter controlling the shape of the distribution. As T increases, the distribution approaches uniform. As T decreases, the probability of selecting an action according to the Greedy policy approaches one. Usually, T is defined to decrease over time, similar to the way $\alpha_t(s,a)$ is defined.

3.2.2 Memory-Based Reinforcement Learning

In the previous section, we described learning algorithms based on dynamic programming which assume successive updates to expected payoff are made to all state-action pairs. Because of the requirement that all state-action pairs be updated, the natural representation for these algorithms is a lookup table. Unfortunately, when problems have extremely large state or action spaces, lookup tables are no longer practical.

For very large state or action spaces, many researchers in reinforcement learning are using techniques in function approximation. One approach to function approximation—memory-based learning—stores a set of examples as representatives of regions in the state-action space. During control, current conditions are matched against the stored examples and actions are derived from the actions associated with these examples. In the simplest

case, a memory base consists of several state-action pairs. When faced with a new state, the example with the state "nearest" to the current state is found, and the associated action is selected. Learning consists of storing state-action pairs that lead to successful control in the past.

Although conceptually simple, this approach to memory-based control has several difficulties. First, selecting the examples to be stored in the first place is non-trivial. Unless an oracle or teacher is available to provide good actions in response to various states, the initial memory base is likely to be populated by trial-and-error. This leads to the second difficulty. Just because a sequence of events leads to a successful outcome, this result is not sufficient to infer that all of the actions taken were, in fact, the best or even good. Storing a large number of intermediate state-action pairs that are not strong positive examples could limit the ultimate performance of the controller. Thus a mechanism for evaluating and, perhaps, editing the examples in the memory base is required. Third, choosing from the set of previously stored actions limits the ability to generalize, especially if operating in a large action space. Then it becomes important to be able to interpolate between actions within a region.

One approach to combating these problems has been proposed by Atkeson, Moore, and Schaal [26, 27, 25, 110, 237]. In their approach, they apply local weighted regression among the examples to determine the proper action. The memory base consists of a set of triples, $\langle s, a, b \rangle$, where s represents the state, a is the action, and b is the "behavior" or the outcome of taking action a in state s. In traditional reinforcement learning, b would

correspond to the expected payoff of taking action a in state s and then performing optimally from that point forward.

In Atkeson's approach, an "inverse" control model is used to determine the desired action [22, 23, 24]:

$$a = f^{-1}(s, b)$$

where, typically, the control model is represented as

$$b = f(s, a)$$

This latter form is referred to as the "forward" model. Learning consists of using experience to develop an approximation to f, $\hat{f}$. If the inverse model is available (i.e., $\hat{f}^{-1}$), then this model can be used for control by matching the current state and desired outcome (using maximum expected payoff or minimum expected cost) with the examples in the memory base and, if necessary, interpolating among the available actions.

Moore suggests that inverse models, although somewhat natural, can lead to problems if there is not a one-to-one mapping between actions and behaviors or if there are noisy examples in the memory base [26, 237]. As a result, he proposes working directly with the forward model. Using the memory base then consists of searching through available actions (in a given state) until the desired outcome is found.

Given either of these approaches to memory-based control, we still need an approach for learning the memory base in the first place. As already discussed, simply storing examples as they are experienced is not a good idea. Two issues must be addressed. First,

examples need to be chosen to represent the underlying search space. Second, behaviors must be learned which will lead to the development of the learned control model, $\hat{f}$.

As mentioned earlier, random exploration can provide an initial set of examples for the memory base. If we also learn resulting behaviors, then it is possible for this naive approach to lead to strong behavior. The disadvantage to the approach is that areas of the search space may be oversampled, leading to an increased search burden, and other areas may be undersampled, leading to degraded performance when in those regions.

Several researchers have investigated approaches to varying the resolution of the memory base according to the requirements of the problem [104, 110, 240, 312]. For example, Moore and Atkeson's parti-game algorithm uses the concept of an adversary attempting to thwart search to determine how to explore the search space [240]. In the current version of parti-game, the problem to be solved is limited to the case where there exists a known goal region rather than a reward function. This limits the class of problems to which parti-game can be applied, but the algorithm provides a powerful approach to partitioning the space.

Initially, parti-game defines a partitioning of the space and attempts to determine the shortest path from each partition to the partition containing the goal. This can be done using an all-pairs shortest path algorithm such as Dijkstra's algorithm [89]. Because this strategy alone can lead to situations where the controller gets hopelessly stuck, parti-game approximates the number of steps to the goal based on the worst-case scenario. In other words, in computing the shortest path, parti-game assumes an adversary can place the controller in the worst position within a partition from which to reach the goal. In such a

case, it is possible that the adversary could place the controller in a position from which there is no path to the goal.

For the above procedure to work, the controller must have a detailed model of the environment. In general, such a model of the environment may not be available, so Moore and Atkeson extend parti-game to the case where the adversary only considers observed experience. Note that the controller may still end up in a partition with no exit. At this point, parti-game decides the partition should be subdivided under the premise that not all areas within the partition lead to the controller getting stuck.

To partition the space further, parti-game collects all partitions identified as "losers" that are adjacent to non-losing partitions. A losing partition is one in which the controller can get stuck. Parti-game also collects all non-losing neighbors to these partitions. The algorithm then splits each of the partitions in half along the long axis and recomputes whether or note the partitions are losers.

In addition to generating the examples, expected behaviors must also be learned. Within the memory-based framework, behaviors (or outcomes) can correspond to the estimate of expected payoff. In this case, any of the real-time dynamic programming or temporal difference methods can be applied to update these behaviors [16, 43, 44, 53, 210, 263, 264, 265]. In fact, this strategy is used with the algorithms in this dissertation and is discussed in Chapters 6 and 7.

3.2.3 Temporal Difference Learning

In reinforcement learning, considerable attention has been given to Sutton's Temporal Difference Learning algorithm [76, 77, 78, 103, 104, 105, 155, 156, 157, 178, 232, 280, 332, 333, 334, 342, 343, 344, 350]. This approach focuses on the problem of predicting expected discounted payoff from a given state. It has been shown that Q-learning is a special form of temporal difference learning where the "look-ahead" is cut off. Specifically, Q-learning is shown to be equivalent to TD(0) when there exists only one admissible action in each state [35, 103].

The temporal difference method is intended to be applied in "multi-step prediction problems" where payoff is not awarded until several steps after a prediction for payoff is made. This is exactly the problem that arises with delayed reinforcement. At each step, an agent predicts what its future payoff will be, based on several available actions, and chooses its action based on the prediction. However, the ramifications for taking the sequence of actions are not revealed until (typically) the end of the process.

According to Sutton [332], the temporal difference method can be considered as an extension of the prototypical supervised learning rule based on gradient descent. If we assume a prediction depends upon a vector of modifiable weights w and a vector of state variables x, then supervised learning uses a set of paired state vectors and actual outcomes to modify the weights to reduce the error between the predictions and the known outcomes.

Let α be a learning rate (as in Q-learning), z be the value of the actual payoff, and P_t be the predicted payoff at time t. Then the supervised learning update procedure

can be represented as

$$\Delta w_t = \alpha(z - P_t)\nabla_w P_t$$

where $\nabla_w P_t$ is the gradient of P_t with respect to the weight vector w.

This method works best for single-step prediction problems. For multi-step prediction, the vector w cannot be updated until the end of a sequence, and all observations and predictions must be remembered until the end of the sequence. Sutton's temporal difference method permits incremental update and is based on the observations that

$$z - P_t = \sum_{k=t}^{m}(P_{k+1} - P_k)$$

where $P_{m+1} = z$. In this case, the supervised learning rule becomes,

$$\Delta w_t = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t}\nabla_w P_k$$

(see [332] for derivation). This update can be computed incrementally because it depends only on a pair of successive predictions (P_t and P_{t+1}), and on the sum of past values for $\nabla_w P_t$.

Sutton goes on to describe a family of temporal difference methods based on the influence past updates have on the current update of the weight vector. These methods are based on a parameter, $\lambda \in [0, 1]$, which specifies a discount factor in the prediction equation (see Section 2.2), and refers to the family as the TD(λ) family. When $\lambda = 0$, past updates have no influence on the current update. When $\lambda = 1$, all past predictions receive equal weight. Assuming it is desirable for the update procedure to be more sensitive to recent predictions than to distant predictions, the changes are weighted according to λ^k. Thus the

update equation becomes

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

When $\lambda = 0$, this equation reduces to

$$\Delta w_t = \alpha(P_{t+1} - P_t) \nabla_w P_t$$

Given the nature of the TD(λ) algorithm, the most common application of the algorithm is for training neural networks. For TD(0), in addition to neural networks, the approach has been applied (through Q-learning) to lookup tables and memory-based approaches.

3.2.4 Genetic Algorithms

The final approach to learning in Markov decision processes that we discuss here is the use of genetic algorithms (GAs) to learn sequential decision rules. As with other methods of learning to solve MDPs, GAs must associate sets of admissible actions to states in the problem and provide a means to select the most appropriate action to maximize expected payoff.

The first application of GAs to MDPs arose from work in classifier systems [50, 149, 167, 168]. In a classifier system, rules are binary condition/action pairs that work within a message-passing architecture (Figure 3.1). State information is converted into a binary message and placed on the message list. The messages on the message list are matched against the condition part of all of the rules (or classifiers) in the system. All rules that match a particular message compete to fire. When a rule fires, the action part of the

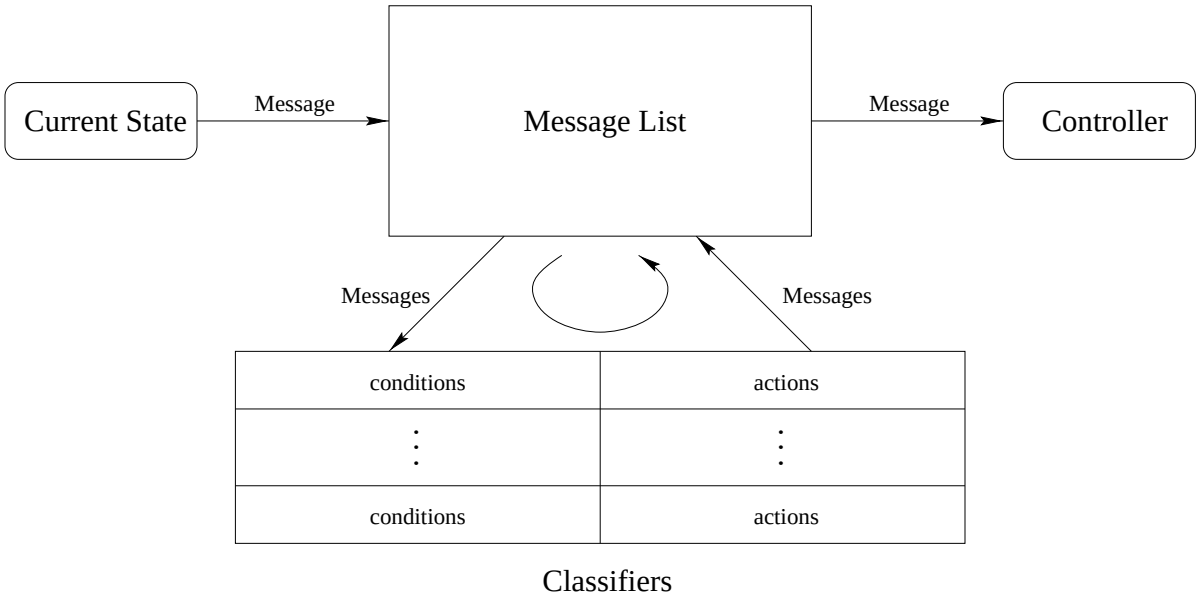


Figure 3.1: Standard architecture of a classifier system.

rule defines a new message to be put on the message list, and the old messages are removed. The cycle continues until a message appears on the message list that can be translated into a set of actions to be applied to the environment. These actions change the state, and the cycle repeats.

Learning takes place in two steps. First, each rule has an associated value (e.g., expected payoff) that represents strength and is used in competition. These values indicate the fitness of the rules and are also used by the genetic algorithm in selection. The values are updated using a reinforcement learning procedure called the *bucket brigade*. In the bucket brigade, when rules match the current message list, they issue a bid to fire, $B(C,t)$. Specifically,

$$B(C,t) = bR(C)s(C,t)$$

where b is a fraction of the strength to be bid, $s(C,t)$ is the strength of the classifier C at time t, and $R(C)$ is the "specificity" of C, defined to be the number of non-don't-cares in the condition part of C divided by the length of the condition part of C. Should C win the competition, the strength is updated as

$$s(C,t+1) = s(C,t) - B(C,t) + aB(C',t)$$

where C' is the classifier that put the message on the message list in the previous time step causing C to fire, and a is the number of classifiers that fired in the current time step.

The second learning approach is the genetic algorithm. Classifiers are modified in a series of generations where individual classifiers are selected to be duplicated with probability in proportion to their fitness. Once a new set of classifiers is selected, certain classifiers are modified through mutation and crossover. Each "bit" in a classifier can take on a value from the alphabet $\{0, 1, \#\}$, where "#" indicates a "don't care." During mutation, a bit is selected and changed at random. During crossover, two classifiers are selected and random substrings of the two classifiers are swapped. Through mutation and crossover, new classifiers are introduced to permit exploration of the space of possible classifiers.

Grefenstette *et al.* [153] propose an alternative approach to learning sequential decision rules using a more general production system architecture which they call CPS (Competitive Production System). In their approach, rules are not limited to binary classifiers but have attributes of several types incorporated into the conditions and actions of several types replacing messages. There is no message list in CPS. Instead, the current state is mapped into the forms needed by the rule conditions, rules compete based on how

well they match the current state and based on strength, and actions have an immediate effect on the environment. In a sense, traditional classifier systems are analogous to "deliberative agents" where the agents may take multiple reasoning steps before selecting an action. CPS, on the other hand, is analogous to a "reactive agent" in which a decision is made immediately based on the current state.

CPS is folded into a two-part learning system similar to the two-part learning of a classifier system. First, a credit-assignment subsystem updates strengths associated with all of the rules. Instead of using the bucket brigade, Grefenstette uses an approach called "profit sharing" in which mean profit and an estimate of the variance of the payoff is updated (see Section 4.3.3 for more details). The rule strength is then the difference between the mean and the variance.

Second, a genetic algorithm generates rule sets. This approach represents the second significant departure from traditional classifiers (the first being the more "natural" rule form). Rather than operating on individual rules (except for mutation), the GA maintains the integrity of the rules and works on rule sets. A set of rules is combined into a tactical plan, and the GA operates on a population of plans. New plans are produced by selecting rules from parent plans and by mutating conditions or actions on individual rules. Crossover between two rules is not permitted.

The complete learning system with CPS, the credit-assignment subsystem, the GA, and a domain-specific simulator is called SAMUEL (Strategy Acquisition Method Using Empirical Learning). The basic architecture of SAMUEL using several different learning

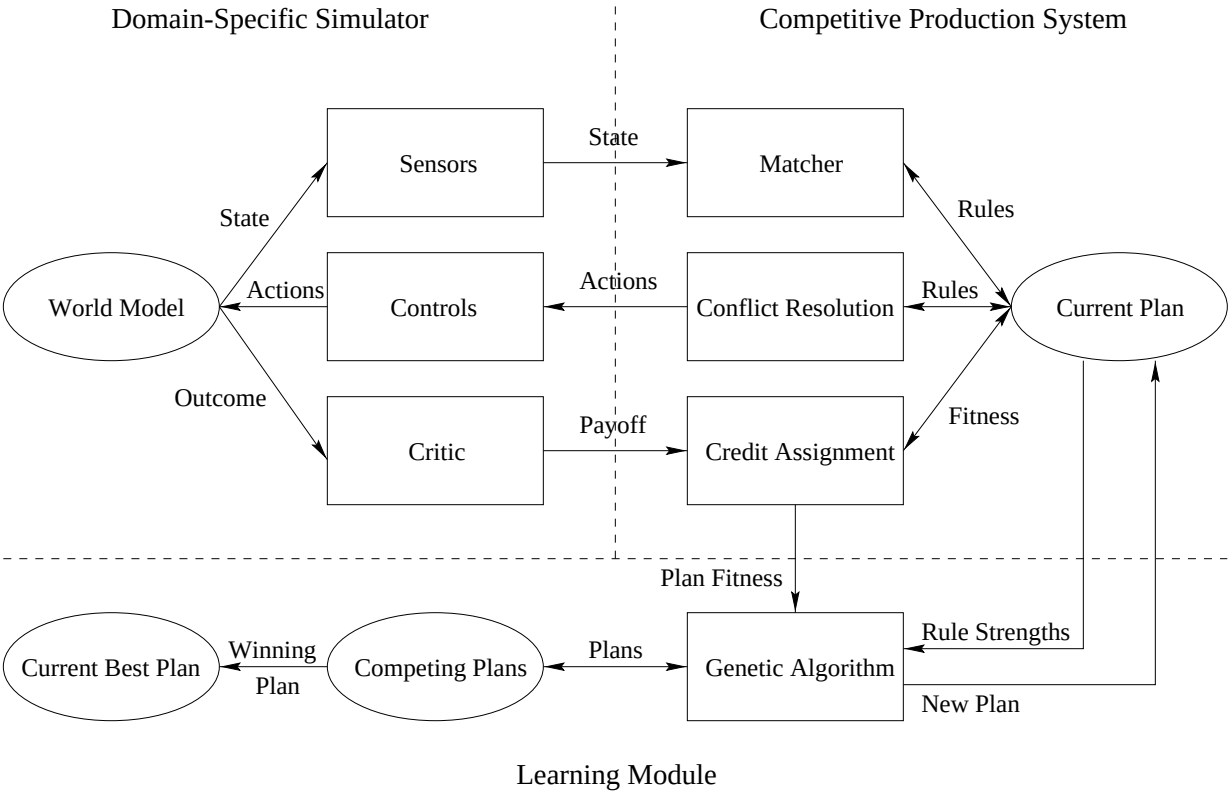


Figure 3.2: Architecture for SAMUEL.

modules provides the basis for all of the algorithms studied in this dissertation and is shown in Figure 3.2 in the context of the GA.

3.3 Co-Learning Methods and Games

In the past ten years, research in the area of reinforcement learning and control has literally exploded. New algorithms and new theoretical results are being published at a high rate. Unfortunately, multi-agent reinforcement learning has received limited attention (although that is changing), and, even then, most of the research has focused on cooperative agents. Research in reinforcement learning and game playing has largely been limited to a single agent learning to play against a fixed opponent. Such situations reduce to problems of control learning.

Recently, some research has begun to appear in which multiple competing agents (usually two) are learning simultaneously. With two notable exceptions, all of this work has been done within the last five years. In the following sections, we review this work and suggest directions for research reported in this dissertation and for further research in the field.

3.3.1 Samuel's Checkers Player

It is interesting that one of the earliest success stories in machine learning was an approach similar to temporal difference learning applied to game playing. In 1959 and 1967, Arthur Samuel reported on experiments he performed with a machine learning an evalua-

tion function for board positions in the game of checkers [297, 298]. Samuel also reported on experiments in rote learning, but we limit the discussion to his novel reinforcement learning procedure.

Early research in computer game playing focused on developing evaluation functions to be used in searching game trees. Because non-trivial games, such as chess and checkers, have game trees that cannot be fully searched, the evaluation functions provided a prediction of performance from interior nodes of the tree, given "optimal" play from that point forward. These evaluation functions were hand constructed and generally provided no better than mediocre performance.

Samuel's idea was to use experience from actual play to learn the evaluation function. Then the computer could adapt its play to improve its performance by gradually improving the ability of the evaluation function to predict performance. In Samuel's approach, two players were designated—one (designated Alpha) modified its evaluation function during play and the other (designated Beta) used a static evaluation function corresponding to the best function learned so far.

Samuel's evaluation function was a linear function of 16 terms (attributes), taken from a set of 38 terms, plus a small set of binary connective terms (permitting pairwise combinations of certain attributes). Each term had a coefficient that was a positive or negative power of two, derived from the correlation between predictive "accuracy" and performance. At a particular state in the game, the scoring polynomial (i.e., the evaluation function) is used with lookahead to select a move using minimax. In addition, the current

board position is scored. The current score is compared to the minimax backed-up score and the difference computed. The coefficients in the scoring polynomial are then modified to reduce the difference in the scores. Note that this approach assumes the backed-up score is more accurate than the current score, presumably because the backed-up scores are computed from states closer to the terminal board positions in the game. Note further that this is exactly the assumption made in Sutton's TD(0) algorithm [332].

In addition to updating coefficients on the terms, Samuel observed that the performance of the scoring polynomial depended on the current 16 terms selected from the 38 available terms. Samuel hand picked the 38 terms and (somewhat arbitrarily) decided the scoring polynomial would only use 16 of these terms. He then associated a "low term" tally with each term, indicating the number of times each term had the lowest correlation between the sign of the term and the sign of the difference between the current score and the backed-up score. When the tally exceeded a pre-set threshold, the term was removed from the polynomial and a new term was selected from the remaining 22 unused terms.

Although not traditionally considered an implementation of a multi-agent learning strategy, the self-play learning methodology, where Alpha plays Beta and Beta has the best learned polynomial so far, is suggestive of a strategy proposed by Grefenstette and Daley [152]. In their approach, competing agents "co-evolve" rules for engagement where each agent alternates learning (see Section 3.3.6). Because the two checker players are essentially the same, transferring a learned polynomial to Beta is analogous to alternating learning between Alpha and Beta. The primary difference is that Alpha and Beta do not

follow two independent paths as they learn—Beta's strategy comes from Alpha's experience.

3.3.2 Temporal Difference Methods and TD-Gammon

More recently, Gerald Tesauro applied temporal difference learning in self-play in the game of backgammon [342, 343, 344]. As with Samuel's checkers player, Tesauro's program (named TD-Gammon) has the two players playing each other using an evaluation function resulting from learning. The approaches differ in that, for TD-Gammon, both players use the *current* evaluation function that has been learned. In addition, where Samuel constructed several abstract features for the terms in his scoring polynomial, Tesauro processes raw state information.

In TD-Gammon, Tesauro constructed a feed-forward neural network with one hidden layer which he trained using TD(λ). For his experiments, he found little difference in using various values for λ (except for learning time), so he arbitrarily set $\lambda = 0.7$. The input layer of the network had 198 nodes. Each of the 24 board positions had four nodes for each player (indicating 1, 2, 3, or 4+ pieces of a particular color) plus six additional nodes were included to encode the number of pieces on the bar, the number of pieces off the board, and which player moved next. The network had only one output node providing a prediction of the probability White would win. At each stage of the game, the dice are rolled and all legal moves from the dice roll are considered by evaluating the successor states with the network. When White moves, the action maximizing the probability White wins is selected. When Black moves, the action minimizing the probability White wins is selected.

Training consisted of using the network for each player and using the sequence of predictions in the TD(λ) update equation. The actual payoff of one (for White winning) or zero (for Black winning) was used at the end of the game. Because both players used the same evaluation function during learning, we can claim that TD-Gammon provides a rudimentary approach to co-learning among homogeneous players (i.e., players with identical, competing objectives and identical capabilities). This, of course, is the simplest form of multi-agent learning.

3.3.3 Reinforcement Learning in Cognitive Game Theory

In the economics community, one of the communities responsible for considerable research in game theory [49, 123, 130, 196, 286, 358, 360], limited work is being done in learning game strategies. This work, however, is largely restricted to single-player games. One notable exception is the work by Roth and Erev [123, 286] in what they call "cognitive game theory." Specifically, they distinguish between "low" game theory and "high" game theory, where low game theory assumes players have limited rationality but adapt to experience playing the game to derive rational strategies. Further, the players may not consider all strategies available to them, and they may not be "subjective expected utility maximizers [123]." High game theory assumes the players already have "full" rationality and can deduce the rational strategy from the rules of the game.

Roth and Erev focus their research on low rationality game theory to facilitate modeling the learning process. Their goal is to better understand the nature of different

economic games and the limitations of learning rational (i.e., optimal) strategies to play these games. The learning model used by Roth and Erev is relatively simple and applies to both players in a two-person game. No requirement exists limiting the game to be zero-sum.

Specifically, if player p_i plays strategy s_{ij} and receives payoff ρ, then the propensity for p_i to play any s_{ik} is updated as

$$q_{ik} = (1 - \phi)q_{ik} + E_j(k, R(\rho))$$

where ϕ us a recency parameter (similar to the learning rate in Q-learning), R is a mapping of payoff into a non-negative reinforcement signal (in the simplest case, $R(\rho) = \rho$), and E is an update value reflecting the experience gained from play. Erev and Roth define E to be

$$E_j(k, R(\rho)) = \begin{cases} R(\rho)(1 - \epsilon), & \text{if } k = j \\ R(\rho)\frac{\epsilon}{2}, & \text{if } k = j \pm 1 \\ 0, & \text{otherwise} \end{cases}$$

where ϵ is a small, positive, user-defined parameter. Note this model assumes neighboring strategies are "similar" in some sense. When no such similarity exists, E is modified as follows:

$$E_j(k, R(\rho)) = \begin{cases} R(\rho)(1 - \epsilon), & \text{if } k = j \\ R(\rho)\frac{\epsilon}{m-1}, & \text{otherwise} \end{cases}$$

where m is the number of available strategies.

Given the "propensities," q_{ij}, the probability of selecting strategy j (thus defining a mixed strategy) is

$$p_{ij}(s) = \frac{q_{ij}}{\sum_{k=1}^{m} q_{ik}}$$

Roth and Erev studied the performance of this approach on three simple economic games

with pure strategy equilibria [286] and eleven simple economic games with mixed-strategy equilibria [123]. They found their simulated results tracked well with several experiments involving human subjects learning to play these same games. The intent of their experiments did not include developing an algorithm for learning the equilibria in the games. Rather, they attempted to model a "psychologically plausible" learning approach that could determine the ability of low rationality agents to learn game equilibria, given a variety of playing conditions. They concluded that their model provided an extremely good approximation of the biological learning process in a wide variety of contexts and believe such a simple model motivates additional development of low rationality, cognitive game theory.

3.3.4 Q-Learning and Markov Games

In Sections 3.3.1 and 3.3.2, game learning was not set up within a multi-agent co-learning framework; however, we saw that the results reported could be interpreted in the context of co-learning among homogeneous competing players. In Section 3.3.3, we discussed several experiments in game learning among heterogeneous competing players, but the intent was in modeling the cognitive process of game learning rather than developing an algorithm to actually learn the equilibrium of the games studied. Michael Littman explored the possibility of using Q-learning for co-learning among homogeneous players in the context of Markov games [211, 213]. It appears his approach also applies to heterogeneous players, but no such experiments were reported. Recall from Section 2.3 that a Markov game is a special form of Markov decision process in which actions by two (or more) competing

players jointly determine the next state of the game. Solving the Markov game consists of developing a policy for play that maximizes the expected payoff to each player under the assumption the other players are playing optimally.

Littman proposes the following approach to applying Q-learning to solve two-person Markov games. Assuming a lookup table exists mapping current state-action-action triples to Q-values, play consists of selecting actions either at random (to promote exploration) or according to the current policy. This policy is given by returning an action according to mixed strategies derived for one player which is then fixed to permit selecting the other player's action through simple minimization [213]. The mixed strategy for the first player is determined by solving the linear program described in Section 2.3. Obviously, this approach is not guaranteed to be optimal because both the linear program and its dual must be solved to determine appropriate strategies satisfying the constraints of the linear program.

Learning the Q-values in the lookup table is analogous to standard Q-learning. Specifically,

$$Q_{t+1}(s, a_1, a_2) = (1 - \alpha)Q_t(s, a_1, a_2) + \alpha[c(s, a_1, a_2) + \gamma V(s')]$$

where

$$V(s') = \max_{\pi \in \Pi(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{\forall a_1 \in \mathcal{A}_1} \pi(a_1) Q(s', a_1, a_2)$$

and $\Pi(\mathcal{A}_1)$ is the set of probability distributions over $\mathcal{A}_1$. Finding $V(s')$ requires solving the linear program associated with the next state. In actual play, this can be done by simply

storing the value of the linear program solved in that state and then substituting during learning.

Littman applied this strategy to a simple variation on the game of soccer. In particular, he constructed a 4×5 playing grid where each player, designated A and B, occupied distinct squares on the grid. At each step in the game, a player could choose to move north, move south, move east, move west, or stand still. Players do not move simultaneously. Rather, the player to move first is selected at random at each time step. One of the players always has possession of the ball. Should players "collide," possession of the ball transfers to the stationary player, and the moving player returns to the previous position. At each end of the playing field is a goal square. When a player possessing the ball reaches the appropriate goal, that player receives one point. Each game lasts 100,000 steps.

The above algorithm was compared to simple Q-learning. Further, each learning player was matched against a random player, another player using the same learning strategy, and a hand-built player. All learning cases performed extremely well (90% won) against the random players. When training against another learning player, each approach performed better against the hand-built player than following learning against the random player. Further, in a surprising result, Q-learning performed better against the hand-built player than minimax-Q-learning did against the hand-built player following training with other learning players.

3.3.5 Advantage Updating and Differential Games

In independent research, Harmon, Baird, and Klopf investigated applying reinforcement learning in function approximators (namely, artificial neural networks) to learning solutions to differential games [29, 30, 31, 161, 162, 163]. For their research, they focused on a single linear-quadratic differential game of pursuit in which a single missile (designated P) pursues a single airplane (designated E), which is similar to the problem studied by Grefenstette *et al.* [149, 153, 163, 273]. As a linear-quadratic game, the kinematic equations are linear functions of the current state and action, and the payoff function is a quadratic function of acceleration and the distance between the players. Specifically, the kinematic equations are

$$x_P(t) = x_P(t-1) + \Delta t \dot{x}_P(t-1)$$

$$y_P(t) = y_P(t-1) + \Delta t \dot{y}_P(t-1)$$

$$x_E(t) = x_E(t-1) + \Delta t \dot{x}_E(t-1)$$

$$y_E(t) = y_E(t-1) + \Delta t \dot{y}_E(t-1)$$

where

$$\dot{x}_P(t) = \dot{x}_P(t-1) + \Delta t \ddot{x}_P(t-1)$$

$$\dot{y}_P(t) = \dot{y}_P(t-1) + \Delta t \ddot{y}_P(t-1)$$

$$\dot{x}_E(t) = \dot{x}_E(t-1) + \Delta t \ddot{x}_E(t-1)$$

$$\dot{y}_E(t) = \dot{y}_E(t-1) + \Delta t \ddot{y}_E(t-1)$$

Actions consist of modifying the acceleration in the x and y directions, thus modifying $\ddot{x}_P$, $\ddot{y}_P$, $\ddot{x}_E$, and $\ddot{y}_E$ respectively. Payoff is defined as

$$\rho(s,a) = [d(P,E)^2 + \text{accel}(P)^2 - 2\ \text{accel}(E)^2]\Delta t$$

where $d(P,E)$ is the distance between between P and E, accel$(P)^2$ is the dot product of P's acceleration vector with itself, and accel$(E)^2$ is the dot product of E's acceleration vector with itself.

To play this game, two values are computed using function approximators—$V(s,a)$, which is an estimate of the expected total discounted reward from state s, and $A(s,a)$, which is the advantage of applying action a in state s over applying the estimated optimal action is state s. In a given state, each player chooses an action which maximizes their expected payoff given their opponent plays optimally.

In game playing, Harmon *et al.* recognize that optimal play may require application of mixed strategies. To simplify their experiments, however, they chose to assume pure-strategy equilibria existed for their game. As a result, they selected actions in each state as follows. First, the pursuer selects an action that maximizes its payoff by selecting the action maximizing *advantage* over all pairs of actions. Then the evader selects the action that minimizes advantage to P, given the action P chose.

To learn $V(s,a)$ and $A(s,a)$, Harmon *et al.* take the following steps. First, they compute the "minimax" for the current state using the strategy described above. Next they calculate

$$V(\mathbf{s}) = \mathbf{s}_P^T \mathbf{D}_P \mathbf{s}_P + \mathbf{s}_E^T \mathbf{D}_E \mathbf{s}_E$$

where $\mathbf{s}_P$ is the vector indicating P's current position, $\mathbf{s}_E$ is the vector indicating E's current position, and $\mathbf{D}_P$ and $\mathbf{D}_E$ are weight matrices that are updated during learning. Then the advantages are calculated using

$$\begin{aligned} A(\mathbf{s},\mathbf{a}) = \quad & \mathbf{s}_P^T\mathbf{A}_P\mathbf{s}_P + \mathbf{s}_P^T\mathbf{B}_P\mathbf{C}_P\mathbf{a}_P + \mathbf{a}_P^T\mathbf{C}\mathbf{s}_P + \\ & \mathbf{s}_P^T\mathbf{A}_P\mathbf{s}_P + \mathbf{s}_P^T\mathbf{B}_P\mathbf{C}_P\mathbf{a}_P + \mathbf{a}_P^T\mathbf{C}\mathbf{s}_P \end{aligned}$$

where $\mathbf{a}_P$ and $\mathbf{a}_E$ are acceleration vectors for P and E respectively, and $\mathbf{A}_P$, $\mathbf{A}_E$, $\mathbf{B}_P$, $\mathbf{B}_E$, $\mathbf{C}_P$, and $\mathbf{C}_E$ are weight matrices updated during learning.

Because payoff is received throughout play, the payoff is calculated next. Then the actions are applied, and the state is updated. This leads to calculating $V(s,a)$ in the next time step. Finally, all of the weights are updated by gradient descent using the equation (letting mm() represent finding the minimax),

$$\begin{aligned} \Delta w = \quad & -\alpha\left\{\left[\rho(s_t,a_P,a_E)+\gamma^{\Delta t}V(s_{t+\Delta t})-V(s_t)\right]\frac{1}{\Delta t} - A(s_t,a_P,a_E) + \mathrm{mm}(A(s_t))\right\} \\ & \times\left\{\left[\gamma^{\Delta t}\frac{\partial V(s_{t+\Delta t})}{\partial w}-\frac{\partial V(s_t)}{\partial w}\right]\frac{1}{\Delta t} - \frac{\partial A(s_t,a_P,a_E)}{\partial w} + \frac{\partial \mathrm{mm}(A(s_t))}{\partial w}\right\} \\ & -\alpha\mathrm{mm}(A(s_t))\frac{\partial \mathrm{mm}(A(s_t))}{\partial w} \end{aligned}$$

which includes a procedure similar to Q-learning but based on the Bellman residual given, in general, by

$$\begin{aligned} \mathrm{res}(s_t,a_P,a_E) = \quad & \left[\rho(s_t,a_P,a_E)+\gamma^{\Delta t}V(s_{t+\Delta t})-V(s_t)\right]\frac{1}{\Delta t} \\ & -\left[A(s_t,a_P,a_E)-\mathrm{mm}(A(s_t))\right] \end{aligned}$$

Their experiments compared their residual advantage updating procedure to the optimal

solution to the game (determined by numerically solving for the optimal weight matrices). Performance of the algorithm was quite good, perhaps suggesting the game did have pure strategy equilibria. It is unclear how well this procedure would perform, given their simplifications, had mixed strategies been required.

3.3.6 Coevolution Methods

Recently, work in co-evolutionary algorithms has begun to suggest approaches to multi-agent co-learning with some encouraging initial results [151, 152, 266, 323]. Extending his work on SAMUEL, Grefenstette defines a uniform sensor architecture for multi-agent environments [151]. He claims that modeling information about all agents in the environment would be too complex, requiring significant computational resources. His approach models the learning agent in three stages—sensors, conflict resolution, and actions. The sensors sense each of the external agents (i.e., the agents other than the learning agent) and represent their states as "tracks." Each track is simply the set of sensor readings for that agent. The rules in the current plan are then matched against each track and bid to fire. For each track, the recommended rule to fire is the rule with the highest bid. Once a rule is identified for each track, the bids are combined over all of the tracks by multiplying the strengths for like actions. The action taken corresponds to the actions with the highest bids.

For example, suppose the following actions are recommended for two separate tracks:

$$(\text{right}(0.9), \text{left}(0.4))$$
$$(\text{right}(0.8), \text{left}(0.9))$$

In this example, the action would be determined by multiplying the bids for "left" and "right". Thus the combined bid would be

$$(\text{right}(0.72), \text{left}(0.36))$$

and the action "right" would be selected. In addition, SAMUEL provides the option of normalizing the combined bids and then selecting the action according to the resulting probability distribution.

Grefenstette tests his multi-agent architecture on a pursuit game in which two pursuers chase a single evader, and the evader is learning to evade. It is interesting to note that the average performance achieved in this architecture was approximately 90% evasion of both pursuers with the best performance being about 95%. This seems to track well with our experiments (reported in Section 4.3.3).

Further extending SAMUEL, Potter, DeJong, and Grefenstette developed an approach to coevolution in which an agent is decomposed into "subagents" each responsible for learning an activity to be combined to solve a complex task [266]. They called this extension *Cooperative Coevolutionary Genetic Algorithms*. In this approach, multiple agents operate on a single task in parallel. The agents are initialized with rules to bias their activity toward some subset of the total problem. For example, an agent seeking food in a hostile environment behaves differently when food is present and when it is absent. In the presence of food, it may move directly toward the food in an attempt to beat its competitors from reaching the food first. When food is absent, it may attempt to position itself such that it avoids other agents but is still in a strong position to capture food when it appears. The

multi-agent (or composite) plan then consists of the concatenation of the best plans learned by the subagents.

Finally, Grefenstette and Daley consider a coevolutionary strategy for cooperative and competitive multi-agent plans [152]. These are the first experiments by Grefenstette *et al.* in which multiple competitive agents learn simultaneously. In these experiments, Grefenstette and Daley consider four multi-agent scenarios:

- A single genetic algorithm learns a plan that is used by both agents. This assumes the agents are homogeneous and have common goals.
- Separate genetic algorithms for each agent are run simultaneously, and each agent is tested against a single random opponent.
- Separate genetic algorithms for each agent are run simultaneously, and each agent is tested against the best opponent from the previous generation.
- Separate genetic algorithms for each agent are run simultaneously, and each agent is tested against a "best opponent" selected randomly from all previous generations

They tested the coevolutionary genetic algorithms on a food-gathering task in which the two agents were competing against each other to obtain the most food. Their results indicated little difference in the various approaches with the last approach performing the strongest. Even then, it appeared none of the differences were statistically significant. Further, because they did not compare the results of coevolution to any static strategy, it was difficult to assess whether any improvement occurred at all.

Smith and Gray describe an alternative approach to coevolution, which they call a co-adaptive genetic algorithm, applied to the game of Othello [323]. Their approach focuses on developing a fitness function that is derived from the ability of a member of the population to compete against other members of the population. Thus, their co-adaptive fitness function appears to be a variation on tournament selection, except that selection takes place at the end of a complete round-robin tournament.

3.4 Related Work in Learning and Game Playing

In this last section, we briefly review some related work in machine learning and games. The literature associated with machine learning and games is quite extensive, in spite of the relatively small amount of work done in co-learning and games. Rather than providing a comprehensive review of the field, this section highlights some recent work that is interesting and may provide further insight into co-learning and games.

3.4.1 Opponent Modeling

In current research in learning and games, the focus of the research is on a player learning a strong strategy against its opponent. Most of the research assumes a fixed opponent, but some assumes all players are attempting to learn what amounts to an "optimal" strategy, applicable to all players. Recent work by Carmel and Markovitch suggests that a more realistic approach is for a player to adapt to the current opponent. This led them to develop their M^* algorithm in which a player learns its opponent's strategy and adapts accordingly

[68, 67, 69]. Others have provided similar strategies to modeling opponents, such as the agent tracking methods of Tambe [336, 337, 338, 339, 340], and the equilibrium search methods of Goldman and Rosenschein [144, 243] and of Koller [194, 195, 196, 197, 359].

For their approach, Carmel and Markovitch assume a two-player game where each player has an evaluation function to be used in a minimax search of the game tree. Typically, the same heuristic evaluation function is used in minimax search (or at least this is the assumption by each player). In the M^* algorithm, each player has a different heuristic evaluation function. A player models its opponent by modeling the opponent's heuristic evaluation function and applying that function during minimax.

Learning takes place in two areas. First, the evaluation depth of the opponent's heuristic is learned (assuming no strategies such as selective deepening). The algorithm assumes a particular evaluation function and then examines past choices by the opponent. Given the evaluation function and the past choices, plausible choices for depth can be deduced. As expected, as error in the evaluation function increases, so does error in the depth.

Second, the evaluation function itself is learned. The algorithm assumes a linear combination of features, and learning consists of determining these weights. The features are known, and it is assumed the heuristic does not change during play (thus limiting the problem to single-agent learning). Several of the approaches discussed earlier can be used to learn these weights. Carmel and Markovitch applied a basic gradient-descent search strategy for several depths to learn, and combine the results with the learned depth.

3.4.2 Learning Chess-Like Games

Barney Pell developed an approach to deriving strategies from declarative specifications of games is a system he calls METAGAMER [258, 259, 260, 261]. METAGAMER processes the rules and constraints of "symmetric chess-like" games and derives the rules and evaluation functions for these games. Pell's approach has been applied to several symmetric chess-like games including chess, checkers, Tic-Tac-Toe, Othello, and Go.

In his work on game learning, Pell assumes a fixed opponent.[3] His approach is to apply 1-ply search and delayed reinforcement. A set of features are provided to the learner, and METAGAMER is used to determine the legal actions from a declarative description of the game. At each stage of the game, the player evaluates all legal moves and computes values w and l for each of these actions. At the end of the game, moves associated with the winner have their w scores incremented, and moves associated with the loser have their l scores incremented [258, 259].

The learning problem is one of determining a function that takes the set of legal moves, the w and l statistics, and the features of the current state and returns an evaluation of those moves. Where other researchers focus on learning the evaluation functions, Pell provided a hand-crafted evaluation function and simply collected the w and l statistics for learning. When applied to the game of Go, the results of Pell's approach were encouraging for small games (e.g., 3×3 boards) in that his program always won. In an experiment

[3] Because his class of games is symmetric, and each of the players has common goals, it should be straightforward to extend his work similar to Samuel's and Tesauro's work in which learning occurred during self-play.

with a 9×9 board, the learning player was ability to match but not surpass a hand-crafted player.

3.4.3 Competitive Learning

In an interesting alternative view to game learning, Rosin and Belew provide a general framework for metalearning in games [285]. Their approach assumes a player is learning to beat a fixed *set* of opponents and that each player is applying a different learning algorithm. In the simplest case, there are only two players, each with its own strategy-learning algorithm. The players use their strategy-learning algorithms to learn to play the game, and "competitive learning" pits the results of these algorithms against each other to select the "best" strategies to play.

In Rosin and Belew's formulation, strategy learning does not take place simultaneously. Rather, each strategy is pitted against a fixed set of opponents and learns to play against those opponents. The competitive learner then holds a tournament between the results of these strategy learners to determine the current best strategy. This current best strategy may then become the strategy for the fixed players used in learning.

Rosin and Belew point out that Samuel's original work in learning to play checkers was a form of competitive learning. In particular, the Beta player always played the current best strategy. The Alpha player applied the strategy learning to derived evaluation functions for playing checkers. When Alpha's strategy succeeded in beating Beta, Beta was given Alpha's strategy.

The work in competitive learning does not focus on developing competitive algorithms. Instead, it provides a formal framework for analyzing learnability of game strategies in a competitive learning environment. Rosin and Belew provide several theoretical results characterizing learning under such conditions, including proofs that under the condition of a single counter-example concept learner (corresponding to one player), perfect strategies can be learned, but not in polynomial time. They then suggest that if both players provide a counter-example learning strategy that covers previous opponents, perfect strategies can be learned in time polynomial in the number of strategies considered.

3.4.4 Artificial Life and Repeated Games

Artificial life is concerned with developing or evolving populations of agents to simulate behaviors of organisms at the population level. Research in this area focuses on population and evolution dynamics; therefore, we should expect them to be concerned with competition among multiple populations. This work, however, does not consider competition on an individual basis. Robert Collins [83] explored issues of sexual selection and female choice in a predatory environment, co-evolution of hosts and parasites, and foraging behavior among artificial ants. Other investigations of host/parasite co-existence and co-evolution have taken a game-theoretic view, but work by Bremermann and Pickering focuses on natural evolution [62]. In another biological study, Hamilton considered the effects of competition between hosts and parasites in the evolutionary process [160]. To promote distribution of skills among populations, Davidor [102] studied the effects of niching and speciation in

genetic algorithms, and Werner and Dyer focused on developing communication mechanisms between organisms through artificial evolution [365].

Some work has been done in evolutionary computation, artificial life, and iterated games. The most common game considered in this context is the iterated prisoner's dilemma [28, 223, 234, 299, 325]. The Prisoner's Dilemma is a two-player non-zero-sum game in which players must decide to cooperate or defect based on their expected payoffs. The "best" strategy to play in this game is for both players to cooperate; however, under the assumption of a Nash equilibrium point, such an equilibrium point is found to exist only when both players defect. The iterated prisoner's dilemma is a game in which players repeatedly play the prisoner's dilemma and choose their strategies based on previous plays of the game. Axelrod held a tournament among players of various strategies and found that the "Tit-for-Tat" strategy (where each player plays the other player's previous strategy) was the best [28].

The work by Stanley *et al.* focused on several agents with different strategies evolving chocie and refusal mechanisms to determine when a game was played [325]. The goal was not on learning strategies but on characterising choice and refusal in a competitive environment. Sandholm and Crites applied Q-learning in the iterated prisoner's dilemma in which multiple learners faced each other and a fixed player using "Tit-for-Tat" [299]. They explored methods involving lookup tables and recurrent networks. They found all learners fared well against the fixed player but had difficulty when playing against other learners (presumably because of the non-stationarity of the problem). Nevertheless, they found that

the agents using lookup tables with long histories performed the best.

Results relevant to problems of multi-agent learning and learning with competition can be found outside of the machine learning community. For example, in starting to address issues of multi-agent interaction and planning, distributed artificial intelligence has focused on developing intelligent agents under the condition that no single agent can perform a particular task. The agents must then be constructed (or must learn) such that they cooperate with one another to perform the task [122, 139]. Work by Genesereth and Rosenschein considered cooperation among agents in which there are competing objectives and where there is no communication between the agents [138, 283]. Although not focusing on competitive situations, Conry *et al.* considered planning in distributed systems when the task requires multiple, intermediate milestones to be reached, and resources are traded between the agents [87].

3.5 Summary

Research in multi-agent learning is just recently receiving extensive attention in the machine learning community. Much of this research focuses on problems of collaboration and cooperation; however, competitive problems (such as games) provide interesting challenges that must be faced in the real world, even when cooperation is the ultimate goal.

In this chapter, we provided a summary of relevant research in single agent and multi-agent reinforcement learning and game playing. The field of machine learning and game playing is vast, and providing a comprehensive review of the field here is not practical.

Therefore, we focused on work that was either directly related to multi-agent learning or suggestive of ways to extend single agent learning into the multi-agent area.

Chapter 4

Sequential Decision Making and Pursuit Games

4.1 Learning Approaches in Game Playing

Reinforcement learning (RL) is challenging in part because of the delay between taking an action and receiving a reward or penalty. Typically an agent takes a long series of actions before the reward, so it is hard to decide which of the actions were responsible for the eventual payoff. Both lazy and eager approaches to reinforcement learning can be found in the literature. The most common eager approach is the use of temporal-difference learning on neural networks [36, 37, 81, 342]. The advantages to a lazy approach are three-fold. First, minimal computational time is required during training, because training consists primarily of storing examples (in the most traditional lazy approach, k-nearest neighbor). Second, lazy methods have been shown to be good function-approximators in continuous state and action spaces [25]. This capability is important for our task of learning to play

differential games. Third, traditional eager approaches to reinforcement learning assume the tasks are Markov decision problems. When the tasks are non-Markovian (e.g., when history is significant), information must be appended to the state to encapsulate some of the prior state information, in order to approximate a Markov decision problem. Because the lazy approach stores complete sequences, non-Markovian problems can be treated in a similar fashion to Markovian problems.

The class of RL problems studied here has also been studied in the field of *differential game theory.* In differential games, differential equations model how actions taken by the players in the game change the state of the game over time. The object of analyzing a differential game is to determine the optimal strategies for each player of the game and to determine the value of the game (i.e., the expected payoff to each player) assuming all of players follow the optimal strategies.

A pursuit game is a special type of differential game that has two players, called the pursuer (P) and the evader (E). The evader attempts to achieve an objective, frequently to escape from a fixed playing arena, while the pursuer attempts to prevent the evader from achieving that objective. Examples include such simple games as the children's game called "tag," the popular video game PacMan, and much more complicated predator-prey interactions in nature. These examples illustrate a common feature of pursuit games—the pursuer and the evader have different abilities: different speeds, different defense mechanisms, and different sensing abilities.

In this chapter, we apply two lazy learning algorithms and one eager learning algo-

rithm to two differential games of pursuit, and we attempt to characterize the performance of these algorithms on the games. Although these limited experiments do not afford us the ability to draw general conclusions about lazy versus eager learning in game playing, we use the results to motivate the research discussed in subsequent chapters on combining learning approaches and developing a lazy and eager approach to co-learning. The specific games used for the experiments in this chapter are two variations of the evasive maneuvers game [153].

4.2 The Evasive Maneuvers Game

The evasive maneuvers task as a differential game is a variation on the Homicidal Chauffeur game. Even though the solution to the Homicidal Chauffeur game is intuitive, the actual surface characterizing the solution is highly nonlinear. Thus we should reasonably expect the surface for extensions to the problem (such as those discussed in this chapter) to be more difficult to characterize. Grefenstette *et al.* [153] studied the evasive maneuvers task to demonstrate the ability of genetic algorithms to solve complex sequential decision making problems. In their two-dimensional simulation, a single aircraft attempts to evade a single missile.

We initially implemented the same pursuit game as Grefenstette *et al.*, and later we extended it to make it substantially more difficult. In this game, play occurs in a relative coordinate system centered on the evader, E. Because of the relative frame of reference, the search space is reduced and games are determined by their starting positions. P uses a

fixed control law to attempt to capture E, while E must learn to evade P. Even the basic game is more difficult than the Homicidal Chauffeur game, because the pursuer has variable speed and the evader has a non-zero radius of curvature. Our extended version includes a second pursuer, which makes the problem much harder. Unlike the single-pursuer problems, the two-pursuer problem has no known optimal strategy [175], and for some initial states, there is no possibility of escape. Second, we gave the evader additional capabilities: in the one-pursuer game, E only controls its turn angle at each time step. Thus E basically zigzags back and forth or makes a series of sharp turns into the path of P to escape. In the two-pursuer game, we gave E the ability to change its speed, and we also gave E a bag of "smoke bombs," which will for a limited time help to hide E from the pursuers.

In our definition of the two-pursuer task, both pursuers ($P1$ and $P2$) have identical maneuvering and sensing abilities. Further, they use the same control strategy: they anticipate the future location of E and aim for a location where they can capture in the fewest time steps. They begin the game at random locations selected according to a uniform probability distribution on a fixed-radius circle centered on the evader, E. The initial speeds of $P1$ and $P2$ are much greater than the speed of E, but they lose speed as they maneuver, in direct proportion to the sharpness of the turns they make. The maximum speed reduction is 70%, scaled linearly from no turn (with no reduction in speed) to the maximum turn angle allowed of 135°. They can regain speed by traveling straight ahead, but they have limited fuel. If the speed of both $P1$ and $P2$ drops below a minimum threshold, then E escapes and wins the game. E also wins by successfully evading the pursuers for 20 times

steps (i.e., both $P1$ and $P2$ run out of fuel). If the paths of either $P1$ or $P2$ ever pass within a threshold range of E's path during the game, then E loses (i.e., the pursuer will "grab" E). We use the term "game" to include a complete simulation run, beginning with the initial placements of all of the players, and ending when E either wins or loses, at most 20 time steps later.

Against one pursuer, E controls only its turn angle, which is sufficient to play the game well. With two pursuers $P1$ and $P2$ in the game, E has additional information about its opponents. This information includes 13 features describing the state of the game, including E's own speed, the angle of its previous turn, a game clock, the angle defined by $P1$–E–$P2$, and the range difference between $P1$ and $P2$. It also has eight features that measure $P1$ and $P2$ individually: speed, bearing, heading, and distance. Bearing measures the position of the pursuer relative to the direction that E is facing (e.g., if E is facing north and $P1$ is due east, then the bearing would be 3 o'clock). Heading is the angle between E's direction and the pursuer's direction. When fleeing two pursuers, E can adjust its speed and turn angle at each time step, and it can also periodically release a smoke bomb, which introduces noise into the sensor readings of $P1$ and $P2$. If smoke is released, the turn angle of the pursuer is shifted by a random factor up to 50% of the current turn angle. As the severity of the turn increases, so does the potential effect from smoke.

In our task, the missiles are launched simultaneously from locations chosen at random according to a uniform probability distribution. The missiles are initially 1,500 units away from the aircraft. The missiles may come from different locations, but their

initial speed is the same and is much greater than that of the aircraft. As the missiles maneuver, they lose speed. If they drop below a minimum threshold, they are assumed to be destroyed. The aircraft successfully evades the missiles by evading for 20 time steps or until both missiles drop below a minimum speed threshold. To make the problem consistent with the Homicidal Chauffeur, we also assume that if the paths of the missiles and the aircraft ever pass within some "lethal envelope," then the aircraft is destroyed; i.e., the missiles need not collide with the aircraft. The missile positions are considered throughout their flight path, not just at the end of a time step (Figure 4.1). In particular, for flight paths for both the airplane and each missile, we compute the distance of the closest point to each other during a time step. If that distance is less than the range of the lethal envelope (which was set to 100), then the aircraft is destroyed. We use the term "engagement" to include a complete simulation run, beginning with the launch of the two missiles and ending either after destruction of the aircraft or successful evasion of the missiles. We demonstrate one engagement with two missiles in which the airplane is destroyed in Figure 4.2.

When flying against one missile, the capabilities of the aircraft are identical to the aircraft used by Grefenstette *et al.* [153]. As noted earlier, in the two missile task, the aircraft has 13 sensors. The nine state variables measured by these sensors are:

- *speed*: Indicates the previous speed of the aircraft. The legal speeds for the aircraft lie in the range [250,400].
- *previous turn*: Indicates the previous turn taken by the aircraft. The legal turns for the aircraft lie in the range $[-135°,135°]$.

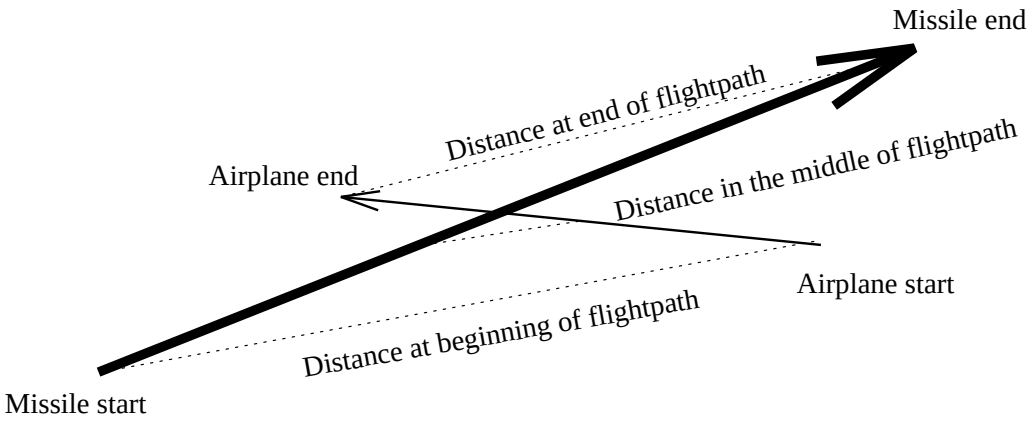


Figure 4.1: Evaluating lethal envelope during flight.

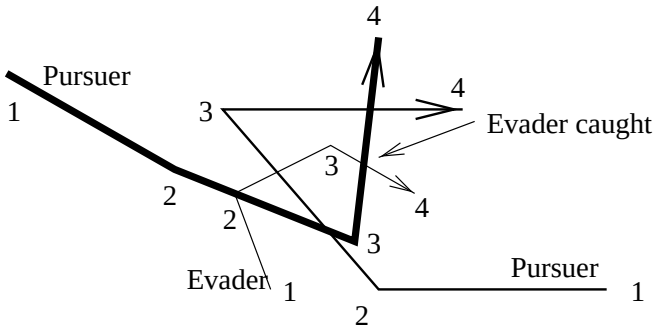


Figure 4.2: An engagement where the aircraft is destroyed.

- *clock*: Indicates the current time step of an engagement. An engagement lasts no more than 20 steps.

- *missile speed*: Indicates the current speed of each missile. The missile speeds lie in the range [0,1000]; however, a missile falls out of the sky when its speed drops below 250.

- *missile bearing*: Indicates in clock coordinates the position of each missile relative to the aircraft where 12 o'clock is directly ahead of the aircraft and 6 o'clock is directly behind.

- *missile heading*: Indicates the direction each missile is flying relative to the aircraft. At each time step, the heading of the aircraft is normalized such that its "compass" heading is reset to 0°. If a missile's heading is 0°, then it is flying in the same direction as the aircraft. If its heading is 180°, then it is travelling in the opposite direction as the aircraft.

- *missile range*: Indicates the distance from the aircraft to each missile in the engagement. Initially, the aircraft has a range of 1500 to each missile.

- *bearing difference*: Indicates the difference in bearing between the two missiles. This is a derived value based on the two missile bearing sensor readings.

- *range difference*: Indicates the difference in range between the two missiles. This is a derived value based on the two missile range sensor readings.

The goal of the learning algorithm is to build a strategy that uses these state variables to decide the appropriate values for the control variables on the aircraft. When flying against two missiles, the aircraft controls three variables:

- *speed*: Determines the speed of the aircraft on the next time step. As with the sensor reading, the speed lies in the range [250,400].
- *turn angle*: Determines the maneuver made by the aircraft at the end of the the current time step. We assume all turns are instantaneous. As with the "previous turn" sensor reading, the turn lies in the range [$-135°$,$135°$].
- *countermeasures*: Determines whether a countermeasure device will be released. Countermeasures are devices (e.g., smoke bombs, chaff, flares, or electromagnetic interference) intended to confuse the sensors of the pursuer with the hopes of causing the pursuer to lock onto a false target or lose track of the target altogether. In these experiments, countermeasures introduce noise into the missile's belief of where the aircraft is (called the *pursuit angle*). Noise is introduced by making a random shift in the pursuit angle of up to 50%.

When flying against one missile, the aircraft is able to control only the turn angle.

4.3 The Learning Algorithms

The following sections discuss the details of the experiments with the three learning algorithms and motivate the need for a learning strategy combining eager learning (as a teacher)

and lazy learning (as a performer). We explored several algorithms to determine the applicability of memory-based learning to control problems in general, and pursuit games in particular. We began by examining the ability of Q-learning to learn to play the evasive maneuvers game. We had to adapt Q-learning because of the large, continuous state space, which resulted in a memory-based variant of standard Q-learning. We then tried a traditional memory-based learning approach, nearest neighbor. Finally, we experimented with an eager learning method, genetic algorithms, to compare with the two memory-based methods.

4.3.1 Q-Learning for Evasive Maneuvers

Q-learning solves delayed reinforcement learning problems by using a temporal difference (TD) learning rule [363]. TD methods usually assume that both the feature space and the variables being predicted are discrete [332, 342]. Q-learning typically represents a problem using a lookup table that contains all states, which naturally causes problems with large, continuous state spaces such as those encountered in differential games. We therefore had to develop a method for predicting the rewards for some state-action pairs without explicitly generating them. The resulting algorithm was a memory-based version of Q-learning.

Rather than constructing a complete lookup table, our implementation of Q-learning stores examples similar to the set of instances produced for a method such as k-NN (Figure 4.3). It begins by generating a set of actions at random for a particular game; these actions do not have to result in successful evasion. Instead, the algorithm applies a payoff

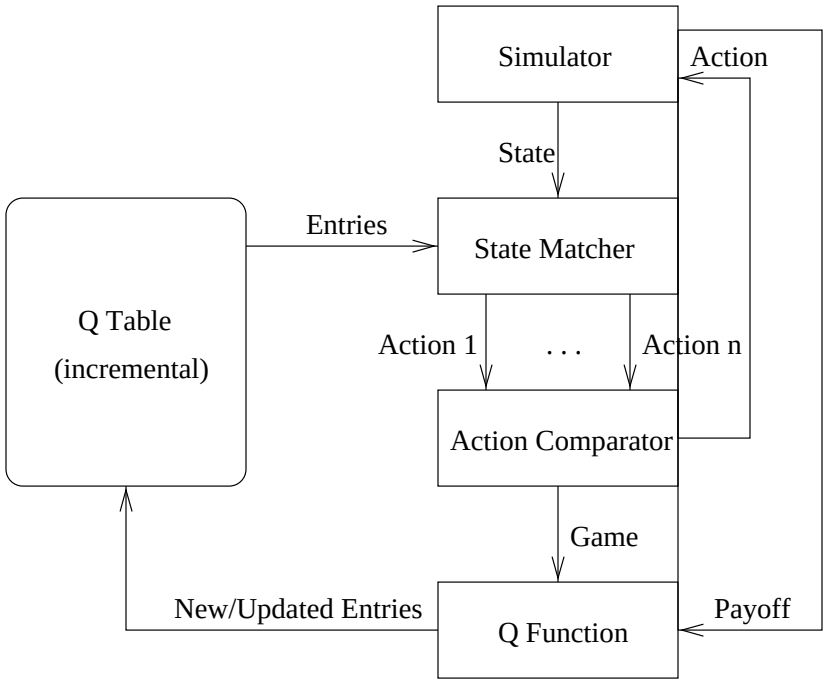


Figure 4.3: Architecture for Q learning.

function (defined below) to determine the reward for that sequence of state-action pairs. Initially, it stores the actual payoff values with these pairs. After generating the first set of pairs, learning proceeds as follows.

First, assuming that neighboring states will require similar actions, we specify two distance parameters: one for the states and one for the actions ($d_1 = 0.01$ and $d_2 = 0.005$ respectively), noting that all distances are normalized. The purpose of these parameters is to guide a search through the instance database. The system begins an evasive maneuvering game by initializing the simulator. The simulator passes the first state to the state matcher which locates all of the states in the database that are within d_1 of the current state. If the

state matcher has failed to find any nearby states, the action comparator selects an action at random. Otherwise, the action comparator examines the expected rewards associated with each of these states and selects the action with the highest expected reward. The resulting action is passed to the simulator, and the game continues until termination. It also has a probability (0.3) of generating a random action regardless of what it finds in the table. This permits it to fill in more of the database; i.e., it is exploring the state space as it is learning. It passes the resulting action to the simulator, and the game continues until termination, at which point the simulator determines the payoff. The Q function then updates the database using the complete game.

At the end of a game, the system examines all of the state-action pairs in the game. It stores in the database any state-action pair that is new, along with the reward from the game. If the pair already exists, the predicted reward is updated as follows:

$$Q(s,a) = (1-\eta)Q(s,a) + \eta[\rho + \gamma Q(s', \pi(s'))]$$

where $Q(s,a)$ is the predicted reward for state x with corresponding action a, η is a learning rate, ρ is the actual reward, γ is a discount factor, and $Q(s', \pi(s'))$ is the maximum Q value for all actions associated with state s'. State s' is the state that follows when action a is applied to state s. Reward is determined using the payoff function in [153], namely

$$\rho = \begin{cases} 1000; & \text{if } E \text{ evades the pursuers} \\ 10t; & \text{if } E \text{ is captured at time } t. \end{cases}$$

Each of the pairs in the game are then compared with all of the pairs in the database. If the distance between a stored state and action are less than d_1 and d_2 respectively for some

state-action pair in the game, then the stored state-action pair's Q value is updated.

4.3.2 1-NN for Evasive Maneuvers

Memory-based learning is a classical approach to machine learning and pattern recognition, most commonly in the form of the 1-nearest neighbor algorithm [3, 10, 90, 119, 292, 293, 366]. 1-NN is rarely used for Markov decision problems, so we had to represent the pursuit game in a format amenable to this algorithm. Further, to be successful, a memory-based approach must have a database full of correctly labeled examples, because 1-NN expects each example to be labeled with its class name. The difficulty here, then, is how to determine the *correct* action to store with each state.

We formulate Markov decision problems as classification problems by letting the state variables correspond to features of the examples, and the actions correspond to classes. Typically, classification tasks assume a small set of discrete classes to be assigned. We do not require quantization of the state space or the action space, but instead use interpolation so that any action can be produced by the 1-NN classifier.

In order to know the *correct* action to store with each state, we must at least wait until we have determined the outcome of a game before deciding how to label each step. (One example can be added at each time step). However, even after a successful game where E evades P, we cannot be sure that the actions at *every* time step were the correct ones; in general, they were not. The architecture for our nearest-neighbor system is shown in Figure 4.4.

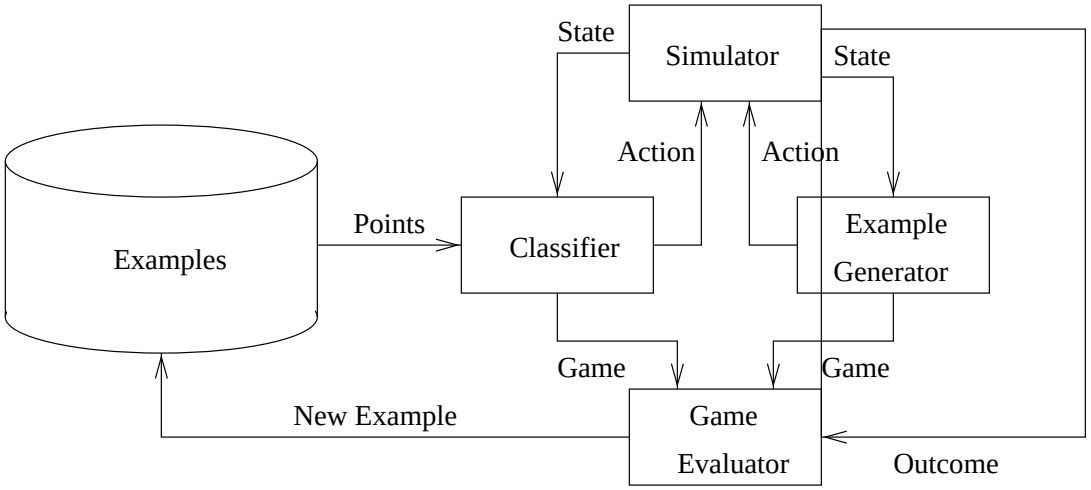


Figure 4.4: Architecture for memory based learning.

To construct an initial database of instances, the simulator generated actions randomly until E evaded P for a complete game. The corresponding state-action pairs for that engagement were then stored. At that point, 1-NN was used for future games. States were passed by the simulator to a classifier which searched the database for the nearest neighbor and selected an action by averaging the associated actions. If 1-NN failed to produce a game that ended in successful evasion, the game was replayed with the example generator randomly selecting actions until play ended in evasion. Once evasion occurred, the corresponding sequence of states and actions (i.e., the complete game) was stored in the database.

Evasion usually occurred after 20 time steps since it was rare in the memory-

based learner that the pursuers' speeds dropped below the threshold. Thus a stored game typically consisted of 20 state-action pairs. Our implementation uses Euclidean distance to find the nearest neighbor and the arithmetic mean of their control values to determine the appropriate actions. Distance is computed as follows:

$$dist(state, instance) = \sqrt{\sum_{\forall attrib} (state_{attrib} - instance_{attrib})^2}$$

Then the nearest neighbor is determined simply as

$$nn = \arg \min_{\forall instance} \{dist(state, instance)\}$$

If E fails to evade when using the stored instances, we reset the game to the starting position and generate actions randomly until E succeeds. We also generate random actions with probability 0.01 regardless of performance. The resulting set of examples is added to the database.

For the initial experiments using k-nearest neighbor, we varied k between 1 and 5 and determined that $k = 1$ yielded the best performance. (This was not completely surprising in that averaging control values with $k > 1$ tended to "cancel out" values that were extreme. For example, if three instances indicated turns of 90 degrees left, 5 degrees right, and 85 degrees right, the selected action would have been no turn. Of course, we are averaging "cyclic" values where, for example, 359 degrees is close to 1 degree. Improving the averaging process might enable $k > 1$ to perform better.) Examples consisted of games generated with random actions that resulted in success for E; thus we could assume that at least some of E's actions were correct. (In random games, every action taken by E

is selected at random according to a uniform probability distribution; the database is not checked for nearby neighbors.)

4.3.3 GA for Evasive Maneuvers

Grefenstette *et al.* demonstrated that genetic algorithms perform well in solving the single pursuer game. Typically, GAs use rules called classifiers, which are simple structures in which terms in the antecedent and the consequent are represented as binary attributes [50, 167]. The knowledge for the evasive maneuvers problem requires rules in which the terms have numeric values; we therefore modified the standard GA representation and operators for this problem, using a formulation similar to [153].

We call a set of rules a *plan*. For the GA, each plan consists of 20 rules with the general form:

$$\begin{aligned} \text{IF} \quad & \text{low}_1 \leq \text{state}_1 \leq \text{high}_1 \wedge \ldots \wedge \text{low}_n \leq \text{state}_n \leq \text{high}_n \\ \text{THEN} \quad & \text{action}_1, \ldots, \text{action}_m \end{aligned}$$

Each clause in the antecedent compares a state variable to a lower and upper bound. "Don't care" conditions can be generated by setting the corresponding range to be maximally general. To map this rule form into a chromosome for the GA, we store each of the attribute bounds followed by each action. For example, suppose we have the following rule (for the single pursuer problem):

$$\text{IF} \quad 300 \leq \text{speed} \leq 350 \wedge$$

$25 \leq$ previous turn $\leq 90 \wedge$

$3 \leq$ clock $\leq 10 \wedge$

$875 \leq$ pursuer speed $\leq 950 \wedge$

$8 \leq$ pursuer bearing $\leq 10 \wedge$

$180 \leq$ pursuer heading $\leq 270 \wedge$

$300 \leq$ pursuer range ≤ 400

THEN turn = 45

The chromosome corresponding to this rule would be:

[300 350 25 90 3 10 875 950 8 10 180 270 300 400 45]

Associated with each rule is a rule strength, and associated with each plan is a plan fitness. A population may contain up to 50 plans, all of which compete against each other in the GA system. Strength and fitness values, described below, determine the winners of the competition.

The genetic system consists of two major components: an inference system and a learning system (Figure 4.5). The inference system consists of a rule matcher and a rule specializer. The rule matcher examines the set of rules in the current plan to determine which rule to fire, and it selects the rule or rules with the most matches. In the event of a tie, it selects one of the tied rules using a random selection scheme based on the strengths of the rules.

Initially, all rules are maximally general. As a result, all rules will match all states,

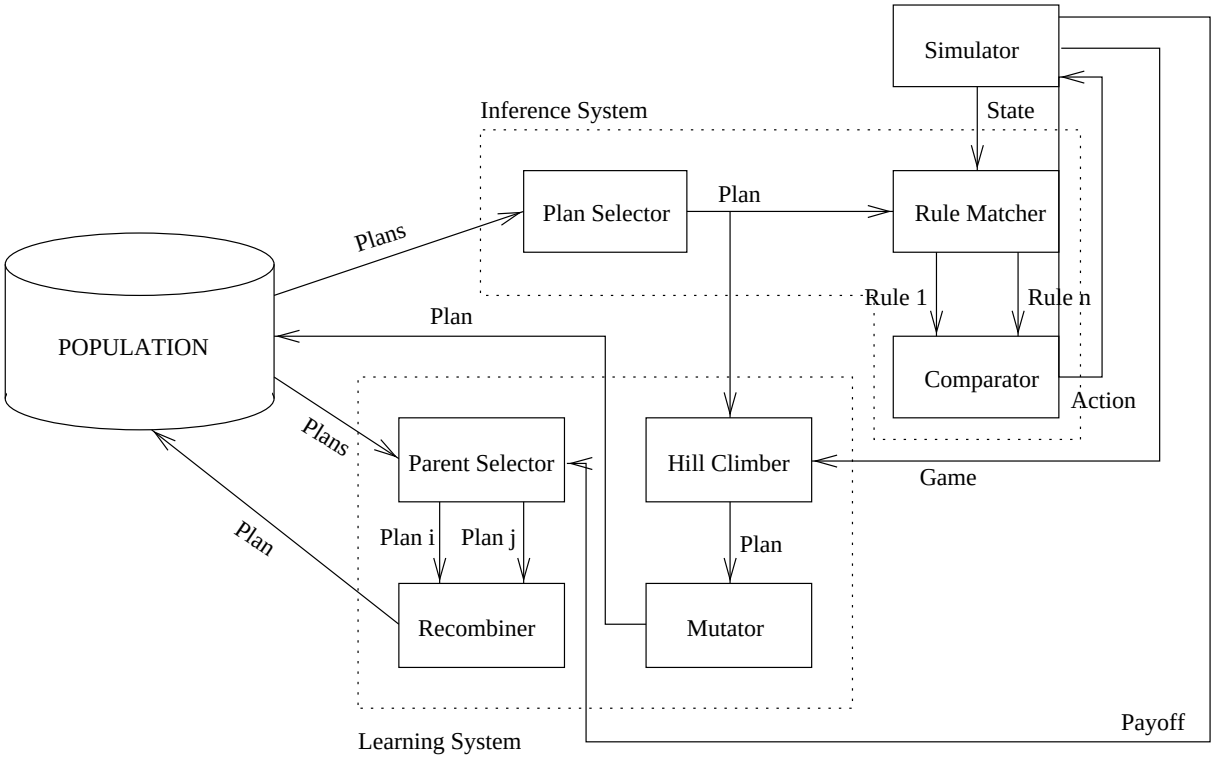


Figure 4.5: Architecture for genetic algorithm learning.

and one rule will be selected with uniform probability. Following each training game, the rules that fired are generalized or specialized by the GA, using hill-climbing to modify the upper and lower limits of the tests for each state variable as follows:

$$LB_i = LB_i + \beta(state_i - LB_i)$$

$$UB_i = UB_i - \beta(UB_i - state_i)$$

where LB_i and UB_i are the lower and upper bounds, respectively, of the rule that fired for $state_i$ and β is the learning rate. If the current state is within the bounds of the predicate, the bounds shift closer to the state based on the learning rate ($\beta = 0.1$ for this study). On the other hand, if the state is outside the bounds, only the nearer bound is adjusted by shifting it toward the value $state_i$. Following a game the strengths of the rules that fired are updated based on the payoff received from the game (the same payoff used in Q-learning).

Given the payoff function, the strength for each rule that fired in a game is updated using the profit sharing plan [149] as follows:

$$\begin{aligned} \mu(t) &= (1-c)\mu(t-1) + c\ \rho \\ \sigma(t) &= (1-c)\sigma(t-1) + c(\mu(t)-\rho)^2 \\ strength(t) &= \mu(t) - \sigma(t) \end{aligned}$$

where c is the profit sharing rate ($c = 0.01$ for our experiments), ρ is the payoff received, μ is an estimate of the mean strength of a rule, and σ is an estimate of the variance of rule strength. Plan fitness is calculated by running each plan against a set of games generated

at random according to a uniform probability distribution, and computing the mean payoff for the set of tests. During testing, the plan with the highest fitness is used to control E.

The heart of the learning algorithm lies in the application of two genetic operators: *mutation* and *crossover*. Rules within a plan are selected for mutation using *fitness proportional selection* [143]. Namely, probability of selection is determined as

$$\Pr(r) = \frac{strength_r(t)}{\sum_{\forall s \in rules} strength_s(t)}$$

where *rules* is the set of rules in a plan and r is the rule of interest. Probability of selection for plans is determined similarly using plan fitness rather than rule strength.

After selection, each clause on the left hand side of a rule is mutated according to a fixed mutation probability. Clause mutation results in one of the bounds being changed at random, keeping the ranges consistent (i.e., if the lower bound is changed to be greater than the upper bound, then the bounds are swapped). The mutated rule then replaces the rule with the lowest strength in the same plan. We selected a mutation probability of 0.01.

Crossover operates between plans. After each engagement, two plans are selected for crossover using fitness proportional selection, with a likelihood determined by the crossover rate. The rules in these plans are sorted by strength, and a new plan is generated by selecting m rules from one plan and $(20 - m)$ rules from the other plan. The new plan replaces the least fit plan in the population. Our crossover probability was 0.8.

In implementing a GA for sequential decision problems, many interesting open problems remain. First, several hill-climbing operators are available for supplementing

the search process. Our algorithm only modified the bounds on the state variables, but other options include rule merging, rule splitting, rule deletion, and rule insertion. Second, the credit assignment problem remains a significant problem. We chose to use the profit sharing plan, but other methods such as bucket brigade and combined methods could be used instead. In one combined approach, "bridges" may be constructed between distance points in a chain of rules that fire. Normal bucket-brigade is used through the complete chain and across the bridges, which results in more rapid reinforcement of rules that fired early in the process.

4.4 Results

For each of the algorithms and for both variations of the evasive maneuvers game, we ran ten experiments. To produce learning curves, we combined the results of the ten experiments by averaging the algorithm's performance at regular intervals. We estimated the accuracy of each algorithm by testing the results of training on 100 uniformly distributed, randomly generated games.

Uniform sampling of the perimeter for starting the game was selected to provide good coverage of possible games. In the single pursuer game, if we had partitioned the starting positions into 100 evenly spaced positions, these positions would have only differed by 3.6°. In the two pursuer game, using the same resolution would require 4,950 starting configurations. Using 100 random configurations sampled 2% of the space which may raise a concern that failed configurations (i.e., configurations where E is not able to evade) may

not have been sufficiently sampled. We believe this is not the case for the following reason.

When a game is started, $P1$ and $P2$ head directly at E. Given the size of the airspace and the speed of the players, the only way for E to be captured on the first move is if E heads directly at one of the pursuers at full speed. Selecting a heading, initially, away from both pursuers (i.e., at an angle that bisects the angle between the pursuers) will force the paths of the pursuers to begin to converge. In the worst case, the initial positions of the pursuers will be at 180° angles. If E moves at a 90° angle to both pursuers, then both pursuers will fall behind E. Since the pursuit strategy is to anticipate a *future* position of E and aim there, the relative angle of attack for the two pursuers will be even less than if they aimed directly at the current position of E. This actually gives E an advantage and enables E to further reduce the angles of attack until the problem reduces to a single pursuer game which, as we will see below, can always be won by E [4].

The results of the Q-learning experiments were encouraging and led to the next phase of our study in which we applied a traditional memory-based learning method, 1-nearest neighbor (1-NN), to the evasive maneuvers task. When we found that 1-NN did not work well, we considered an eager learning algorithm, the genetic algorithm. This choice was motivated by the previous work by Grefenstette *et al.* which indicated the GA was capable of solving this type of task [153]. In fact, we were able to replicate those results for the one-pursuer problem and scale up the GA so that it still worked quite well for the

[4] We will see in Chapter 5 that, in fact, we are able to learn strategies for E that result in 99% to 100% evasion when tested against 10,000 random games. Given starting configurations at 3.6° intervals, 10,000 games provides a good sampling of these initial configurations.

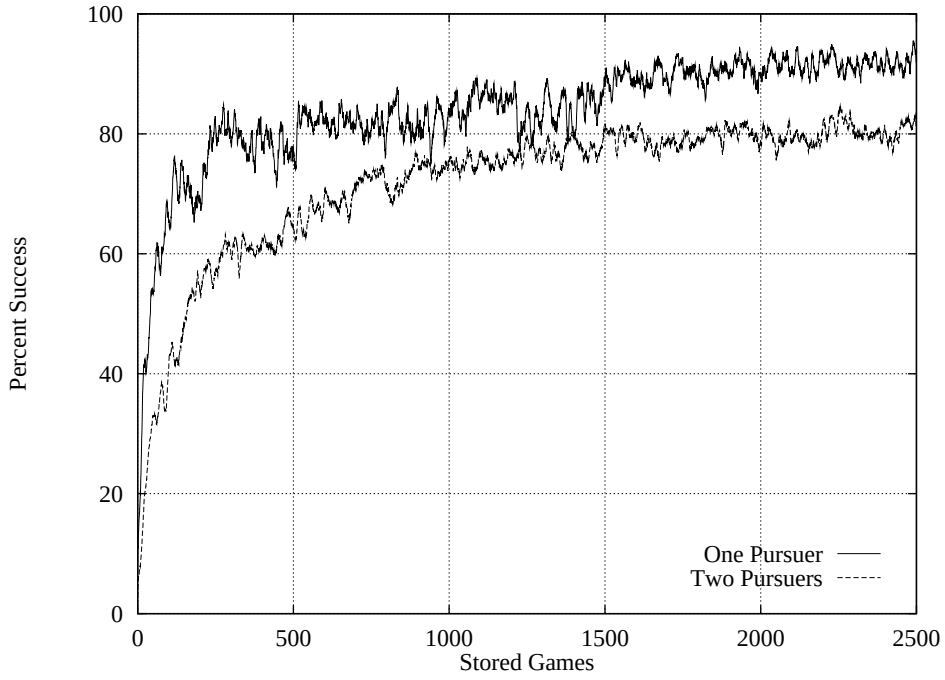


Figure 4.6: Performance of Q-learning on one- and two-player pursuit games.

two-pursuer game.

4.4.1 Performance of Q-Learning

In the one-pursuer task, Q-learning did extremely well initially (Figure 4.6), reaching 80% evasion within the first 250 games (i.e., approximately 5000 examples, given up to 20 examples per game), but then performance flattened out. Peak performance (when the experiments were stopped) was about 90%. There was an apparent plateau between 250 games and 1500 games where performance remained in the range 80%–85%. Then performance jumped to another plateau at 90% for the remainder of the experiment.

Q-learning's performance on the two-pursuer task was also encouraging. It reached 60% evasion within 250 games and continued to improve until reaching a plateau at 80%. This plateau was maintained throughout the remainder of the experiment. Because our implementation of Q-learning uses a form of memory-based learning, these results led us to believe it might be possible to design a more traditional memory-based method (i.e., k-NN) to solve the evasion task. At first, however, our hypothesis was not supported, as we see in the next section.

4.4.2 Performance of 1-NN

Figure 4.7 shows how well 1-NN performed on the two versions of the evasive maneuvers game as the number of training examples (and games) increased. This figure compares the performance on the two problems with respect to the number of games stored, where a game contains up to 20 state-action pairs as examples. These experiments indicate that the problem of escaping from a single pursuer is relatively easy to solve. Nearest neighbor developed a set of examples that was 95% successful after storing approximately 1,500 games, and it eventually reached almost perfect performance. The distance between P and E at the start of the game guarantees that escape is always possible. However, the results were disappointing when E was given the task of learning how to escape from two pursuers. In fact, the memory-based learning approach had difficulty achieving a level of performance above 45%. This demonstrates that the two-pursuer problem is significantly more difficult for 1-NN.

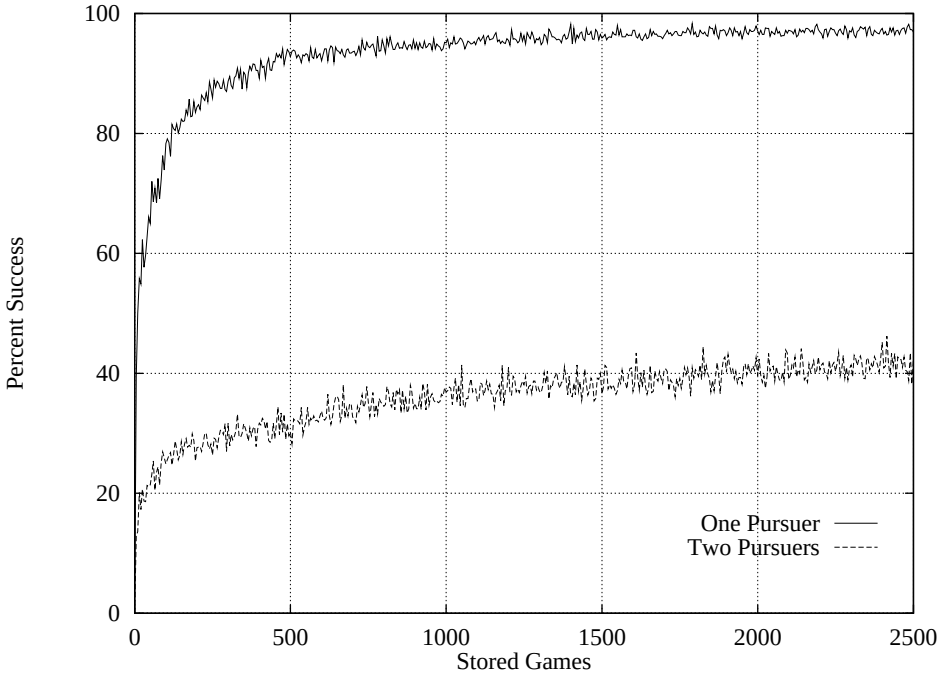


Figure 4.7: Performance of 1-NN on one- and two-player pursuit games.

One possible reason for 1-NN's poor performance on the two-pursuer task is presence of irrelevant attributes, which is known to cause problems for nearest neighbor algorithms [5, 293]. We experimented with a method similar to stepwise forward selection [112] to determine the set of relevant attributes. However, determining relevant attributes in a dynamic environment is difficult for the same reason that determining good examples is difficult: we do not know which attributes to use until many successful examples have been generated.

Another possible reason for the poor performance of 1-NN on the two pursuer task is the size of the search space. For the one-pursuer problem, the state space contains $\approx 7.5 \times 10^{15}$ points, whereas for two-pursuer evasion, the state space has $\approx 2.9 \times 10^{33}$ points. The one-pursuer game showed good performance after 235 games; to achieve similar coverage of the state space in the two-pursuer game would require roughly 5.4×10^{22} games (assuming similar distributions of games in the training data).

But the most likely reason for 1-NN's troubles, we concluded, was that we were generating bad examples in the early phases of the game. As stated above, a memory-based learner needs to have the "correct" action, or something close to it, stored with almost every state in memory. Our strategy for collecting examples was to play random games at first, and to store games in which E succeeded in escaping. However, many of the actions taken in these random games will be incorrect. E might escape because of one or two particularly good actions, but a game lasts for 20 time steps, and all 20 state-action pairs are stored. Our memory-based learning approach had no way (at first—see Section 5.4) to throw away

examples; therefore, collecting too many bad examples could have resulted in it getting stuck forever at the low level of performance.

4.4.3 Performance of the GA

We show the results of the GA experiments in Figure 4.8. As with 1-NN, the GA performs well when faced with one pursuer. In fact, it achieves near perfect performance after 15,000 games and very good performance (above 90%) after only 5,000 games. The number of games is somewhat inflated for the GA because it evaluates 50 plans during each generation, thus we counted one generation as 50 games. In fact, the simulation ran for only 500 generations (i.e., 25,000 games) in these experiments.

The most striking difference in performance between 1-NN and the genetic algorithm is that the GA learned excellent strategies for the two-pursuer problem, but nearest neighbor did not. Q-learning's performance, though much better than 1-NN, is still inferior to the GA. Indeed, the GA achieved above 90% success after 16,000 games (320 generations) and its success rate continued to improve until it reached approximately 95%.

4.4.4 Comparing One- and Two-Pursuer Evasion

Figure 4.9 shows a sample game in which E evades a single pursuer, which gives some intuition of the strategy that E had to learn. Essentially, E just keeps turning sharply so that P will be unable to match its changes of direction. Figure 4.10 then shows a sample game in which E evades two pursuers. Intuitively, the strategy by E is the same except

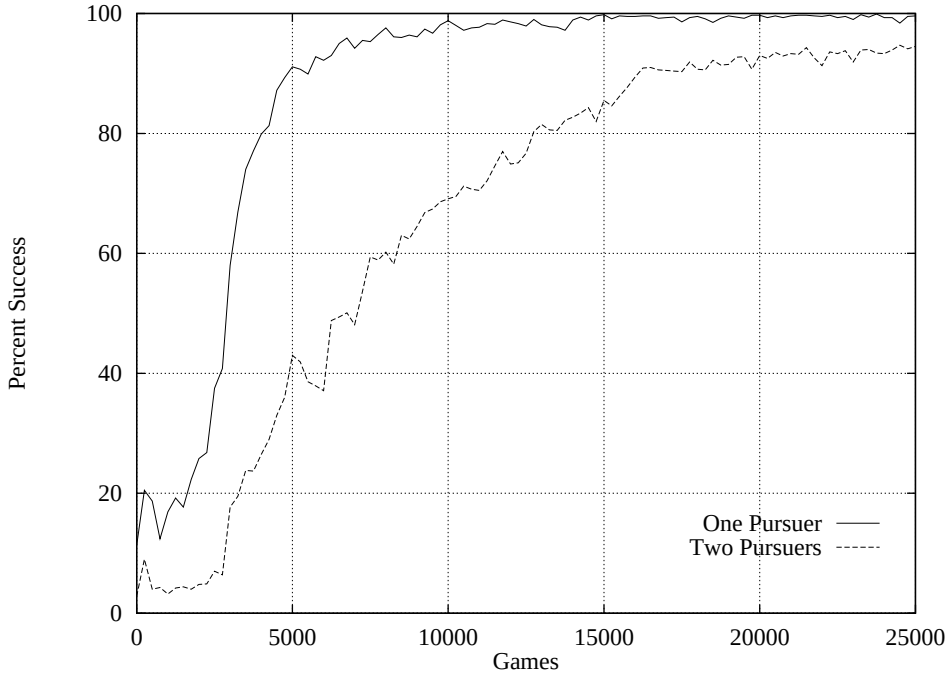


Figure 4.8: Performance of the genetic algorithm on one- and two-player pursuit games.

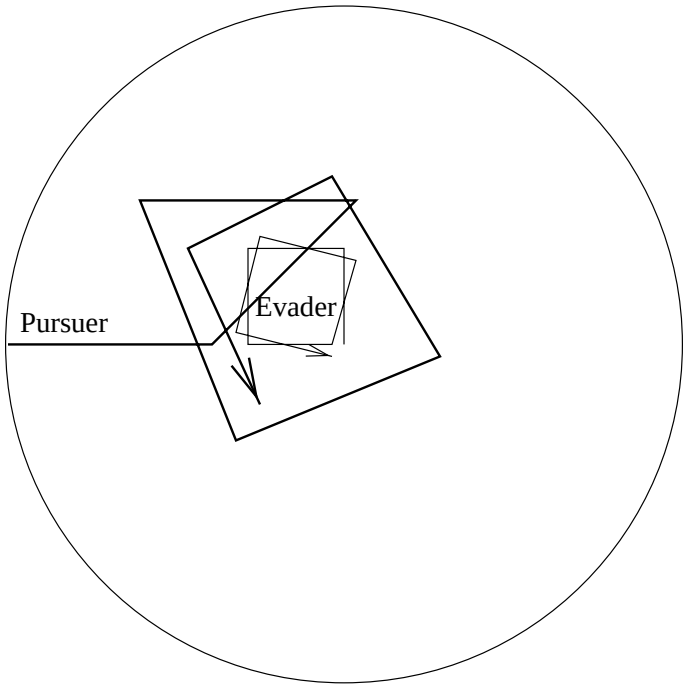


Figure 4.9: Sample game where E successfully evades one pursuer.

the goal for E is to have the trajectories of both $P1$ and $P2$ collapse to a single trajectory. At that point, the problem reduces to a single-pursuer game. This collapse, in fact, is experienced in this game.

Although all three algorithms did well on the single-pursuer task, a closer examination of the results reveals some interesting differences. Nearest neighbor eventually reached a successful evasion rate of 97%–98%, and it reached 93% evasion after only 10,000 games. This was superior to Q-learning's asymptotic performance, and 1-NN performed better than the GA through 5,000 games. Of course, the GA eventually achieved near perfect performance. Q-learning also learned rapidly in the beginning, exceeding the GA's ability through the first 3,000 games, but then its learning slowed considerably. In fact, at the

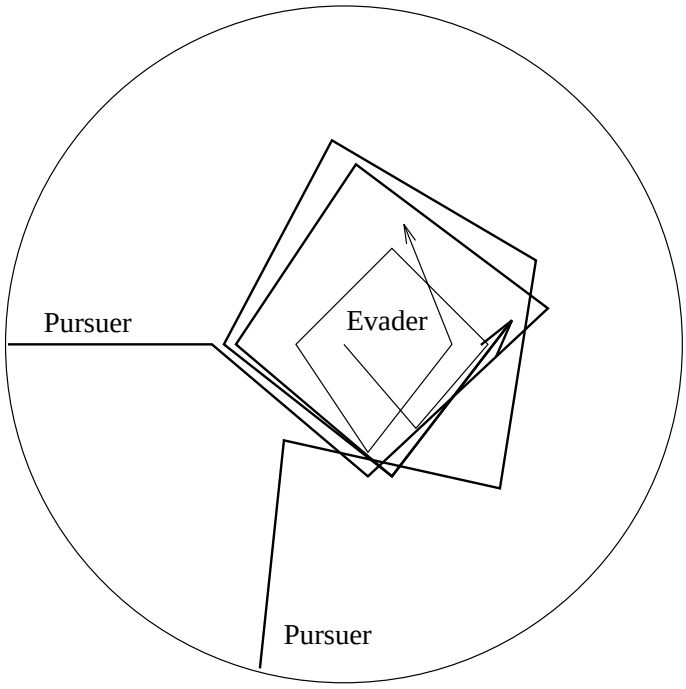


Figure 4.10: Sample game where E successfully evades two pursuers.

Table 4.1: Comparing learning for the evasive maneuvers task at convergence.

Algorithm	One Pursuer	Two Pursuers
1-NN	96.9%	42.3%
Q-learning	93.3%	81.7%
GA	99.6%	94.5%

point the GA was performing nearly perfectly, Q-learning's performance was only around 85%. After twice as many games as the GA, Q-learning (now achieving 91% evasion) was still performing considerably poorer than both the GA and 1-NN.

Table 4.1 shows the results of comparing the three algorithms on the two evasion tasks at convergence. We considered the algorithms to have converged when they showed no improvement through 500 games (for 1-NN and Q-learning) or through 100 generations

(for the GA). Recognizing the difficulty of the two-pursuer task (relative to the one-pursuer task), we now see profound differences in the performance of the three approaches. (See Figure 4.10 for a sample game where E evades two pursuers.) As before, the GA started slowly, being outperformed by both 1-NN and Q-learning. After about 3,000 games, the GA began to improve rapidly, passing 1-NN almost immediately, and catching Q-learning after an additional 5,000 games. The end results show the GA surpassing both Q-learning (by a margin of 11%) and 1-NN (by a margin of 52%). The more striking result, though, is the poor performance of 1-NN for the two-pursuer game. In the next chapter, we set out to improve this figure.

4.5 Summary

In this chapter, we compared three distinct learning algorithms on two pursuit games. The purpose of this study was to explore the applicability of different types of learning approaches to a difficult problem of sequential decision making. Although it is difficult to provide a fair comparison between such diverse algorithms, we believe that the results are suggestive of likely advantages and disadvantages for each algorithm on this type of task.

We note that the types of games studied in this experiment are somewhat limited. Specifically, we defined all starting configurations of the games to place the pursuers at equal distances from the evader. While it is reasonable to expect E to detect P at the same distance, a more realistic game would have the pursuers approaching in a staggered configuration. In addition, except in a dogfight against multiple pursuers, the missiles

can usually be expected to originate from the same launch site. Finally, the dynamics of the players were not realistically modeled. In particular, we permitted instantaneous turns (within the turn limitations) rather than modeling angular acceleration and physical limitations of the players.

The most prominent result of the experiments reported in this chapter is that simple memory-based approaches such as nearest neighbor and traditional Q-learning approaches require some modification to work well in a difficult domain such as differential game playing. Specifically, problems with large state and action spaces had to be addressed for both algorithms. For nearest neighbor, the definition of a *class* had to be modified to include a near match and to permit multiple dimensions of the class (e.g., speed, angle, and countermeasures). For Q-learning, the traditional lookup table was not capable of modeling the state and action space; therefore, a function approximator (i.e., a memory-based method) was required to cover the space. In spite of problems anticipated in using memory based approaches to approximate value functions with Q-learning [53], strong performance was still demonstrated.

In the next chapter, we discuss an additional modification in which we couple a second learning algorithm—the genetic algorithm—to 1-NN as a teacher. Because the GA, when run on two difficult problems, consistently performs well on these problems, we expect the rules generated by the GA to be useful in filtering out the effects of irrelevant attributes and in generating strong positive examples. In Chapter 6, we provide another variation of nearest neighbor in which we associate Q values with each of the examples (similar to the

implementation of Q-learning described in this chapter); however, we focus on co-learning to approximate solutions to differential games.

Chapter 5

A Teaching Method for Memory-Based Control

5.1 Using a Teacher in Reinforcement Learning

When two people learn a task together, they can both benefit from the different skills that each brings to the table. The result is that both will learn better than they would have on their own. Likewise, machine learning methods should be able to work together to learn how to solve difficult problems. This chapter describes how a memory-based learning algorithm and a genetic algorithm can work together to produce better solutions than either method could produce by itself.

As our experiments show, we were successful at developing a method to solve the difficult reinforcement learning task of one airplane attempting to evade two missiles. The key idea behind our success was the combined use of both memory-based learning and GAs. We observed after comparing two memory-based methods (1-NN and an adaptation

of Q-learning) with genetic algorithms that memory-based methods can learn to solve the task, but were dependent on having good examples in the memory base. In this chapter, we demonstrate an improved learning agent that first uses a GA to generate examples, and then switches to 1-NN after reaching a certain performance threshold. Our experiments demonstrate significant improvement in the performance of memory-based learning, both in overall accuracy and in memory requirements, as a result of using these techniques. The combined system also performs better than the GA alone, demonstrating how two learning algorithms working together can outperform either method when used alone.

Initially, we were surprised with 1-NN's poor performance on the two-pursuer task. In an attempt to improve its performance, we considered how to provide "good" examples to 1-NN, based on our hypothesis that the primary cause of its poor performance is the poor quality of its training experiences. For memory-based learning to work effectively on control tasks, the stored examples must have a high probability of being good ones; i.e., the action associated with a stored state should be correct or nearly correct. Determining the value or worth of examples is an instance of the credit assignment problem. Because of this credit assignment problem, and because of the difficulty of the tasks we studied, initial training is difficult for a memory-based learner. In contrast, a GA initially searches a wide variety of solutions, and for the problems we studied tends to learn rapidly in the early stages. These observations suggested the two-phase approach that we adopted, in which we first train a GA and then use the trained GA to provide examples to bootstrap 1-NN.

It is reasonable to question whether it is possible to exceed the capabilities of the

GA on this task. Specifically, Chapter 4 demonstrated that the GA was capable of reaching approximatly 94.5% evasion against two pursuers, but is it even possible for the aircraft to evade the missiles the other 5.5% of the time? We will demonstrate in this chapter that it is possible to evade nearly 100% of the time.

5.2 Methods in Learning with a Teacher

Before discussing our approach to bootstrapping the learning process in 1-NN, we consider other work that has been done in the area of training and machine learning. We focus on work related specifically to reinforcement learning, recognizing that the concept of using a teacher is applicable in other teaching domains as well.

5.2.1 ACE/ASE

Early work in reinforcement learning focused on developing simple neural networks and linear evaluation functions to solve control problems. One of the most successful approaches incorporated two separate neuro-controllers coupled together. Barto, Sutton, and Anderson described an *adaptive critic element* that provides a predicted reinforcement signal to a separate *associative search element* responsible for determining appropriate actions in a control problem [36].

The associative search element (ASE) mapped state information into an appropriate control signal to maximize expected reinforcement. The reinforcement signal is used with information from previous actions to modify weights in a linear threshold unit. Specifically,

the output signal is determined as follows:

$$y(t) = f\left[\sum_{i=1}^{n} w_i(t)x_i(t) + \text{noise}(t)\right]$$

Generally, the function $f()$ implements a simple threshold function.

$$f(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$

Learning consists of updating the weights, w_i,

$$w_i(t+1) = w_i(t) + \alpha r(t) e_i(t)$$

where α is a learning rate, $r(t)$ is the reinforcement received at time t, and $e_i(t)$ is the *eligibility* at time t defined by

$$e_i(t) = \delta e_i(t-1) + (1-\delta) y(t-1) x_i(t-1)$$

where δ is a decay rate. The eligibility is analogous to a momentum term which tends to smooth learning and continue the learning progress in a particular direction.

Barto *et al.* argue that, in a delayed reinforcement task, intermediate reinforcement signals stabilize the learning process. Such intermediate signals are provided by the second element—the adaptive critic element (ACE). ACE begins by computing a prediction of future reinforcement, given as

$$p(t) = \sum_{i=1}^{n} v_i(t) x_i(t)$$

where v_i are updatable weights, similar to ASE. These weights are updated using the following:

$$v_i(t+1) = v_i(t) + \beta[r(t) + \gamma p(t) - p(t-1)]\bar{x}_i(t)$$

where β is a learning rate, γ is a discount factor, and $r(t)$ is the current reinforcement signal. Similar to ASE, ACE also has an eligibility trace, this one given by

$$\bar{x}_i(t) = \lambda \bar{x}_i(t-1) + (1-\lambda) x_i(t-1)$$

where λ is a decay rate. Given the prediction of future reinforcement, ACE then calculates an estimate of current (i.e., intermediate) reinforcement that replaces $r(t)$ is the ASE equations. This estimate is given by

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1)$$

This method was one of the first reinforcement learning methods that used two separate learning structures to complement one another. Although not traditionally regarded as a teaching architecture, this method suggests that one learning method can be used to "bootstrap" another learning method. We describe such an approach in Section 5.3.

5.2.2 Adding a Teacher in Reinforcement Learning

Extending the work of Barto *et al.* on ACE/ASE, Clouse and Utgoff provided a method for incorporating a separate teacher to guide the learning process [81]. They also explored a modification to Q-learning using a similar teacher. In both cases, eligibility traces were used to smooth the learning process. A separate teacher monitored the performance of the learner, and when it felt intervention was necessary, the control signal provided by the learner was replaced with the control signal from the teacher. At that point, the eligibility traces were restarted using the teacher's control signal.

Specifically, the ASE/ACE weight update equations are modified such that, when the teacher's signal is used, the eligibilities are reset using the teacher's signal as follows:

$$v_i(t+1) = v_i(t) + \beta\hat{r}(t)x_i(t)$$

and

$$w_i(t+1) = w_i(t) + \alpha\hat{r}(t)y(t)x_i(t)$$

The current value of $x_i(t)$ also becomes the starting value for $\bar{x}_i(t)$, and $y(t)x_i(t)$ becomes the starting value for $e_i(t)$.

Their implementation of Q-learning uses several linear threshold units with update equations similar to the ACE equations given above. As with ACE, their implementation of Q-learning includes an eligibility trace which is reset by the teacher. The reinforcement learning with a teacher was applied to two control problems—the cart and pole problem and a race track navigation problem. In both cases, significant speedup over using reinforcement learning without a teacher was demonstrated.

5.2.3 Phases of Learning

Research by Dorigo and Colombetti focused on what they call the "three stages of development" consisting of a baby phase, a young phase, and an adult phase [84, 85, 115]. During the *baby phase*, immediate reinforcement is provided by a trainer after every action. This continues until the learner reaches some performance threshold. At that point, the learner passes into the *young phase*. In this phase, reinforcement is provided only by the environment and is generally delayed. After a second performance threshold is passed, the agent

passes into the *adult phase* in which no learning takes place—only the learned strategies are applied.

Dorigo and Colombetti applied a distributed classifier architecture, which they call ALECSYS, to demonstrate this approach to learning. They implemented ALECSYS in a simulated and a real robot, which they call AutonoMouse. Due to training limitations in the real robot, they applied two approaches to reinforcement—reward-the-intent and reward-the-result. The former method compares actions to an ideal environment, thus relying on an external oracle to evaluate performance. The latter method takes the actual reinforcement taken from the environment. These methods were tested on several navigational experiments in which the robot attempt to approach a light or, depending on a noise signal, decide to either approach a light or flee from the light. The most interesting result from this work is that they succeeded in training an actual robot using both approaches rather than limiting their experiments to a simulated environment.

5.2.4 Incorporating Advice Into Reinforcement Learners

Other work in teaching and reinforcement learning has been focusing on coupling external advice into the learning process as a way of speeding up learning. One approach studied by Gordon and Subramanian uses explanation based learning to devise a set of operational rules for obstacle avoidance in a noisy environment [146, 147]. In this approach, high-level domain-specific and spatial knowledge is coded into rules and then operationalized into low-level reactive rules based on the stated goals of the problem. These rules generally are

not sufficient to solve the problem but provides useful rules in various situations. Gordon and Subramanian use these initial rules to seed a genetic algorithm which then refines the rules to solve the specific problem.

Note this approach differs from previous teaching approaches in that only high-level advice is provided rather than specific instructions to be carried out during the task. Maclin and Shavlik also provide an advice-taking reinforcement learner [218]. In their system, they use a connectionist Q-learner that interacts with an external advisor in five steps:

1. The agent requests and receives advice from the advisor.
2. The advice is converted into an internal representation.
3. The advice is converted into a usable form.
4. The reformulated advice is integrated into the agent's knowledge.
5. The value of the advice is evaluated.

Maclin and Shavlik's connectionist Q-learner uses the notion of a "knowledge-based neural network" KBANN. In a KBANN network, each node of the network represents a Boolean concept or proposition. Connections of the network emulate AND and OR connectives by assigning weights and biases to yield the desired effect. Hidden nodes are incorporated to capture information on state. Thus, as advice is received, that advice is encoded as a KBANN network and attached to the current network. Weights are modified from there using the Q-update procedure.

The KBANN advice-taker was tested on the Pengo task as studied by Chapman and Agre (called Pengi in their work) [75]. Pengo involves an agent moving obstacles in a field with food and enemies. The object is to collect the most food. Obstacles hitting food destroys food, and enemies can capture food as well. Typically, the enemies follow a fixed strategy. When evaluated on the ability to collect food, eliminate enemies, and avoiding enemies, Maclin and Shavlik found the advice-taker able to survive longer and to collect more food than when advice was not used.

5.2.5 Learning with a Helpful Teacher

Given that approaches can be devised in which a teacher can be coupled with a machine learning algorithm to learn a set of concepts or to learn a task, an interesting question arises as to how effective the teacher must be for the learner to learn that task. Much work in the area of computational learning theory has focused on the problem of assessing the complexity of a learning task in terms of the samples required to learn that task (called the *sample complexity*).

Salzberg *et al.* and Heath [296, 166] considered an alternative to the Probably Approximately Correct (*PAC*) learning model [353, 354] in which a *Helpful Teacher* is providing good examples to the learning. Their research focused on answering four questions:

1. What is the minimum number of examples needed to learn a concept?

2. What is the minimal representation for a given concept?

3. What is the most effective way to teach a concept?

4. What is the effect of providing additional examples to a partially constructed representation?

The Helpful Teacher knows the algorithm to be used by the learning and provides examples to the learning in the best way possible for learning the task. In this model, "best" is defined to be the minimal set of examples represented in an appropriate fashion for the learner to learn the task.

Salzberg *et al.* and Heath apply this model to determining best case sample sizes for nearest neighbor classification when learning several geometric concept classes. For example, they prove that only two examples are required for nearest neighbor to learn a concept exactly defined by a single hyperplane in any number of dimensions. They also prove that, given a convex polytope with n faces, only $n + 1$ examples are required to learn the concept exactly. They also apply the approach to approximate learning in general function approximation and demonstrate the approach with nearest neighbor applied to learning a parabola.

The significance of this research lies, not with the definition of a new approach to teaching in machine learning, but in characterizing the benefits of teaching in terms of the complexity of the underlying task. The approach was motivated by the fact that many results based on *PAC*-learning provided sample complexities that were significantly higher than experimental results. This is because *PAC*-learning provides a worst-case analysis. The Helpful Teacher model, on the other hand, provides a best-case analysis and helps bound the complexity of the learning task. In fact, Salzberg and Heath have demonstrated that

many experimental results tend to more closely follow Helpful Teacher complexity bounds than *PAC* complexity bounds [166].

5.3 Bootstrapping Nearest Neighbor

Given the potential benefit of using a teacher in learning tasks, we sought to provide a teacher to 1-NN to improve 1-NN's performance. We did not have *a priori* knowledge of the correct actions for the evasive maneuvers task with two pursuers, therefore, we decided to use a second learning algorithm to bootstrap 1-NN. Our bootstrapping idea requires that one algorithm train on its own for a time, and then communicate what it has learned to the second algorithm. At that point, the second algorithm takes over. Later, the first algorithm adds additional examples. This alternation continues until the combined system reaches some asymptotic limit. Because the GA learned much better for the two-pursuer game, we selected it as the first learner, with 1-NN second. Details of the communication or "teaching" phase are given in Figure 5.1. Using this approach, the examples continue to accumulate as the genetic algorithm learns the task.

The results of training 1-NN using the GA as the teacher are shown in Figure 5.2. We call this system *GAMB* because it first uses a GA and then uses a memory-based learning algorithm (i.e., 1-NN). All points shown in the graph are the averages of 10 trials. The first threshold was set to 0%, which meant that the GA provided examples to 1-NN from the beginning of its own training. The second threshold was set to 50% to permit the GA to achieve a level of success approximately equal to the best performance of 1-NN

```
algorithm GAMB;
    init population;
    do
        run genetic algorithm;              /* Run the GA for one generation */
        perf = select best plan;            /* Determine performance of GA */
        if perf ≥ θ then                    /* Evaluate performance against θ */
            do i = 1, n                     /* For our experiments n = 100 */
                evade = evaluate best;      /* Determine if best plan evades */
                if evade then
                    store examples;         /* Stores up to 20 examples */
                endif;
            enddo;
        endif;
    enddo;
    evaluate memory-base;                   /* Test on 100 games */
end;
```

Figure 5.1: Pseudocode for *GAMB*.

on its own. Thus only plans that achieved at least 50% evasion were allowed to transmit examples to 1-NN. Finally, the threshold was set at 90% to limit examples for 1-NN to games in which a highly trained GA made the decisions about which examples to store.

When $\theta = 0\%$, *GAMB* almost immediately reaches a level equal to the best performance of 1-NN on its own (around 45%). From there, it improves somewhat erratically but steadily until it reaches a performance of approximately 97% success. The figure shows performance plotted against the number of examples stored. The number of examples stored here is higher than the number of examples stored for 1-NN alone. If we halt learning after 50,000 examples (which is consistent with the earlier 1-NN experiments), performance would be in the 85% range, still an enormous improvement over 1-NN's performance, but not better than the GA on its own.

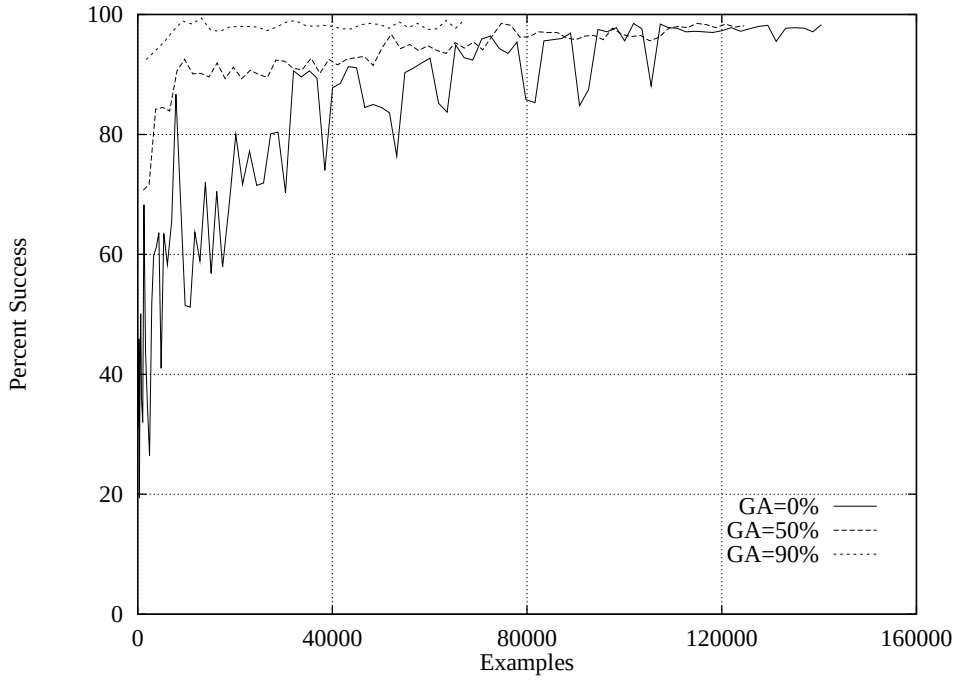


Figure 5.2: Results of GA teaching 1-NN.

When $\theta = 50\%$, *GAMB* starts performing at a high level (above 70%) and quickly exceeds 90% success. After 50,000 examples, *GAMB* obtained a success rate above 95%, with some individual trials (on sets of 100 games generated according to a uniform distribution) achieving 100% success. In addition, the learning curve is much smoother, indicating that 1-NN is probably not storing many "bad" examples. This confirms, in part, our earlier hypothesis that 1-NN's fundamental problem was the storage of bad examples. If it stores examples with bad actions, it will take bad actions later, and its performance will continue to be poor whenever a new state is similar to one of those bad examples.

Finally, with $\theta = 90\%$, *GAMB*'s performance was always superb, exceeding the GA's 90% success rate on its first set of examples. *GAMB* converged to near-perfect performance with only 10,000 examples. One striking observation was that *GAMB* performed better than the GA throughout its learning. For example, when $\theta = 0\%$, *GAMB* achieved 50–80% success while the GA was still only achieving 2–10% success. Further, *GAMB* remained ahead of the GA throughout training. Even when $\theta = 90\%$, *GAMB* achieved 98–100% evasion while the GA was still only achieving around 95% evasion. Neither the GA nor 1-NN were able to obtain such a high success rate on their own, after any number of trials.

5.4 Reducing Memory Size

Our bootstrapping algorithm, *GAMB*, performs well even when only a small number of examples are provided by the GA, and it even outperforms its own teacher (the GA) during

training. But the amount of knowledge required for the GA to perform well on the task was quite small—only 20 rules are stored as a single plan. The number of examples used by *GAMB*, though small in comparison with 1-NN, still requires significantly more space and time than the rules in the GA. Consequently, we decided to take this study one step further, and attempted to reduce the size of the memory store during the memory-based learning phase of *GAMB* [321, 377].

In the pattern recognition literature, e.g., in [98], algorithms for reducing memory size are known as *editing* methods. However, because memory-based learning is not usually applied to control tasks, we were not able to find any editing methods specifically tied to our type of problem. *GAMB* performs quite well as described above, and we would like to reduce its memory requirements without significantly affecting performance. We therefore modified a known editing algorithm for our problem, and call the resulting system *GAME* (GA plus memory plus editing).

5.4.1 Editing Methods for Nearest Neighbor

Early work by Wilson [373] showed that examples could be removed from a set used for classification, and suggested that simply editing would frequently improve classification accuracy (in the same way that pruning improves decision trees [235]). Wilson's algorithm classifies each example in a data set with its own k nearest neighbors. Those points that are incorrectly classified are deleted from the example set, the idea being that such points probably represent noise.

```
algorithm Edit-Incorrect;
    ∀i ∈ memory-base do;
        if class(i) ≠ class(k-NN(i)) then
            delete(i,memory-base);
        endif;
end;
```

Figure 5.3: High-level pseudocode of Wilson editing algorithm.

Procedurally, Wilson's algorithm worked as follows. Editing could proceed following storage of all examples or at periodic intervals. Wilson considered the case where all of the training examples had been stored. Then each example was classified using k-nearest neighbor as if the example did not exist in the database. Let *class*(i) return the class associated with instance i. Let *class*(k-NN(i)) return the class associated with the k nearest neighbors of instance i. If *class*(i) $\neq$ *class*(k-NN(i)), then delete instance i from the database. This is shown procedurally in Figure 5.3.

Tomek [349] modified this approach by taking a sample (> 1) of the data and classifying the sample with the remaining examples. Editing then proceeds using Wilson's approach. Specifically, Wilson's approach is modified as in Figure 5.4.

Ritter *et al.* [281] described another editing method, which differs from Wilson in that points that are *correctly* classified are discarded. The Ritter method, which is similar to Hart's [164], basically keeps only points near the boundaries between classes, and eliminates examples that are in the midst of a homogeneous region. Procedurally, this method is shown in Figure 5.5.

Finally, Aha *et al.* evaluate the differences between several instance filtering (i.e.,

```
algorithm Edit-Incorrect-Sample;
    sample = select(n,instance-base);
    memory-base = memory-base - sample;
    ∀i ∈ sample do;
        if class(i) ≠ class(k-NN(i)) then
            delete(i,sample);
        endif;
    enddo;
    memory-base = memory-base + sample;
end;
```

Figure 5.4: High-level pseudocode of Tomek editing algorithm.

```
algorithm Edit-Correct;
    ∀i ∈ memory-base do;
        if class(i) = class(k-NN(i)) then
            delete(i);
        endif;
    enddo;
end;
```

Figure 5.5: High-level pseudocode of Ritter editing algorithm.

editing) and instance averaging algorithms in instance-based learning [3, 5, 7, 6, 189, 190]. Aha's IB2 and IB3 algorithms apply a standard nearest-neighbor rule for classification, but then proceed to edit examples based on Ritter's approach. For Aha *et al.*, the assumption is that misclassification comes from noisy attributes in the data. Rather than focusing on the attributes themselves (for these studies), they attempted to retain examples with noise and discard examples without noise. In later studies, they provided several approaches to weighting the features as well [367].

Instance averaging algorithms provide techniques for replacing actual instances with "prototypical" instances derived from the data. In this approach, an initial instance for a concept receives a weight of 1.0. When a new instance for the concept is encountered, the both instances are replaced by a new instance determined by taking the weighted average of the attributes in the two instances. The weight of the prototype instance is simply the sum of the weights of t, the new instance, and nn, the previously stored instance. The weight for t is determined based on its distance from nn and frequently decreases exponentially as distance increases.

5.4.2 An Editing Algorithm for Evasive Maneuvers

The editing approach we took combined the editing procedure of Ritter *et al.* and a variation on the sampling idea of Tomek [111]. We began by generating ten example sets with $\theta = 90$ where each set consisted of a single set of examples from the GA. We then selected the set with the best performance on 10,000 test games, which in this case obtained nearly perfect

accuracy with 1,700 examples. Next we edited the memory base by classifying each example using all other examples in the set. For this phase, we used the five nearest neighbors. If a point was correctly classified, we deleted it with probability 0.25. (This probability was selected arbitrarily, and was used to show how performance changed as editing occurred.) Prior to editing and after each pass through the data, the example set was tested using 1-NN on 10,000 uniformly distributed random games.

To determine if a point was correctly classified, we found the k nearest neighbors for that point and examined the associated actions. One complication in "classifying" the points for editing was that the class was actually a three-dimensional vector of three different actions, two of which were real-valued (turn angle and speed) and one of which was binary (emitting smoke). It was clear that an exact match would be too strict a constraint. Therefore, the approach we took was to normalize the actions and determine the Euclidean distance between the target action and the neighboring actions. We said the actions "matched" if the Euclidean distance was less than a pre-specified threshold, ϕ. In these experiments, we set $\phi = 0.05$. If the point "matched" a majority of the neighbors, we considered that point to have been classified correctly.

5.4.3 Experimental Results

The results of running *GAME* on the 1,700 examples are shown in Figures 5.6 and 5.7. We used a logarithmic scale on the x-axis to highlight the fact that accuracy decreased slowly until almost all the examples were edited. When read from right to left, the graph

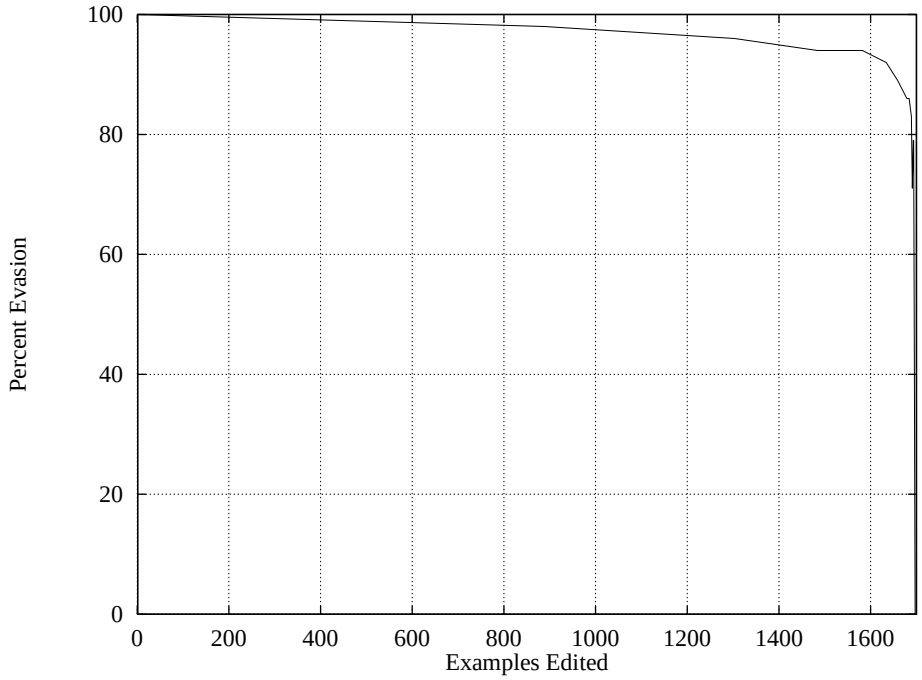


Figure 5.6: Results of editing examples provided by the genetic algorithm for 1-NN.

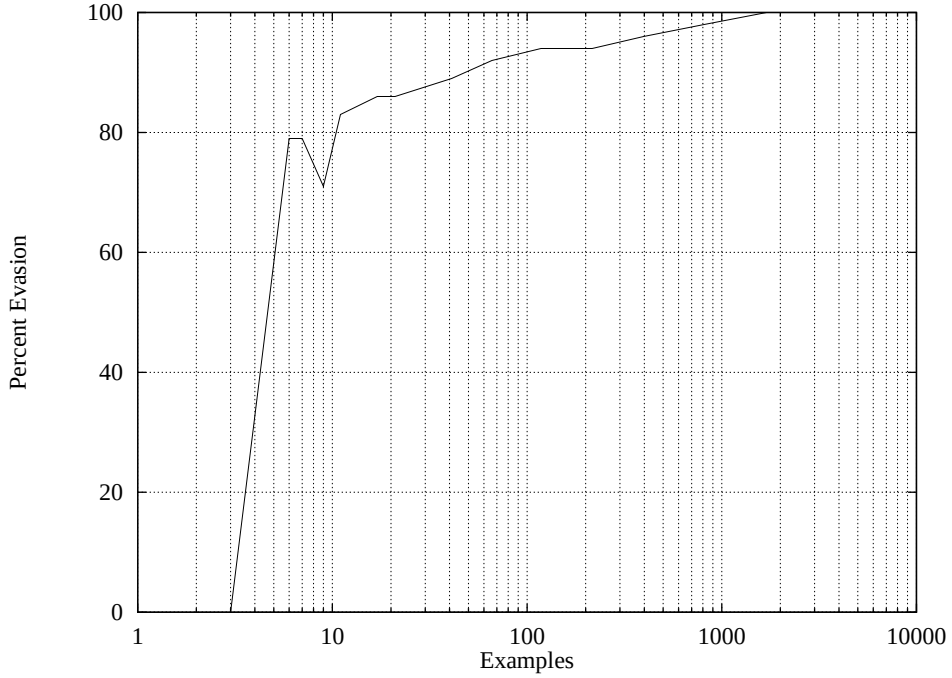


Figure 5.7: Number of examples compared to performance on evasive maneuvers in *GAME*.

in Figure 5.7 shows how accuracy decreases as the number of examples decreases. With as few as 11 examples, *GAME* achieved better than 80% evasion, which is substantially better than the best ever achieved by 1-NN alone. With 21 examples (comparable in size to a plan in the GA), *GAME* achieved 86% evasion. Performance remained at a high level (greater than 90% success) with only 66 examples. Thus it is clear that a small, well chosen set of examples can yield excellent performance on this difficult task. Furthermore, such a small memory base guarantees that the on-line performance of nearest neighbor will be quite fast.

The strong performance by *GAME* provides strong empirical evidence that it is possible for the aircraft to evade both missiles in all cases. In specific trials, *GAME* was able to evade 100% of the test games. Recall that the test games are generated at random according to a uniform probability distribution and that $10,000$ were generated for the test trials. If we consider selecting a starting position every $3.6°$ around the perimeter of the starting circle, and we consider these positions for two pursuers, we have 10,000 possible positions. Given this partitioning, we can determine further that there are only $4,950$ unique pairs of starting positions. Further, if we consider the symmetry of the game field, we can further reduce the number of starting configurations to $2,475$. Thus we claim that the $10,000$ test games is actually more than enough to sample the starting configurations. Thus, with 100% evasion, it must be possible to evade in all possible starting configurations of this game.

5.5 Summary

The experiments reported here show that it is possible to use a second learning algorithm in conjunction with memory based learning to produce agents that perform well on difficult delayed reinforcement learning problems. The experiments demonstrate clearly the power of having a teacher or other source of good examples for memory-based learning methods. For complex control tasks, such a teacher is probably a necessary component of any memory-based method. Our experiments demonstrated this power by using a genetic algorithm to learn plans or control laws in complex domains, which then trained a memory-based learner by using its learned rules to generate good examples. The result was a hybrid system that outperformed both of its "parent" systems. This hybrid approach can of course be applied in many ways; for example, standard Q-learning is notoriously slow to converge, and approaches such as ours could be used to accelerate it.

One surprising result was that the performance of *GAMB* outperformed the GA at the same point in training. We hypothesize this was because only the best examples of a given generation were passed to 1-NN, rather than all of the experiences of the GA during that generation. The fact that *GAMB* outperformed GA right away indicates that perhaps it could have been used to teach the GA, instead of the other way around. Taken further, perhaps a system in which the GA and 1-NN alternated in its role as teacher could yield superior performance yet.

In addition, we found that editing the example set produced a relatively small set of examples that still play the game extremely well. Again, this makes sense because

editing serves to identify the strongest examples in the database, given that poor examples were still likely to be included in the early stages of learning. It might be possible with careful editing to reduce the size of memory even further. This question is related to the theoretical work by Salzberg and Heath [166, 296] that studies the question of how to find a minimal-size training set through the use of a Helpful Teacher, which explicitly provides good examples (see Section 5.2.5). Such a Helpful Teacher is similar to the oracle used by Clouse and Utgoff [81] except that it provides the theoretically minimal number of examples required for learning.

Chapter 6

Memory-Based Co-Learning

6.1 Learning "Solutions" to Differential Games

In static games, the strategies of all players are generally known. Further, in solving static games, payoffs assigned to selected strategy combinations are also known. If the players do not know the payoffs, then learning these payoffs is a variation of the k-armed bandit problem in which all players are trading exploration (determining expected payoffs for each strategy combination) with exploitation (playing the optimal mixture of strategies based on current knowledge about expected payoffs).

In dynamic games, the strategies of all the players are *not* necessarily known. Thus it is impossible to assign payoffs to strategy combinations because the combinations are not known. Even though the players do not necessarily know the strategies of their opponents, generally they do know the dynamics of their opponents (i.e., they know the differential equations that characterize the performance of all of the players). As described above, this

leads to a solution concept in which the players attempt to characterize their strategies in terms of regions in strategy space based on the dynamics of the game and assign payoff to those regions.

Because the definition of a solution to any game (whether static or dynamic) involves determining strategies and expected payoffs for all players of the game, we wish to explore methods for the competing players to learn their optimal strategies on line. Fixing the strategy of one player while the other learns, is interesting from a control theory perspective. It is uninteresting from a game theory perspective, however, because the learning agent is focusing on a single competing strategy. As just described, usually competing strategies are unknown, and players must learn to maximize their expected payoff in the presence of this unknown. Also, in the case of learning strategies, the dynamics of the players are also unknown (i.e., the players do not know the differential equations characterizing performance).

6.2 A Memory-Based Co-Learning Algorithm

Traditionally, memory-based learning consists of storing examples of classification instances or previous experiences in a memory base (i.e., the memory) and searching the data base for "similar" examples when presented with a new problem. The action taken corresponds to the action associated with the closest example (or some combination of examples) in the data base. Other variants on the memory-based approach consist of constructing tables of state-action combinations and storing expected payoff with these table entries. Learning

```
algorithm MBCL;
        seed(memory);
        do
                game = play-game(memory);
                store-game(game,memory);
                update-Q-values(memory);
        enddo;
end;
```

Figure 6.1: High-level pseudocode of memory-based co-learning algorithm.

consists of updating the expected payoffs over time.

Our memory-based approach is based on a combination of these two variants. In particular, we use k-nearest neighbor to identify the examples that most closely match the current state of the game. We also use Q-learning to update expected payoff associated with each of the examples in the data base. A high-level description of the learning algorithm is shown in Figure 6.1.

The first step in the algorithm involves seeding the population with an initial set of games (*seed*). This step is needed because MBR cannot function without a pre-existing memory base. Seeding consists of generating several random games. Specifically, games are generated where one-third of the games apply random actions for both players, one-third of the games fix P on a single random move per game and generate random actions for E, and one-third of the games fix E on a single random move per game and generate random actions for P. In all cases, a uniform probability distribution was used for the random number generator. The initial Q values associated with each of the examples are largely irrelevant, so we assign the value of zero to all non-terminating moves and the actual payoff

```
algorithm play-game(memory);
    neighbors = find-neighbors(state,memory,k_s);
    max-E-move = find-max-move(neighbors,E);
    min-E-move = find-min-move(neighbors,E);
    max-P-move = find-max-move(neighbors,P);
    min-P-move = find-min-move(neighbors,P);
    moves-E = partition(min-E-move,max-E-move,n);
    moves-P = partition(min-P-move,max-P-move,n);
    for i = 1 to n
        for j = 1 to n
            move-neighbors = find-neighbors(moves-E[i],moves-P[j], k_m);
            expected-payoff[i][j] = sum_{p=1}^{k_m} w_p move-neighbors.rho_p
        endfor
    endfor
    E-move = linear-program(expected-payoff);
    P-move = dual-linear-program(expected-payoff);
    game[state] = update-state(E-move,P-move,state);
    return(game);
end;
```

Figure 6.2: Pseudocode for playing the differential game.

of the game to the terminating moves.

When playing the game, the moves for both players are determined by examining the memory base. This process occurs in three steps which are illustrated by the pseudocode in Figure 6.2. Following this procedure, in each state of the game, the k_s neighbors to the current state in the memory base are found. Next only the range of moves found among the neighbors are considered, and the moves are partitioned into n representative moves for each player. These moves are used to determine the k_m nearest neighbors based on stored moves from among the k_s neighbors found previously.

Because n moves are considered for each player, and we wish to compare the

performance of the two players on each of the pairs of moves, we can construct a payoff matrix of the form

$$\begin{pmatrix} E[\rho_{1,1}] & E[\rho_{1,2}] & \cdots & E[\rho_{1,n}] \\ E[\rho_{2,1}] & E[\rho_{2,2}] & \cdots & E[\rho_{2,n}] \\ \vdots & \vdots & \ddots & \vdots \\ E[\rho_{n,1}] & E[\rho_{n,2}] & \cdots & E[\rho_{n,n}] \end{pmatrix}$$

where $E[\rho_{i,j}]$ represents the expected payoff when pairing moves i and j. This matrix is the basis for the tableau used to solve the corresponding linear program (and its dual). The moves for the two players are then selected from the resulting mixed strategies. Specifically, because we quantize the possible actions to construct a tableau for linear programming, when one of these discrete actions is selected, an actual action is generated uniformly at random from the interval $[\text{action} - \delta, \text{action} + \delta]$, where δ is equal to one-half of the size of a partition.

For each entry in the matrix, we compute the expected payoff as $E[\rho_{i,j}] = \sum_{p=1}^{k_m} w_p \rho_p$. The weights w_p can be computed as,

$$w_p = e^{\left(\frac{-f_d(x_p, x_q)^2}{2K_w^2}\right)}$$

or

$$w_p = \frac{1}{1 + 20\left(\frac{f_d(x_p, x_q)}{K_w}\right)^2}$$

where K_w is a smoothing kernel width that determines the distance over which the weight is significant, and f_d is a distance function (usually based on normalized Euclidean distance). In our experiments, we used the exponential form.

When learning, we update the associated payoffs with the stored points using Q-learning. The points actually updated consist of the k_s nearest neighbors identified in each

state. As indicated above, the initial payoff stored with the points will be the actual payoff received from the first play. Subsequent updates will only occur if the point is one of the k-nearest neighbors in some state. In this case, the Q-learning update rule is applied.

$$Q(s_i, a_p, a_e) = (1 - \alpha_j)Q(s_i, a_p, a_e) + \alpha_j(\rho + \gamma Q(s', \pi(s')))$$

where $Q(s_i, a_p, a_e)$ is the current Q value associated with applying the pair of actions $\langle a_p, a_e \rangle$ in state s_i, η_j is a learning rate determined by the distance between the point being updated and the actual point encountered, γ is a discount factor, $Q(s', \pi(s'))$ is the maximum Q value in state s', and s' is the state resulting from applying actions $\langle a_p, a_e \rangle$.

It might appear that an additional pair of linear programs need to be solved to determine $Q(s', \pi(s'))$. In fact, we know this value from playing the game. At each step, we need to store $f^{\pi}(s_i)$, the result of solving the linear programs, since this will be used in subsequent updates to the Q values.

Another interesting benefit of applying this approach to learning is that the k-armed bandit problem is potentially "solved." In particular, because mixed strategies are generated at each point in playing a game, we apply the probability mixtures to determine actions to be taken. This automatically forces a level of exploration in the space without explicitly coding an exploration strategy. Of course, one possible drawback with relying on the mixed strategies for exploration is that, should estimates yield a pure strategy, no exploration will occur unless the reward estimates are modified to later produce a mixed strategy. For this reason, we still incorporate explicit exploration by permitting the players to select actions at random.

6.3 Experiments

To evaluate *MBCL*, we ran several experiments using four different differential games. These games include the simple game of force described in Section 2.7, the simple pursuit game described in Section 2.8, the extension to the simple pursuit game in which a boundary is placed at $x = 0$ (also described in Section 2.8), and a pursuit game in which both the pursuer (P) and the evader (E) have limited mobility. This latter game is an extension of the traditional Homicidal Chauffeur game which limits the mobility of P but not E. To reduce the size of the state space, all games are played with state variables relative to P. Further, all game matrices are constructed such that each player has $n = 10$ strategies.

6.3.1 A Game of Force

In Section 2.7, we described a differential game in which two players are applying a force to a falling object in an attempt to make the object land at a certain point. For player P, the objective is to push the object as far to the left as possible; player E is attempting to push the object as far to the right as possible. Each player is constrained differently, thus requiring different strategies. This means that a single strategy cannot be learned through self-play and given to both players. For P, the magnitude of the force is fixed and P must determine the appropriate angle with which to apply the force. For E, the angle is fixed and E must determine the appropriate magnitude of force to apply.

This simple game of force is derived from the *dolichobrachistochrone* game—one of

the classic games studied in differential game theory [177, 205]. [5] In this game, play takes place in the quadrant of a plane where $x \geq 0$ and $y \geq 0$. The objective is for P to drive an object into the y axis. E attempts to keep P from succeeding. Thus this game can also be posed as a pursuit game in which P is attempting to force E into a barrier.

Our game of force is much simpler than the *dolichobrachistochrone* game in that the optimal solution for each player is constant (but different). Thus we use this game as a "proof-of-concept" to demonstrate that *MBCL* is capable of learning strategies for each player. Recall that the dynamics of the game are given by

$$\begin{aligned} \dot{x} &= Av + B\sin u \\ \dot{y} &= -1 + B\cos u \\ |v| &\leq 1 \\ u &\in [0, 2\pi) \end{aligned}$$

where A and B are parameters defining the dynamics of the game, and u and v are the controls set by P and E respectively. The payoff function is defined to be the x value at the point the object lands. For this game, with $A = B = 1$, we expect optimal solutions when $u = \frac{3\pi}{2}$ and $v = 1$.

Prior to running these experiments, we also considered a one-step version of the game in which we applied an immediate payoff. We evaluated 50 possible moves for each player and constructed a 50×50 payoff matrix. Each entry in the matrix was determined

[5] The word *dolichobrachistochrone* is derived from the Greek *dolichos* meaning long, *brachys* meaning short, and *chronos* meaning time. It attempts to capture the opposing objective in time of lengthening and shortening distance to the terminal state.

as the change in x position following application of the corresponding two moves. The resulting payoff matrix was then used in the tableau to solve a linear program. The results of applying linear programming to the this payoff matrix yielded results consistent with the optimal solution.

For these experiments, several parameters needed to be set. In particular, we ran each simulation for 10,000 games and tested the results of learning after every 250 games. For each test, we played 50 games and averaged the "payoff" received after each game. For each of the 50 test cases, we used a uniform probability distribution and randomly generated a new starting position such that $x_0 \in [-0.25, 0.25]$ and $y_0 \in [0.85, 1.0]$. For the game parameters themselves, we set $A = 1$ and $B = 1$. For the learning algorithm, we fixed the learning rate at $\alpha = 0.15$ and set $\gamma = 0.95$, $k_s = 30$, and $k_m = 5$. We also defined the kernel, $K_w = 4$.

The results of this experiment are shown in Figure 6.3 with a comparison to optimal play shown in Figure 6.4. These graphs were produced by running the experiment 10 times and averaging the results at 250 game intervals. The track at the center of Figure 6.3 marked with an 'x' shows optimal play throughout the experiment. The track marked with diamonds indicates performance of both P and E as they learn. In addition, through the course of learning, P and E played against an optimal opponent to demonstrate their personal progress. The track marked with a square indicates P's performance against an optimal E, and the track marked with a plus indicates E's performance against an optimal P.

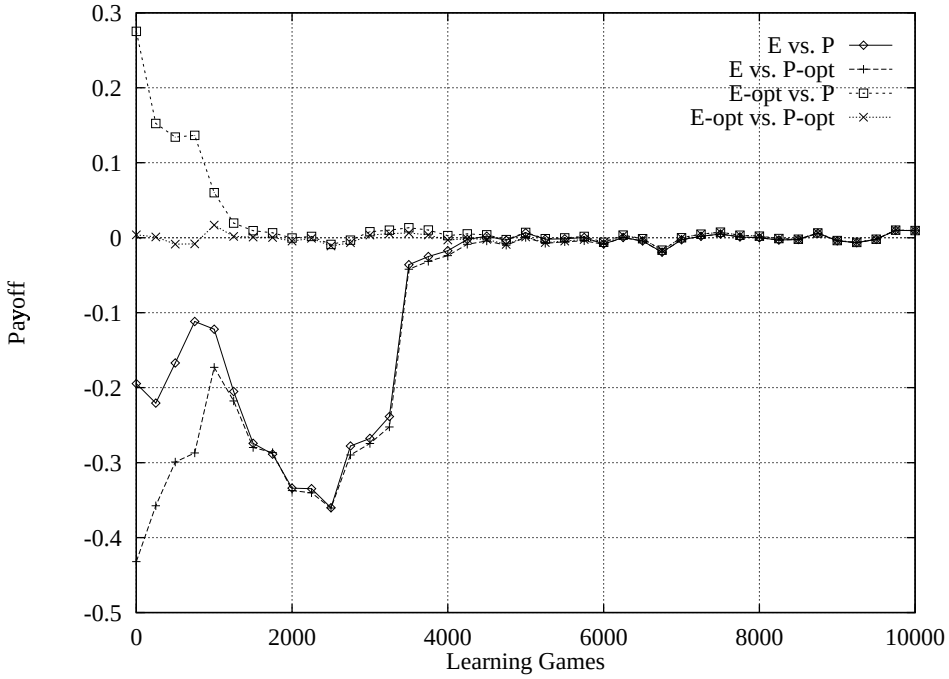


Figure 6.3: Learning performance for game of force.

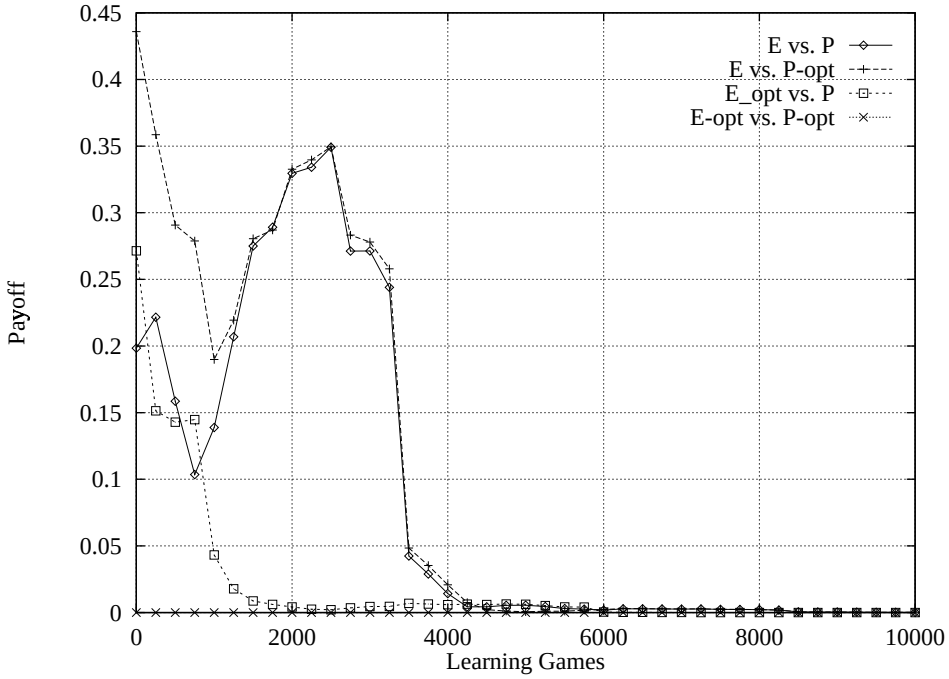


Figure 6.4: Deviation from optimal for game of force.

It is interesting to note that P learns to play optimally relatively quickly, achieving near-optimal performance after only 1250 games. P seems to converge to optimal performance after 2000 games, and E is still struggling. Finally, it seems E gains sufficient experience, playing against what is essentially a fixed opponent, and learns to play optimally after 4000 games.

One possible explanation for this difference in performance may arise by examining the landscape of the payoff function for each player. P is permitted to select any angle in the range $[0, 2\pi)$. The optimal move comes at $u = \frac{3\pi}{2}$, and sampling the action space provides a smooth slope on either side of optimal. For E, on the other hand, the optimal move ($v = 1$), arises at the boundary of legal moves ($v \in [0, 1]$). Sampling only has benefit on one side of optimal side (because the other side is infeasible). When compared to P which is able to sample on both sides, it is possible that E obtains half the benefit of exploration that P receives.

6.3.2 Pursuit with Simple Motion

In Section 2.8, we described a differential game in which one player is pursuing another player in a two-dimensional playing field. For player P, the objective is to capture E within some time limit (unknown to either player). For player E, the objective is to evade P for that period of time. Neither player has any constraints on its mobility, meaning each player can turn instantaneously in any direction. Each player moves at a fixed speed, and P is twice as fast as E.

Once again, we see that the two players cannot be modeled as one to allow "self-play" learning. In other words, each player has different objectives and capabilities and must learn appropriate strategies on their own. Thus they are heterogeneous agents [329]. Further, this game is more difficult than the game of force in that a separate action must be taken depending on the position of the opponent—no single fixed action applies.

Recall that the dynamics of the game are given by

$$\dot{x}_P = -v_P \sin \phi_P$$

$$\dot{y}_P = v_P \cos \phi_P$$

$$\dot{x}_E = -v_E \sin \phi_E$$

$$\dot{y}_E = v_E \cos \phi_E$$

where $\langle x_P, y_P \rangle$ and $\langle x_E, y_E \rangle$ are instantaneous positions of P and E respectively. Also, assume that P has a lethal envelope, $l > 0$, such that P captures E if

$$\sqrt{(x_P - x_E)^2 + (y_P - y_E)^2} \leq l$$

We specify $l = 0.05$ for these experiments. As described earlier, optimal solutions for the game exist for both players at

$$\phi_P = \phi_E = \arctan \frac{x}{y}$$

in coordinates relative to P. Payoff was defined to be the change in distance between P and E from the start of the game to the end of the game.

Once again, several parameters needed to be set for the experiments. We ran each simulation for 5000 games and tests the results of learning after every 250 games. For each test, we played 50 games and averaged the payoff received after each game. For each of the 50 games, we used a uniform probability distribution and randomly generated a new starting position such that $x_P \in [-1, 1]$, $y_P \in [-1, 1]$, $x_E \in [-1, 1]$, and $y_E \in [-1, 1]$.

For these and all subsequent experiments, we used a variable learning rate. Specifically, a learning rate was associated with each example stored in the memory base. Initially, the learning rate was set to 1.0 meaning that the first update of the associated Q value results in the actual payoff being assigned. Each time an instance is updated, the learning rate is changed according to the following schedule:

$$\alpha_i = \frac{1}{\chi_i}$$

where χ_i is a count of the number of times instance i has been updated, including the current time. Thus, initially, $\chi_i = 1$. In addition to the learning rates, we preset $\gamma = 0.95$, $k_s = 30$, $k_m = 5$, and $K_w = 4$.

The results of this experiment are shown in Figure 6.5 with a comparison to optimal play shown in Figure 6.6. These graphs were also produced by running the experiment ten times and averaging the results at 250 game intervals. As before, the track marked with an 'x' indicates optimal play by both players, the track marked with a diamond indicates both players using their currently learned strategies, the track with a square indicates P's performance against an optimal E, and the track marked with a plus indicates E's performance against an optimal P.

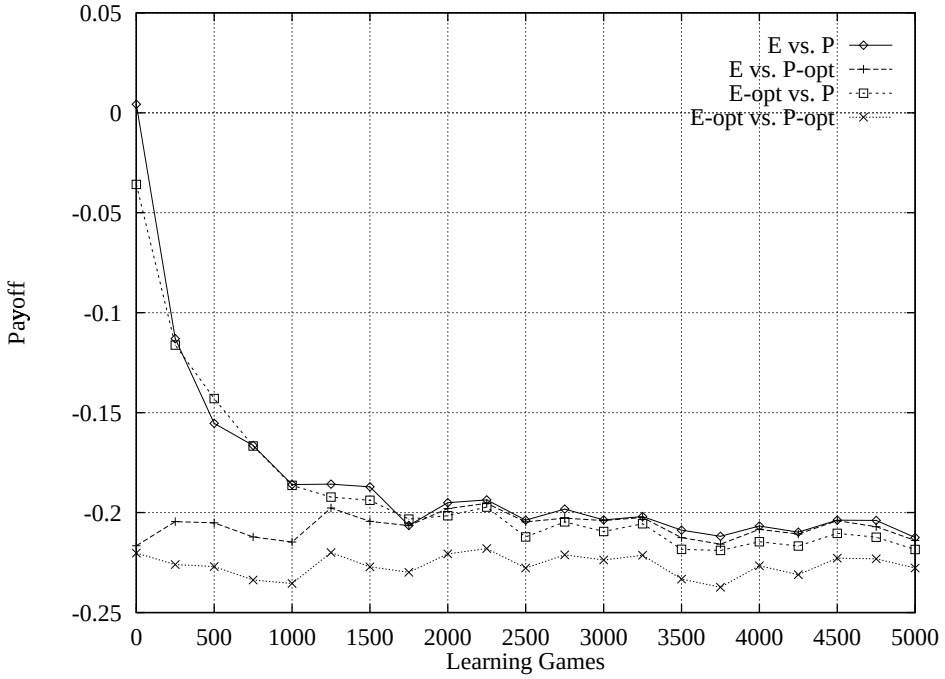


Figure 6.5: Learning performance for pursuit game with simple motion.

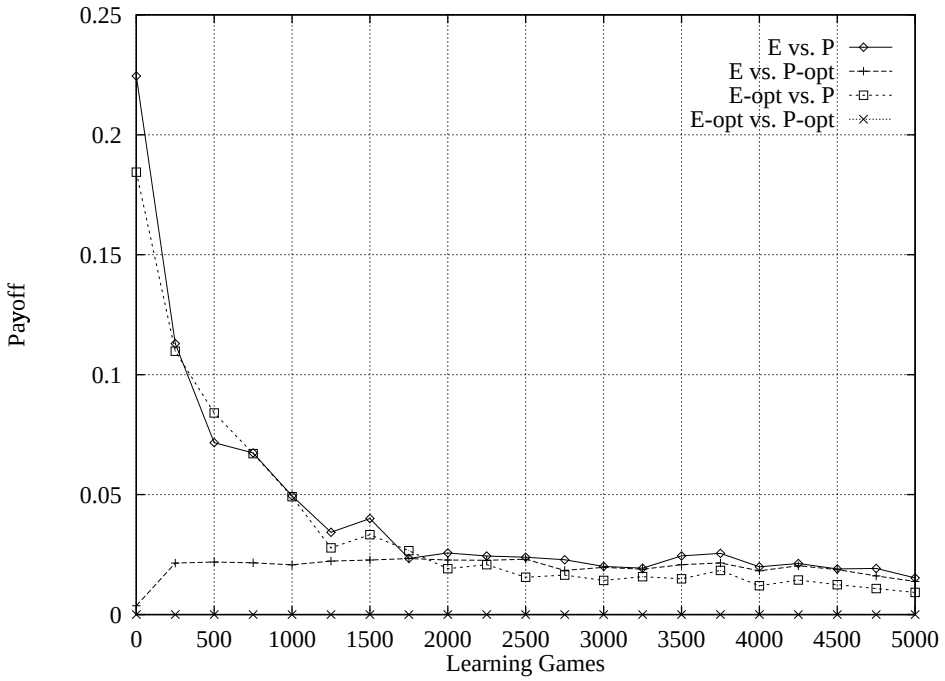


Figure 6.6: Deviation from optimal for pursuit game with simple motion.

Examining these figures, we note that E is able to perform well throughout. It appears as if no learning is required for E to maximize its ability to evade. We know, in general, this is not true. If E does not proceed directly away from P and P aims directly at E, then P must capture E more quickly.

We believe this surprise performance arises from an interesting artifact of the simulation. The simulation quantizes the game (i.e., plays the game in discrete steps as an approximation of the differential game); therefore, it appears E's performance can be attributed to a time-lag (similar to reaction time) in P's ability to aim at E. Because P must aim at the current position of E, a random move by E could have P aiming at the wrong position in the current time step.

When examining P's performance, we see that random motion is clearly not preferred for pursuit. In fact, initially, E is always able to get away from P. However, after only 1000 games, P has been able to direct its movements at E and brings its performance to a level comparable to optimal.

One additional curious result arises from examining these figures. If the simulation truly implements an optimal strategy for comparison, we would expect the optimal track to lie between the two tracks when each player is being tested against optimal players. In other words, when P and E both play optimally, the resulting performance should *always* be equal to or better than when one of the players is using the learned strategy. In Figure, 6.5, however, we find that this is not the case. Specifically, E's learned strategy appears to always beat the case where E plays optimally. Further, we find that all three tracks

(excluding the case where both players play optimally) converge to approximately the same performance after about 1750 games. We claim this performance is also a result of the artifact described earlier, but note that the players were still able to learn the appropriate "optimal" strategies for this revised game.

To characterize this artifact, we ran two simple experiments in which we varied the step size (i.e., Δt) in the kinematic equations of the simulation. In the first experiment, we considered average initial performance over 25 trials for step sizes of 0.05, 0.1, 0.25, and 0.5. Because each step size specifies the "distance" traveled in a single time step, cutting the step size in half necessitated doubling the number of steps in a simulation to make the comparison fair. The results of this experiment are shown in Figure 6.7.

In the second experiment, we considered average initial performance over 25 trials for step sizes of 0.001, 0.005, 0.01, 0.05, 0.15, 0.25, and 0.5. In this experiment, we kept the number of steps the same throughout all experiments. This was necessitated by the exponential growth in simulation time from the previous experiment. The results of this experiment are shown in Figure 6.8.

From these two figures, we see that increasing the resolution of the game (i.e., decreasing the step size) results in a closer approximation of the actual dynamics of the game. This is not surprising. In fact, we see that as the game becomes coarser, the "optimal" performance no longer appears optimal. This trend is demonstrated in both variations of the experiments (i.e., Figure 6.7 and Figure 6.8).

Returning to the results of the learning experiment, we claim learning resulted in

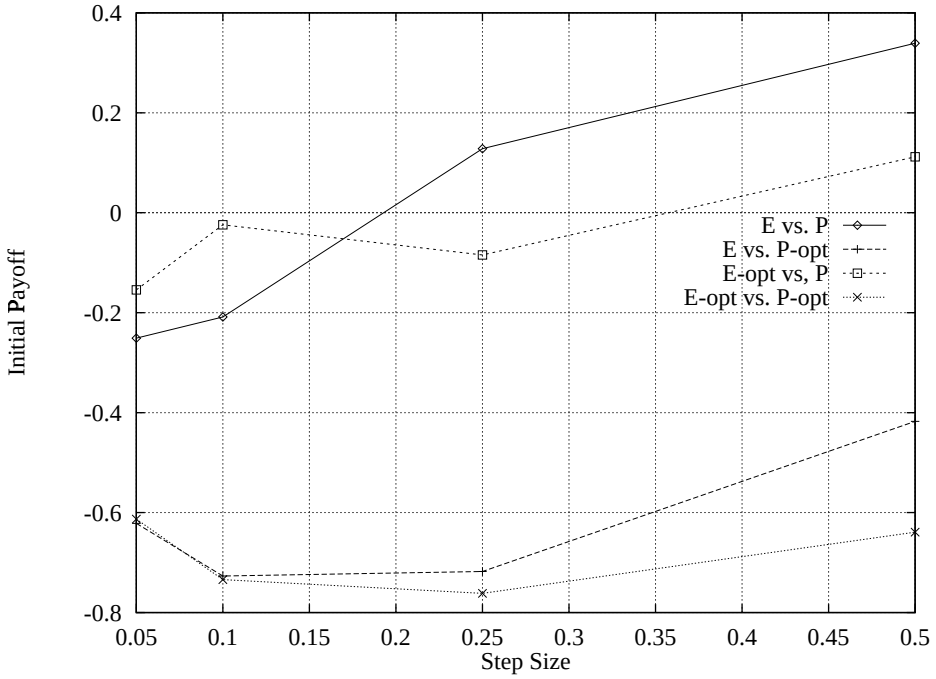


Figure 6.7: Initial learning performance with variable game length.

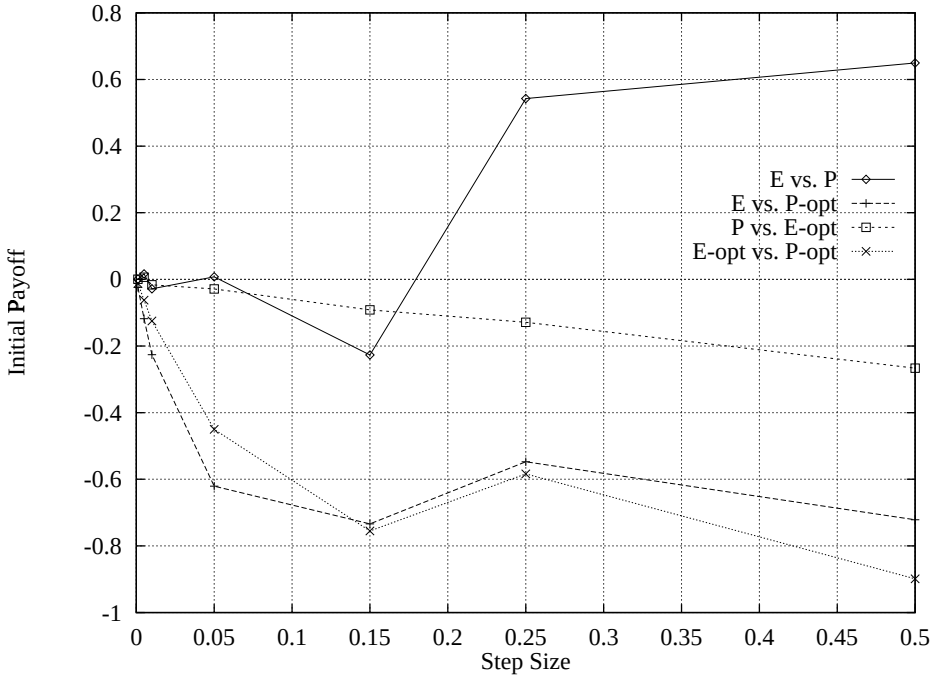


Figure 6.8: Initial learning performance with variable game length.

convergence that, although sub-optimal for the original differential game, may be optimal for our discrete implementation of the game. We also note from Figure 6.6 that E did learn a strategy other than random play as indicated by its relative improvement after 3500 games. For the first 3500 games, E's payoff deviated by approximately 0.02 from optimal. After 3500 games, the deviation dropped to approximately 0.01 with a trend toward continuing this reduction over the last 1000 games of the experiment.

6.3.3 Pursuit with Simple Motion in a Half Plane

At the end of Section 2.8, we described a variation of the pursuit game with simple motion in which a boundary exists at $x = 0$ thus forcing the game to take place in the half plane. On the surface, this game appears to be, essentially, the same as the previous game. In fact, this is true when the players are not near the boundary. However, repeating a figure from Chapter 2, we see that the boundary introduces a sharp change in the optimal strategy for E (Figure 6.9).

In the previous section, we noted that P and E are heterogeneous players, but we also saw that the optimal strategies were essentially the same. This game further extends the difficulty in learning optimal strategies in three ways. First, the strategies for both players still depend on the position of the other player, but there is a new factor affecting the strategies—the boundary. Second, because of this boundary, when play occurs in close proximity to the boundary, the strategies for the two players become different. Third, the transition between the two strategies for E is not smooth, indicating a discontinuity in the

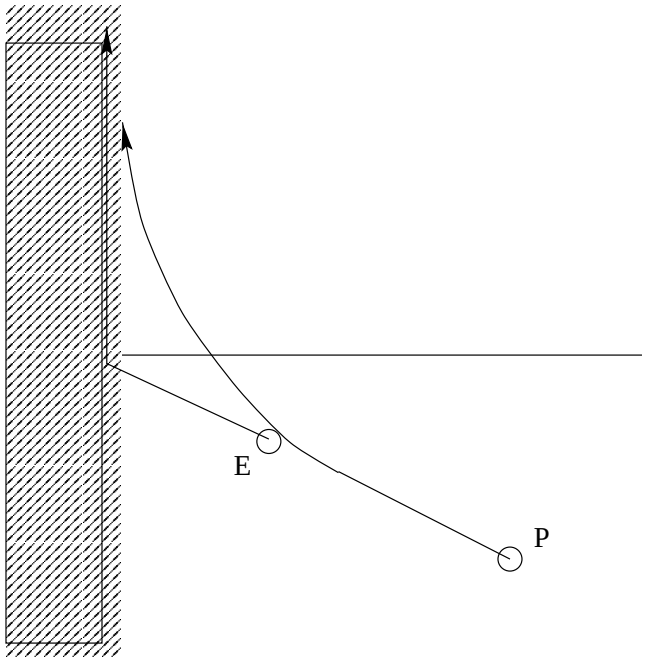


Figure 6.9: Optimal strategy for simple pursuit in the half plane.

optimal strategy for E arising from the boundary.

The dynamics of this game are identical to the previous game. Further, both players continue to be able to move with no limitation on mobility (except for that arising from the boundary) and with fixed speeds. All of the parameters from the previous game were used for this game as well. If either E or P collide with the boundary during play, they do not pass through the boundary but skid along the boundary a distance proportional to the y component of their force vector.

The results of this experiment are shown in Figure 6.10 with a comparison to optimal play shown in Figure 6.11. These graphs were produced by running the experiment ten times and averaging the results at 250 game intervals. Once again, the track marked with an 'x' indicates optimal play by both players, the track marked with a diamond indicates both players using their currently learned strategies, the track with a square indicates

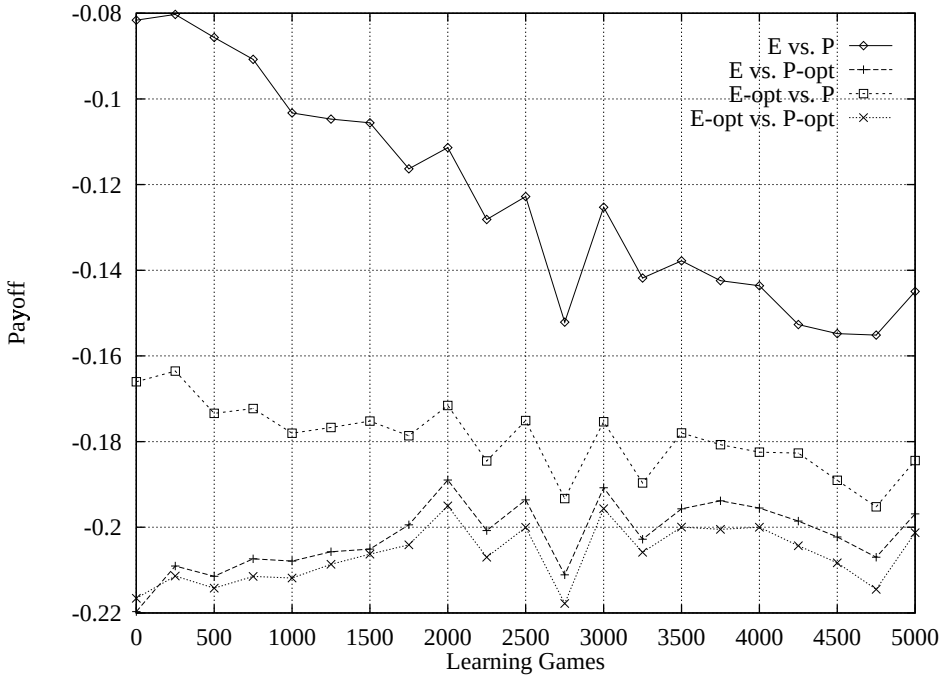


Figure 6.10: Learning performance for pursuit game in half plane.

P's performance against an optimal E, and the track marked with a plus indicates E's performance against an optimal P.

The first observation we make from these experiments is that the same artifact observed in the previous experiments appears to be present here as well. This is not surprising given we used the same simulator, adapted to include the boundary. It is also interesting to note that the average performance for optimal play is slightly lower (i.e., more in favor of P) than in the previous experiments. This can be explained through the presence of the boundary. P's optimal strategy has not changed, but E's has. Further, the boundary forces E to move in such a way that would be suboptimal given the boundary did not exist.

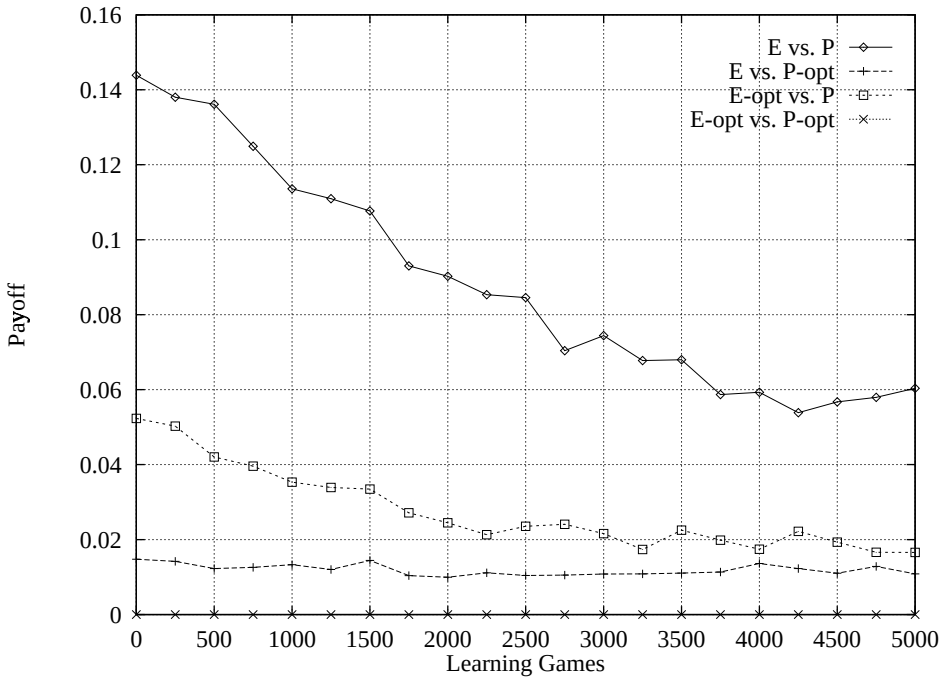


Figure 6.11: Deviation from optimal for pursuit game in half plane.

Therefore, it is reasonable to expect the boundary to favor P.

As with the previous experiment, we note that E appears to perform well using the strategy implicit when the memory base was seeded (i.e., random motion). This time, however, there appears to be no convergence toward optimal. As we saw in the last section, when the boundary was absent, some convergence did occur. It is possible that the boundary can serve as an advantage to E under the condition P's reaction to E's action is delayed.

We also note that P's performance does improve relative to E, but the level and rate of improvement has degraded. This would arise from the difficulties P would have when colliding with the barrier, thus slowing its advance towards E. Even though P's optimal action need not consider the boundary, the additional state variables in the examples increase the search space, making it more difficult for P to learn this fact. Indeed, the additional state information for P is "irrelevant," and irrelevant attributes are known to degrade performance in instance-based and memory-based learning [5, 293].

6.3.4 Pursuit with Limited Mobility

The final game we studied with *MBCL* further extends the pursuit game by limiting the mobility of both players. For this experiment, we remove the boundary, but we limit each player such that they can only make turns within a constrained range of possible turns. Specifically, we limited P's mobility such that it can turn only in the range $\pm\frac{\pi}{4}$, and we limited E's mobility such that it can turn only in the range $\pm\frac{\pi}{2}$. In other words, P can make instantaneous turns between $-45°$ and $+45°$ while E can make instantaneous turns

between $-90°$ and $+90°$.

This game is a generalization of the Homicidal Chauffeur game [39, 177, 205]. In the Homicidal Chauffeur, only the mobility of P is limited. Given the added complexity of the game, no optimal solution was available; however, we were able to define a heuristic based on the optimal solution for the Homicidal Chauffeur. Specifically, the heuristic strategy for P was to aim, as closely as possible, at E. If turning towards E required an angle exceeding P's limits, P turned as sharply toward E as possible. E's heuristic strategy, again based on the optimal strategy for the Homicidal Chauffeur was to turn sharply in the direction of P, attempting to get inside P's radius of curvature. This strategy was demonstrated in Chapter 4.

The dynamics of this game are identical to the original pursuit game with simple motion, except for the limitations on mobility. In addition, all of the experimental parameters are the same, except that we permitted learning to occur over 10,000 games rather than limiting to 5000 games.

The results of this experiment are shown in Figure 6.12 with a comparison to heuristic play shown in Figure 6.13. These graphs were produced by running the experiment ten times and averaging the results at 250 game intervals. This time, the track marked with an 'x' indicates heuristic play by both players, the track marked with a diamond indicates both players using their currently learned strategies, the track with a square indicates P's performance against a heuristic E, and the track marked with a plus indicates E's performance against a heuristic P.

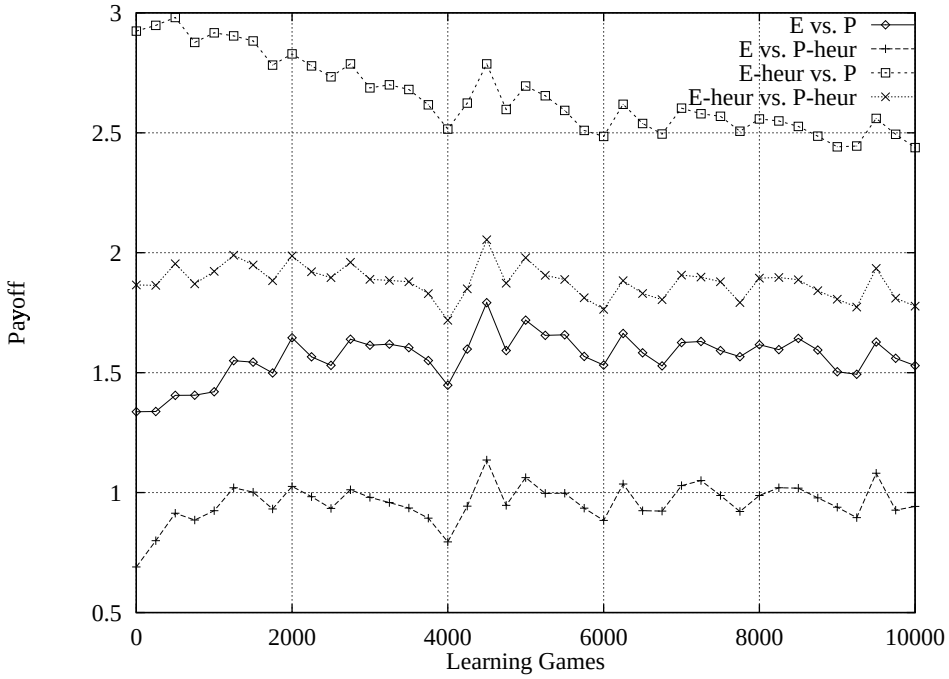


Figure 6.12: Learning performance for pursuit game with limited mobility.

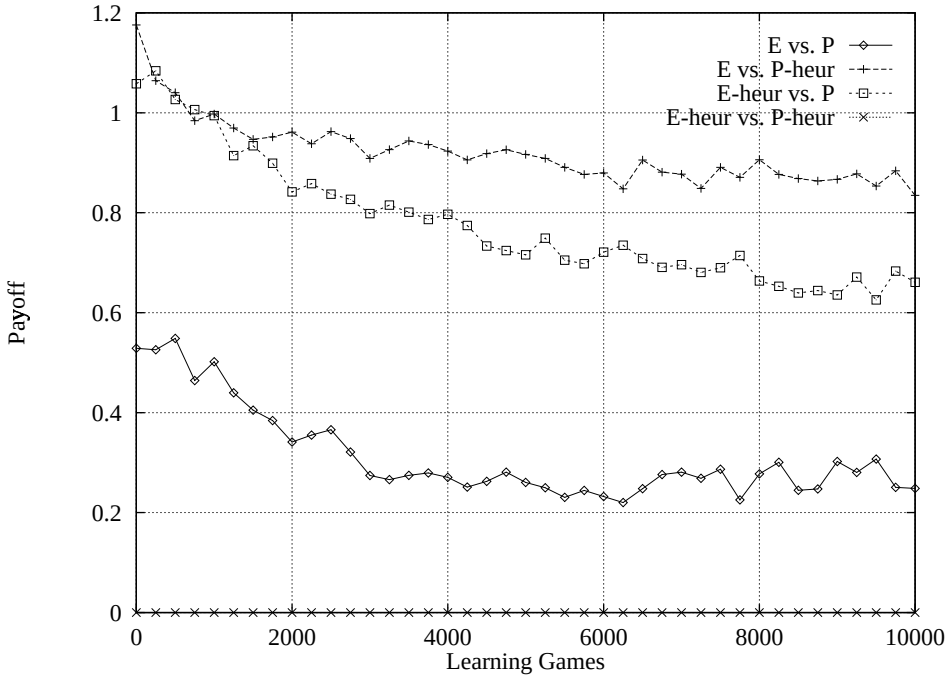


Figure 6.13: Deviation from heuristic for pursuit game with limited mobility.

Figure 6.12 appears to show little, if any, learning by the two players. However, if we examine Figure 6.13, we find that learning does, indeed, occur. In all cases, E was able to evade P, which is also consistent with the results in Chapter 4. When examining E's ability to play against the heuristic P, we find P losing a little ground at the start (demonstrating the power of random motions by E when P's mobility is limited) but losing about 30% more ground (relative to the heuristic) at the end of the experiment. Further, E was still improving when the experiment was terminated, albeit slowly.

When examining P's ability to play against the heuristic E, we find P losing considerable ground at the start but reducing its losses by approximately 40% (relative to the heuristic) by the end of the experiment. As with E, P was still improving when the experiment was terminated.

6.4 Discussion

The results of applying *MBCL* to solving differential games are quite encouraging. In all cases, learning was demonstrated, and in the simplest cases, near optimal performance was achieved. Unfortunately, the computational burdens for learning these games was quite extensive.

As mentioned in Section 6.2, the process *MBCL* follows requires seeding the memory base with several examples. The current version of *MBCL* does not modify the memory base at all, except to update the Q values associated with each example. All of the experiments were run on either Sun Sparc 2 or Sun Sparc 10 processors. To give an idea of

Table 6.1: Relative computational burdens for solving games with *MBCL*.

Game	Games Stored	Examples Stored	Minutes
Force	7500	50,000	96
Simple	2000	400,000	1376
Half	2000	400,000	1372
Limit	2800	560,000	3193

the computational burden, Table 6.1 shows the number of examples in each of the memory bases and the clock time required, in the best case, to complete the experiment.

Clearly, the time required to learn solutions to these games is excessive. Of course, our method of searching for neighbors did not use efficient structures to reduce search time, such as *kd*-trees. The simplest game, in which only a single action needed to be found, required an hour and a half (actually less, given it converged in less than half of this time). When we considered only the one-step game of force, we found the solution immediately. The other games required approximately a day to run, and convergence only occurred with one game.

This observation points out the advantage of a proper representation for the problem to be solved. When we posed the problem as a delayed reinforcement task, we found the task learnable, but only after a large amount of simulation. On the other hand, posing the problem as an immediate reinforcement learning problem yielded a solution in one step. Considering the pursuit games, we could have represented them as immediate reinforcement learning problems as well, using the change in distance between the players as the immediate payoff. But in preliminary experiments doing just that, we found no difference

in performance from using delayed reinforcement.

As can be seen, the number of examples required to learn the game of force was relatively small. The pursuit games required approximately an order of magnitude more examples and were not able to learn as well. This provides an indication of the difficulty of these tasks.

What is not shown in either the graphs or the table above is that the examples for the pursuit games were probably not chosen intelligently. For seeding the memory base, initial states for games were chosen at random according to a uniform probability distribution. One third of the games were played with random strategies for both players, one third with P playing a single, fixed random strategy, and one third with E playing a single, fixed random strategy.

This approach to seeding was chosen to provide a wide sample of state-action combinations. Initial Q values did not matter since they would be learned over time. Nevertheless, a uniform random sampling of the space was, apparently, not sufficient to approximate some of the surfaces encountered in these games. Applications of variable resolution techniques [16, 110, 240, 312] may be more appropriate for problems such as these.

In some ways, the results from *MBCL* are highly encouraging. They indicate co-learning can occur and suggest it is possible to learn optimal solutions to two-player differential games. Unfortunately, the computational resources required to learn these solutions are too excessive. In the next chapter, we explore an alternative strategy to co-learning

with the focus being on reducing the required computational resources while maintaining or improving learning performance. We find several encouraging results improving the computational requirements and also find that the two algorithms are comparable in their ability to learn to play differential games.

6.5 Summary

In this chapter, we provided a new algorithm for memory-based co-learning in which two opposing agents learn control strategies simultaneously. These results (and the results of the next chapter) can be extended to alternating Markov games (in which players take turns) [213], team games (in which teams of players cooperate to devise mutual strategies) [337, 338], and community games (in which players choose opponents to maximize their personal payoff) [325]. They can also be applied to more traditional games with homogenous agents such as backgammon [342], checkers [297, 298], and othello [323]. The strengths of the approach include the relative simplicity in storing examples and updating value estimates for game play. Unfortunately, the approach is both memory and computation intensive. The next chapter specifically addresses these concerns.

Chapter 7

Tree-Based Co-Learning

7.1 Coping with Limitations in Time and Memory

In the previous chapter we present an algorithm for co-learning in differential games that, although providing promising results, had large computational and memory requirements. In this chapter, we consider an alternative algorithm that, although not memory-based, is inspired by the results of applying *kd*-trees in memory-based learning [46, 110, 128, 129, 237].

A *kd*-tree is a data structure used to store a set of examples in a memory base such that nearest neighbors can be found in logarithmic expected time. Specifically, a *kd*-tree is a binary tree where each interior node of the tree tests the values of one of the attributes in the k-dimensional attribute space. In addition, each node corresponds to a single instance in the memory base [237]. Nodes are selected for splitting until no further splits are required (i.e., until all points are represented in the tree).

In memory-based learning, the *kd*-tree can provide significant speed-up in search-

ing for nearest neighbors; however, the size of the memory based does not change. Actually, the resulting memory base will be larger than a "monolithic" memory base because of the overhead associated with storing the tree.

To address the problem of storing all examples, alternatives such as editing have been offered to reduce the size of the memory base (see Section 5.4). However, striking a balance between sufficient coverage of the problem space and small size of the memory base is tricky at best. Methods for variable resolution memory-based learning are discussed in Section 3.2.2 as possible approaches to provide this balance.

Our approach applies the speed advantage of the *kd*-tree with the space advantage of variable resolution memory-based learning without the need to store explicit examples. Instead, we incrementally construct a "decision" tree that partitions the state space and strives to maintain balance to minimize search. Rather than storing examples at interior nodes of the tree, we store a game matrix at the leaves that represent behavioral strategies for playing the game. When performance converges, the game matrix can be discarded, and the mixed strategies associated with the game matrix retained. Further, if any of the pure strategies have an associated probability of zero, these can be dropped as well.

7.2 A Tree-Based Co-Learning Algorithm

To describe the tree-based algorithm (which we call *TBCL*), we begin by considering the degenerate case where the tree consists of a single node. In this case, the node covers the entire state space of the game. Associated with this node is a single game matrix, pairing

expected payoffs for the strategies of the two players. Mixed strategies are computed by solving the linear program defined by the game matrix. Learning consists of updating the entries in the game matrix based on actual play and resolving the linear program.

Because most games will require different actions in different states, a single node with a single game tree will not be adequate. When learning converges, if the performance is not adequate, the node can be split into two additional nodes and learning restarted. Splitting consists of selecting one of the state variables and dividing the state space along (in the simplest case) the midpoint of the dimension defined by that variable. The game tree of the parent is then copied to each of the children, the learning rates reset, and learning proceeds as before. A high-level description of the learning algorithm is shown in Figure 7.1.

The first step in the algorithm is to create the tree. This consists of creating a single node, covering the entire state space. A single game matrix is constructed with uniformly distributed random values. Then the corresponding linear program is solved to provide an initial set of mixed strategies for the two players to follow. This initial set of strategies is tested against 50 uniformly distributed random games, and the result is compared to a performance goal (either in terms of convergence or number of iterations).

If the performance goal is not met, *TBCL* passes into a two-part learning loop. The first part consists of performing Q-learning on the current game structure. The second part consists of selecting a node in the tree to split should the performance goal not be obtained.

```
algorithm TBCL;
    tree = create-tree;
    initialize-game-matrix(tree);
    update-strategies(tree);
    performance = test(tree);
    do-until (performance ≥ goal)
        do-until converged
            game = play-game(tree);
            update-Q-values(tree,game);
            update-strategies(tree);
        enddo;
        performance = test(tree);
        if (performance < goal) then
            node = select-leaf(tree);
            split-node(node);
            update-strategies(node→left-child);
            update-strategies(node→right-child);
        endif;
    enddo;
end;
```

Figure 7.1: High-level pseudocode of tree-based co-learning algorithm.

In the Q-learning portion of the algorithm, a game is played and evaluated. In *MBCL*, the history of the game was stored to permit evaluation of the Q values associated with the instances in the memory base. (Alternatively, the game itself could have provided additional examples to be added to the memory base, but we did not use this alternative). In *TBCL*, the sequence of the game is traversed to determine which node in the tree was used at each step and what actions were taken at that step. The corresponding cell in the game matrix at that node is then updated using Q-learning. Once the game is finished, all game matrices that were changed are solved using linear programming to find new strategies for the associated nodes.

The Q-learning loop continues until some convergence criterion is satisfied. Again, this criterion could be a measure of the change in performance of the players, or it could be a fixed number of iterations. We chose the latter for our experiments. Once the loop finishes, performance is measured and compared against the performance goal. If the goal is satisfied, the algorithm terminates. Otherwise, a node is selected and split. Nodes are selected by considering the number of updates. The node receiving the most updates in a Q-learning loop is split because this indicates a large number of visits to the states represented by that node. Splitting takes place according to the algorithm given in Figure 7.2. Once the node has been split and new game matrices generated for the children, these matrices are also solved, and Q-learning continues.

If we consider the algorithm in Figure 7.2, we see that the attribute is selected that maximizes the difference in the game matrices following the split. Specifically, each

```
algorithm split-node(node);
    for-every attrib do
        split = 1/2 node.attrib.low + node.attrib.hi;
        for-every P-strat do
            for-every E-strat do
                init-game(attrib,split,LEFT);
                matrix1[P-strat][E-strat] = play-from(P-strat,E-strat);
                init-game(attrib,split,RIGHT);
                matrix2[P-strat][E-strat] = play-from(P-strat,E-strat);
            enddo;
        enddo;
        diff[attrib] = distance(matrix1,matrix2);
    enddo;
    split-attrib = argmax(diff);
    split-along(node,split-attrib);
end;
```

Figure 7.2: Pseudocode for splitting algorithm.

attribute is considered by assuming the split is made along the attribute. Two game matrices are generated for each split. A game matrix is constructed by initializing a game from the midpoint of the partition and playing a game. The game is played according to the strategies stored in the tree, except for the first move. All pair-wise combinations of moves are considered for this first move, and the results of the game are stored in the matrix cell indicated by the initial pair of moves.

Each pair of matrices is compared by computing the Euclidean distance between the matrices. The attribute whose pair of matrices is maximally distance is selected for splitting. When the node is split, the limits for that attribute are updated within the node, and the game matrix from the parent node is copied into each child. All of the counts used

to update the learning rate for Q-learning are reset to zero to restart the learning process for that partition.

When playing the game, the moves for both players are determined by traversing the tree to find the partition that covers the current state. When the partition is found, the mixed strategies associated with each player are used directly to pick their respective moves. Because the strategies are updated at the end of each learning game, there is no need to solve the linear programs online as in the memory-based approach. The result is fast search through the state space and fast determination of the actions.

As an example, a decision tree for the pursuit game with simple motion is shown in Figure 7.3. The mixed strategies stored at each of the tree's leaves is shown in Table 7.1. Note the splits occur along a single attribute, thus the corresponding tree is analogous to the axis-parallel decision trees generated by approaches such as ID3 and C4.5 [270, 271, 272]. When limited to two dimensions, axis-parallel trees can be shown as a partitioning of the attribute space. We provide a similar representation for the game decision tree in Figure 7.4.

7.3 Experiments

We evaluated the performance of *TBCL* using the same games and same procedures as for the evaluation of *MBCL*. As before, we applied *TBCL* to four games, including the simple game of force, the pursuit game with simple motion, the pursuit game in a half plane, and the pursuit game with limited mobility. Once again, we assume all states are represented

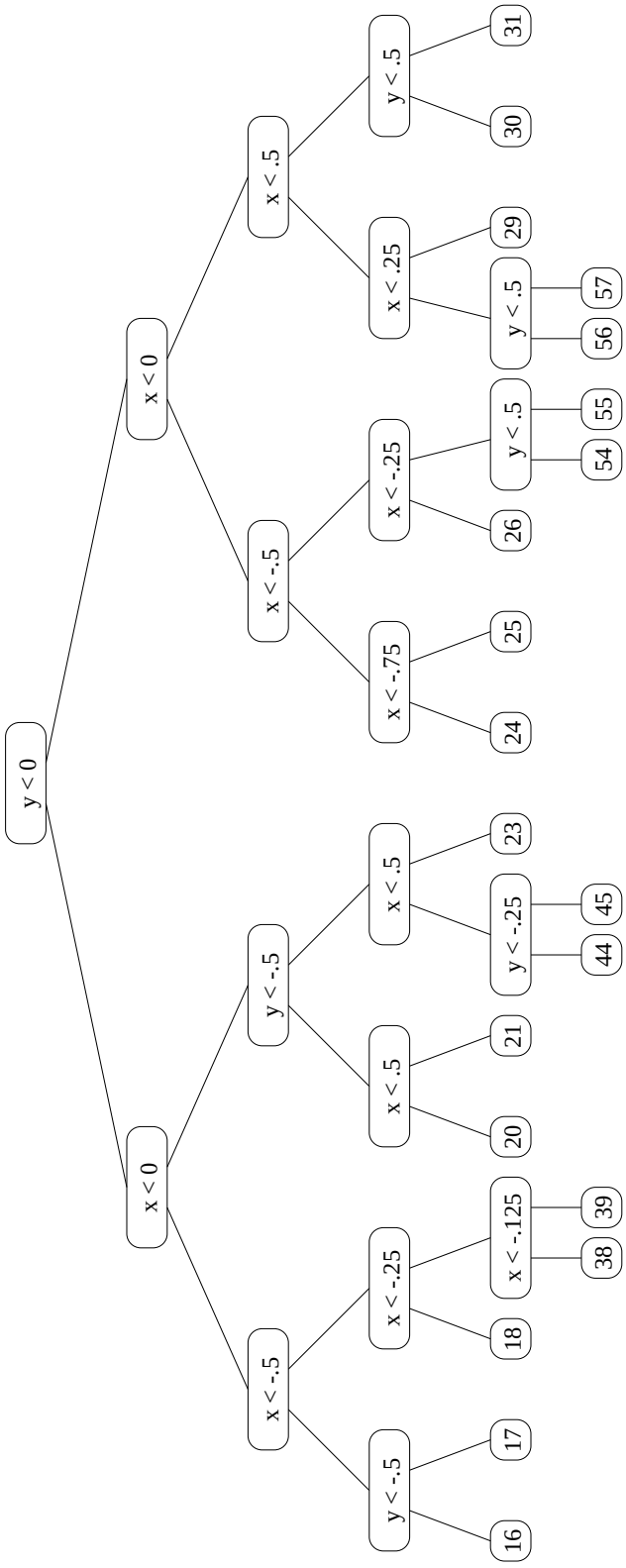


Figure 7.3: Sample tree derived for pursuit game of simple motion.

Table 7.1: Mixed strategies at leaves of learned tree.

Node	Player	$\frac{-9\pi}{10}$	$\frac{-7\pi}{10}$	$\frac{-5\pi}{10}$	$\frac{-3\pi}{10}$	$\frac{-\pi}{10}$	$\frac{\pi}{10}$	$\frac{3\pi}{10}$	$\frac{5\pi}{10}$	$\frac{7\pi}{10}$	$\frac{9\pi}{10}$
16	E	0.000	0.514	0.486	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	P	0.000	0.364	0.636	0.000	0.000	0.000	0.000	0.000	0.000	0.000
17	E	0.000	0.000	0.640	0.000	0.360	0.000	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.931	0.069	0.000	0.000	0.000	0.000	0.000	0.000
18	E	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	P	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
38	E	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	P	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
39	E	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	P	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
20	E	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
	P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
21	E	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.295	0.705
	P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.429	0.571
44	E	0.000	0.796	0.000	0.000	0.000	0.000	0.000	0.204	0.000	0.000
	P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.669	0.331
45	E	0.000	0.078	0.742	0.000	0.084	0.016	0.000	0.000	0.080	0.000
	P	0.073	0.078	0.000	0.000	0.000	0.000	0.000	0.133	0.716	0.000
23	E	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
	P	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
24	E	0.000	0.000	0.553	0.447	0.000	0.000	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
25	E	0.000	0.000	0.342	0.480	0.000	0.178	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.550	0.268	0.182	0.000	0.000	0.000	0.000	0.000
26	E	0.000	0.000	0.000	0.000	0.358	0.642	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
54	E	0.000	0.283	0.000	0.000	0.000	0.717	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.041	0.959	0.000	0.000	0.000	0.000	0.000	0.000
55	E	0.000	0.000	0.000	0.000	0.053	0.947	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.000	0.000	0.355	0.645	0.000	0.000	0.000	0.000
56	E	0.000	0.000	0.000	0.000	0.000	0.698	0.000	0.000	0.000	0.302
	P	0.000	0.000	0.000	0.000	0.000	0.512	0.488	0.000	0.000	0.000
57	E	0.000	0.375	0.000	0.000	0.000	0.625	0.000	0.000	0.000	0.000
	P	0.000	0.000	0.000	0.000	0.000	0.906	0.094	0.000	0.000	0.000
29	E	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
	P	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
30	E	0.000	0.000	0.000	0.000	0.000	0.000	0.893	0.000	0.107	0.000
	P	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.993	0.000	0.000
31	E	0.000	0.000	0.000	0.000	0.000	0.156	0.884	0.000	0.000	0.000
	P	0.000	0.000	0.000	0.480	0.000	0.520	0.000	0.000	0.000	0.000

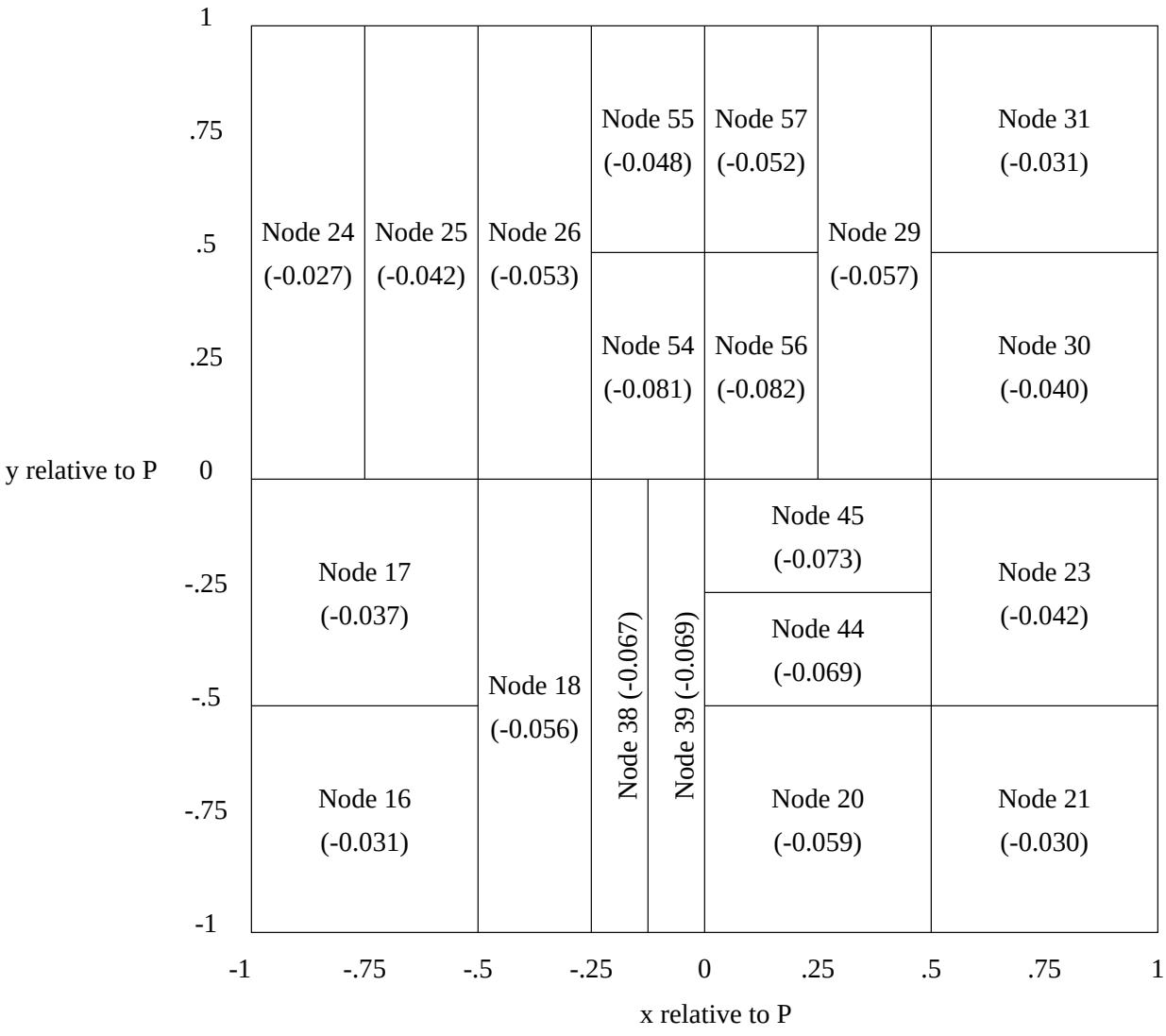


Figure 7.4: Sample partitioning derived for pursuit game of simple motion.

relative to P, and the game matrices are constructed with $n = 10$ for each player.

7.3.1 A Game of Force

For the simple game of force, we used the same kinematic equations as in Section 6.3.1. Because of the nature of the algorithm several different parameters were set. For this game, we trained for 100,000 games and only generated one node in the tree. We decided not to subdivide the space given the simplicity of the game. As with *MBCL*, this game served as a "proof-of-concept" for the algorithm. We tested the algorithm after ever 1000 games and averaged the "payoff" received after each game. For each of the 50 test cases, using a uniform probability distribution, we randomly generated a new starting position such that $x_0 \in [-0.25, 0.25]$ and $y_0 \in [0.85, 1.0]$. For the game parameters themselves, we once again set $A = 1$ and $B = 1$. For the learning algorithm, we allowed the learning rate to vary and set $\gamma = 0.95$.

The results of this experiment in shown in Figure 7.5 with a comparison to optimal play shown in Figure 7.6. As before, these graphs were generated by running the experiment ten times and averaging the results at 1000 game intervals. The track at the center of Figure 7.5 marked with an 'x' shows optimal play throughout the experiment. The track marked with diamonds indicated performance of both P and E as they learn. The track marked with a square indicates P's performance against an optimal E, and the track marked with a plus indicates E's performance against an optimal P.

In these experiments, we find convergence occurs relatively quickly with perfor-

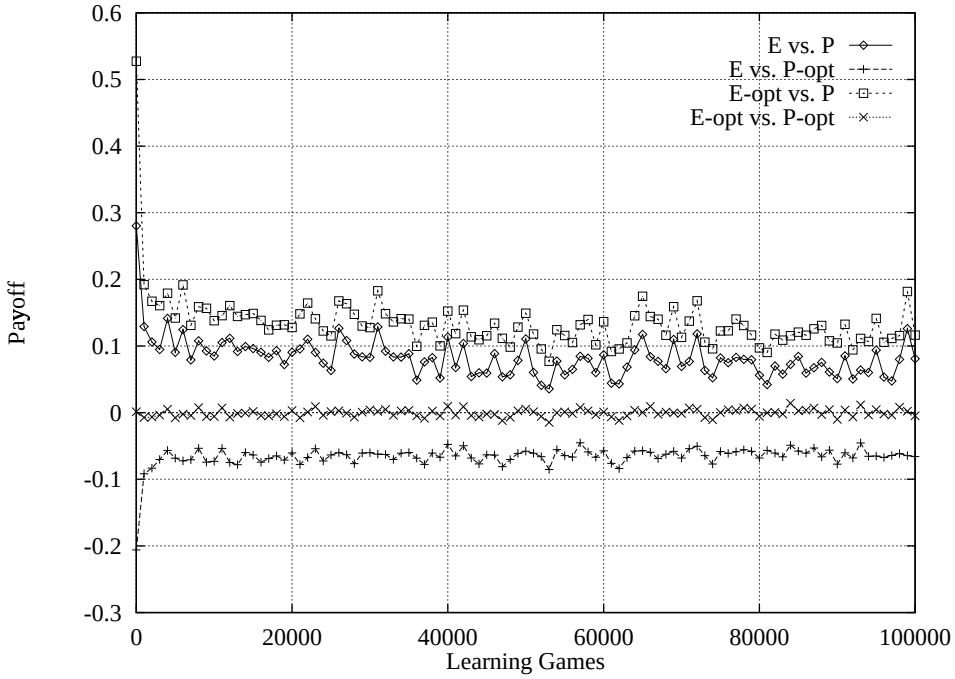


Figure 7.5: Learning performance for game of force.

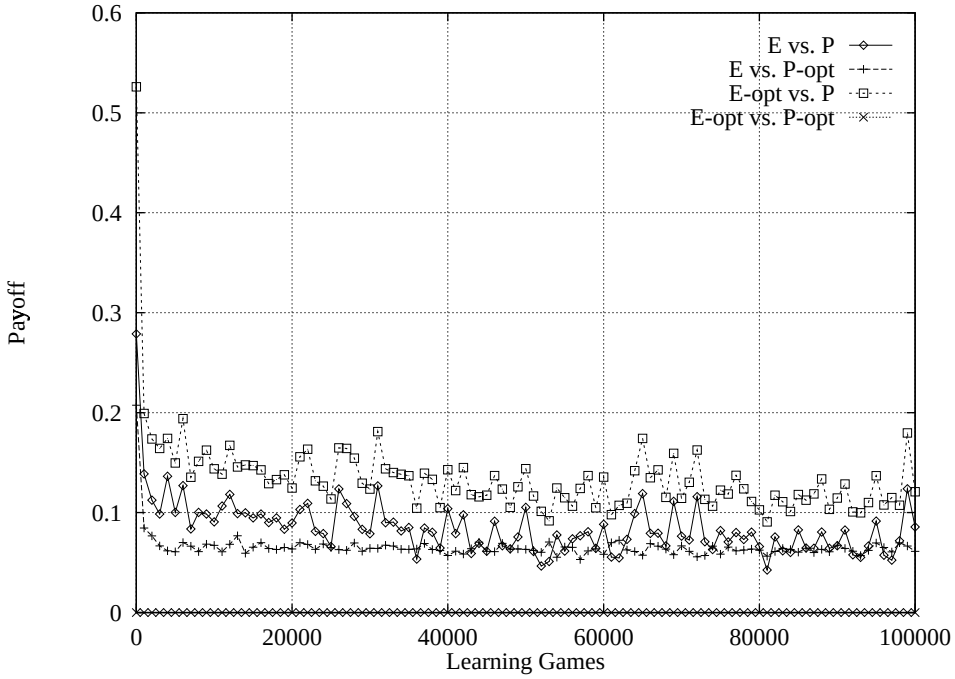


Figure 7.6: Deviation from optimal for game of force.

mance settling after approximately 20,000 games. Performance seems to improve some through 60,000 games, but then there is a small jump causing performance to degrade followed by a return to the previous level of performance. It is also interesting to note that when P and E play each other, performance is fairly constant throughout and is degraded from optimal in favor of E. Further, performance seems to converge to this level rather than to the optimal level. We suspect this is due to quantizing and interpolating between strategies; although, no experiments were run to verify this hypothesis. Another possibility is that performance converged based on the simulation artifact described in Section 6.3.2, but this is unlikely given the constant optimal strategies for this game.

7.3.2 Pursuit with Simple Motion

For the game of pursuit with simple motion, we again used the same kinematic equations as in Section 6.3.2. Once again, we trained for a period of 100,000 games, but this time we split a node in the tree after every 5000 games. This resulted in a tree with 20 leaf nodes. We tested the results of learning after every 1000 games to monitor the level of convergence while a tree's structure was fixed and to observe the effects of adding a new node to the tree. Whenever we tested the performance of the algorithm we played 50 games generated at random according to a uniform probability distribution and averaged the payoff received after each game. For each of the 50 games, we generated starting positions such that $x_P \in [-1, 1]$, $y_P \in [-1, 1]$, $x_E \in [-1, 1]$, and $y_E \in [-1, 1]$. Once again, we permitted the learning rate to vary and set $\gamma = 0.95$.

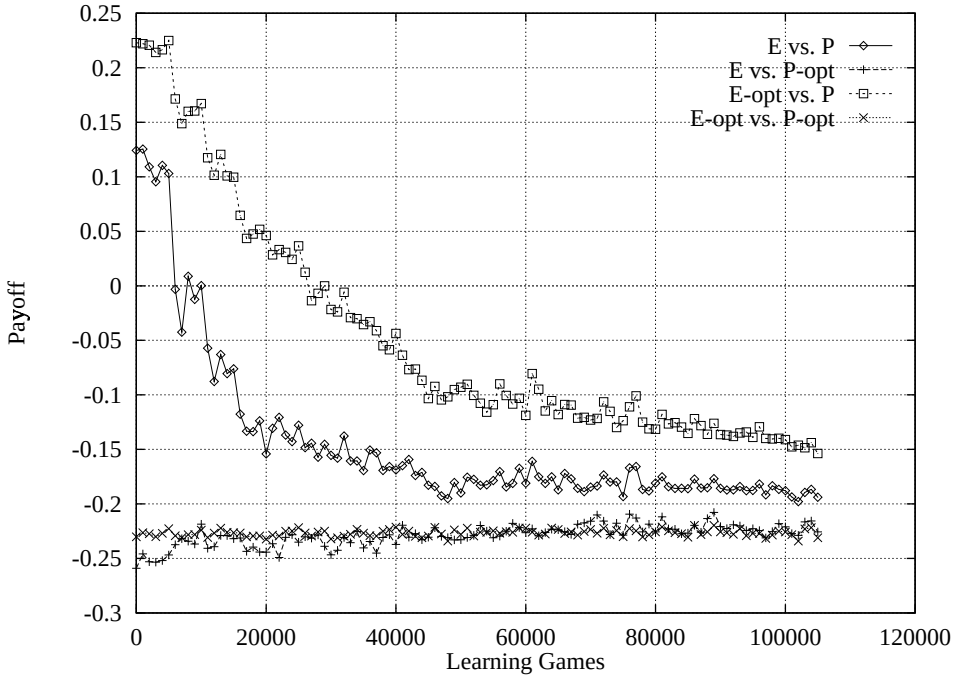


Figure 7.7: Learning performance for pursuit game with simple motion.

The results of this experiment are shown in Figure 7.7 with a comparison to optimal performance shown in Figure 7.8. These graphs were also produced by running the experiment ten times and averaging the results at 1000 game intervals. As before, the track marked with an 'x' indicates optimal play by both players, the track marked with a diamond indicates both players using their currently learned strategies, the track with a square indicates P's performance against an optimal E, and the track marked with a plus indicates E's performance against an optimal P.

Examining these figures, several interesting results can be observed. First, similar to *MBCL*, performance by E appears to be good without any learning, thus indicating the

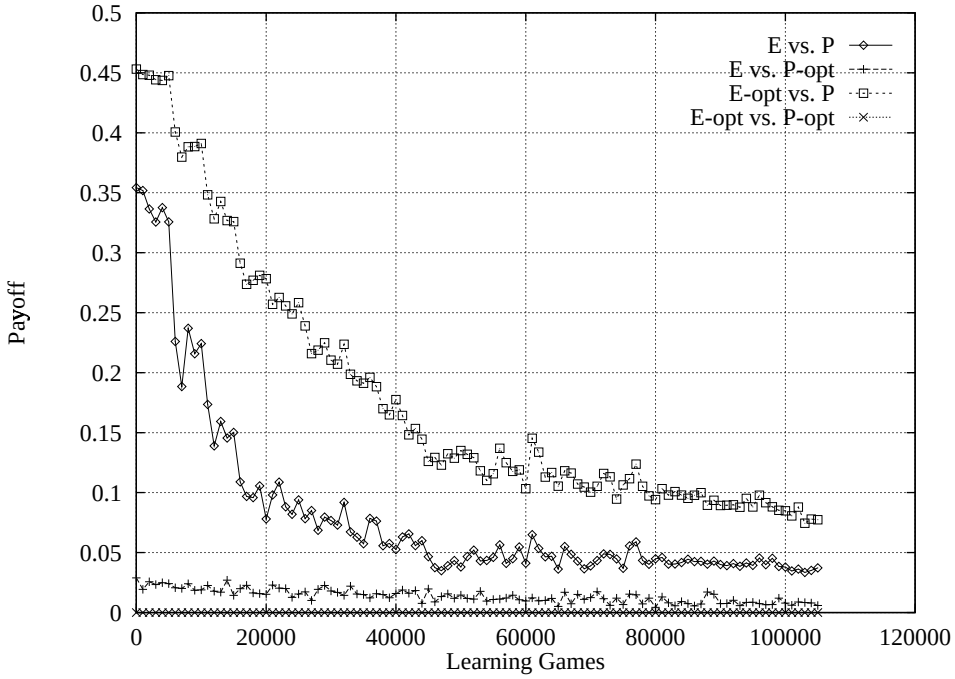


Figure 7.8: Deviation from optimal for pursuit game with simple motion.

power of random moves in our simulation. In fact, we find when E applies its initial strategy against an optimal P, it is able to do nearly as well as an optimal E. We do note some movement toward optimal through 40,000 games, however.

When examining P's performance, we find it starts out performing poorly, never capturing E (as shown by a positive payoff). However, after 5000 games, P has been able to improve to at least prevent E from gaining any additional ground. By the time 40,000 games have been played, P is able to advance on E fairly consistently. When the experiment was terminated at 100,000 games, the slope of P's learning curve indicated it was still improving.

Taking a closer look at these learning curves reveals an interesting, but not surprising, behavior of *TBCL*—especially when examining the performance of E playing against P rather than their optimal counterparts. Notice that performance is fairly steady through the first 5000 games. At 5000 games, the first split occurs and average payoff drops from 0.1 to about 0.0. This suggests a single node was not sufficient for improving the performance of either player. In fact, if we examining the performance of each player against the optimal counterpart, we find similar flat performance. Examining performance between 5000 and 10,000 games, we find a similar flat trend. When the tree splits again at 10,000 games, a similar change in payoff is experienced with average payoff dropping from 0.0 to about -0.075. We find yet another drop at 15,000 games. From approximately 20,000 games onward, we do not see any additional sudden changes but note a relatively steady improvement when examining P's performance. It is possible the performance change should still

be attributed to the tree splitting, but by this time the impact of splitting on the total tree is so small, it is difficult to discern the reason for improvement.

7.3.3 Pursuit with Simple Motion in a Half Plane

Continuing with the experiments, we next added the barrier at $x = 0$ to play the pursuit game in the half plane (see Section 6.3.3). All of the parameters used in this experiment were identical to the parameters in Section 7.3.2. The dynamics of the game are identical to the previous game, and each player still has the ability to turn instantaneously in any direction (except when constrained by the boundary). Once again, if either P or E collide with the boundary, they skid along the boundary a distance proportional to the y component of their force vector.

The results of this experiment are shown in Figure 7.9 with a comparison to optimal play shown in Figure 7.10. These graphs were produced by running the experiment ten times and averaging the results at 1000 game intervals. Once again, the track marked with an 'x' indicates optimal play by both players, the track marked with a diamond indicates both players using their currently learned strategies, the track with a square indicates P's performance against an optimal E, and the track marked with a plus indicates E's performance against an optimal P.

We find that the performance of each player is similar to performance without the barrier, except that the barrier apparently causes some difficulties that need adaptation. In particular, we note that E does not start performing "optimally" as before, but its

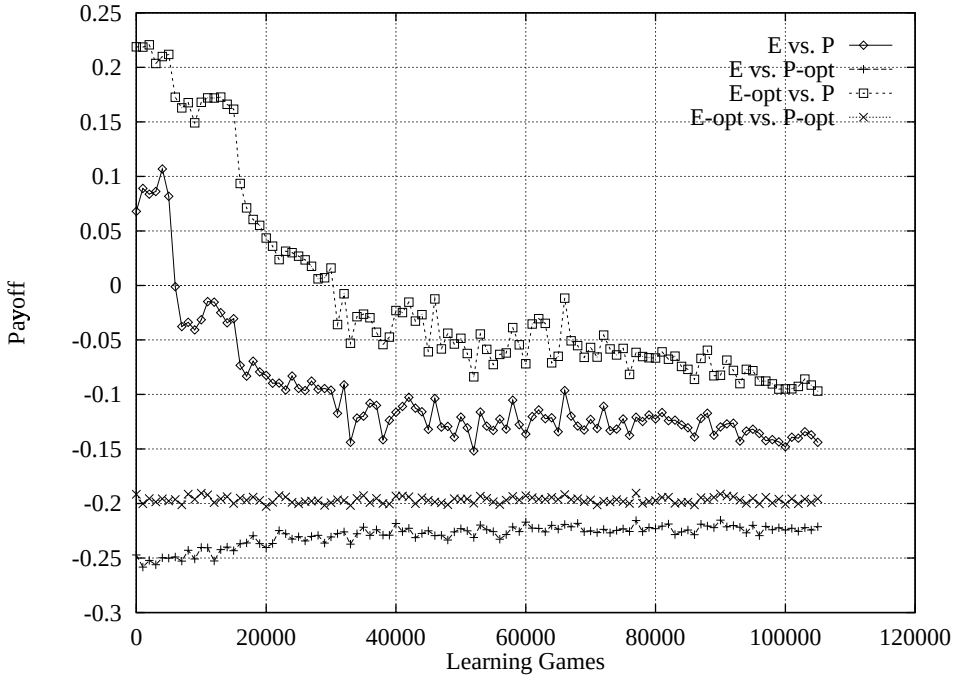


Figure 7.9: Learning performance for pursuit game in half plane.

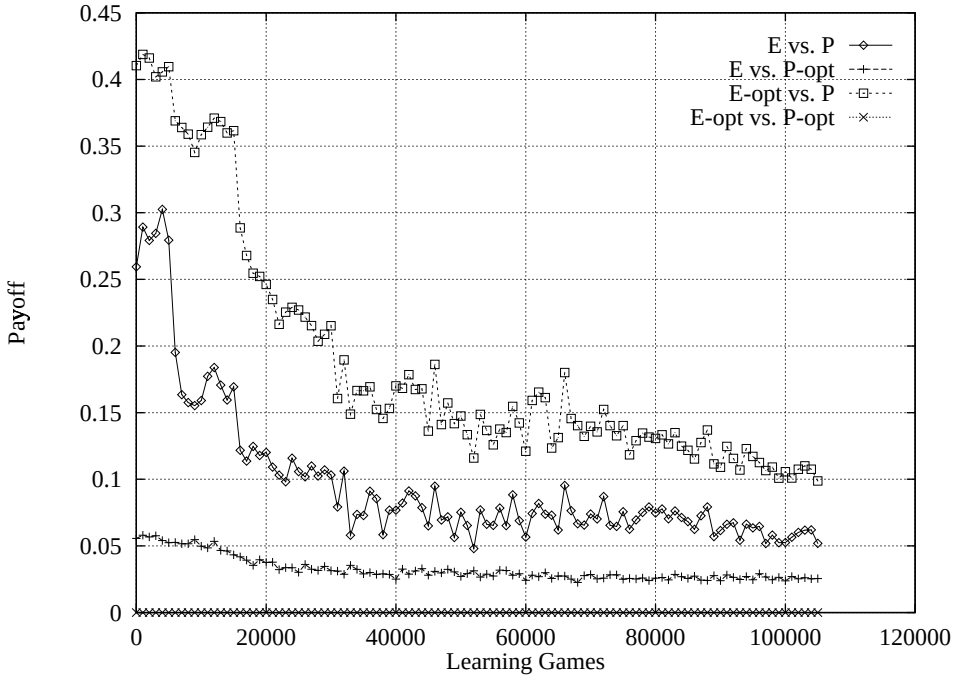


Figure 7.10: Deviation from optimal for pursuit game in half plane.

performance is still relative good. Further, as learning proceeds, E clearly changes its strategy and approaches optimal. After 100,000 games, E's performance appears to have flattened out

For the pursuer, performance is also similar to no boundary. Once again, we find that performance still seems to be improving through 100,000 games. This trend is especially visible when examining Figure 7.10. We also see the trend moving from E always getting away to E losing ground. Nevertheless, the performance compared to Section 7.3.2 is degraded as well.

These degradations in performance are not surprising for several reasons. First, we introduced an "obstacle" that increases the state space (we must now keep track of our distance to the boundary). Second, this boundary complicates E's strategy due to the sudden shift in performance when E approaches the boundary.

We also note that the impact of node splitting is visible again. This time, however, there appears to be a slight improvement for both E and P during the first 5000 games. As before, a sudden change in performance occurs when the first split is made. This time, however, the second split (at 10,000 games) has no noticeable effect. This suggests the possibility of *TBCL* periodically choosing an inappropriate node to split. However, at 15,000 games, we see another sudden change, indicating *TBCL* found a node to split that would help. This suggests further study in selecting a node for splitting would be appropriate and beneficial.

7.3.4 Pursuit with Limited Mobility

Finally, we apply *TBCL* to the pursuit game with limited mobility. This game is identical to the game described in Section 6.3.4. Once again, we removed the boundary, but we also limited the mobility of both players to permit them to make instantaneous turns within a constrained range of possible turns.

We do make one change in the experiments. Specifically, we only permit training to take place through 20,000 games. We test after 250 games, as before, but this time we also split after 1000 games. This change was justified by the fact there was little evidence of change between splits. Consequently, we reduced the number of games played before splitting. We still wanted to monitor the progress between splits, so we increased the frequency of testing accordingly.

The results of these experiments were both encouraging and surprising. The performance learning curves are shown in Figure 7.11 with a comparison to heuristic play shown in Figure 7.12. These graphs were produced by running the experiment ten times and averaging the results at 250 game intervals. This time, the track marked with an 'x' indicates heuristic play by both players, the track marked with a diamond indicates both players using their currently learned strategies, the track with a square indicates P's performance against a heuristic E, and the track marked with a plus indicates E's performance against a heuristic P.

The results of this experiments, especially when compared to *MBCL* (see Section 7.4) were quite encouraging but with a surprise. First, it was clear that E's random strategy

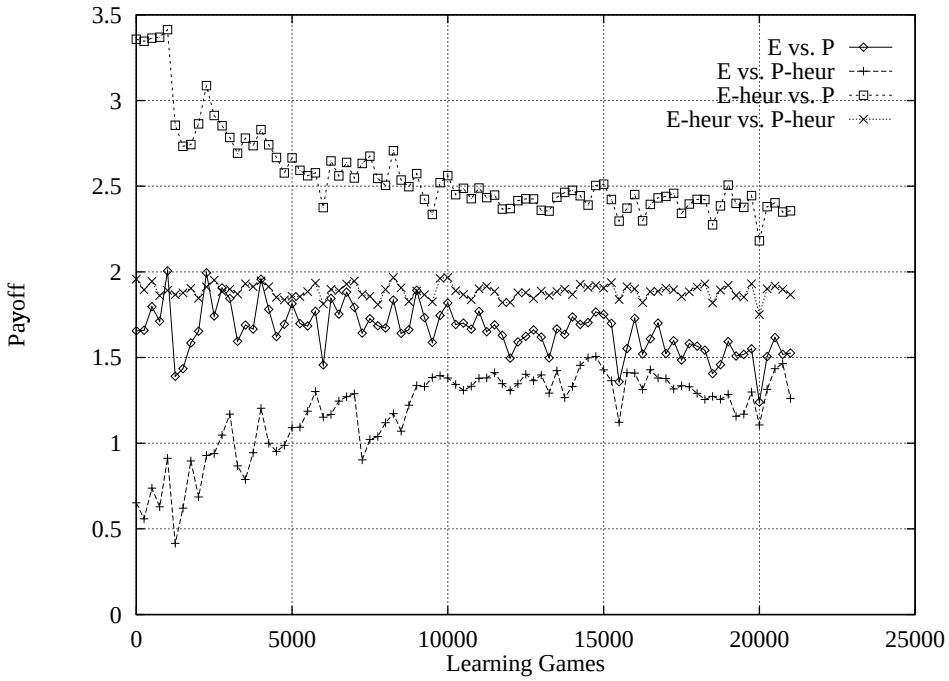


Figure 7.11: Learning performance for pursuit game with limited mobility.

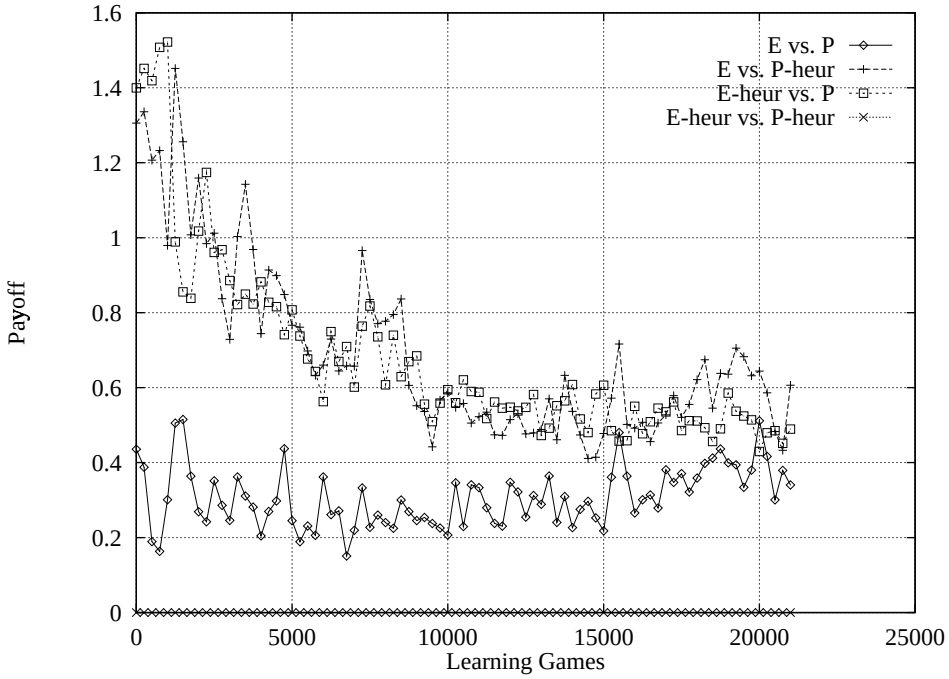


Figure 7.12: Deviation from the heuristic for pursuit game with limited mobility.

was not satisfactory against an optimal P, and E learned a strategy that became competitive (although still somewhat inferior). Improvement by P relative to an optimal opponent was similar.

The surprising result concerns comparing E versus P to optimal E versus optimal P. As we see, optimal performance is fairly constant throughout (which it should be). The performance of E versus P; however, was not constant. Of course, this is what we would like, except that we found the performance to *diverge* from optimal. Even though optimal performance appears to yield an average payoff of approximately 1.9, we found the average payoff for E versus P to drop from around 1.7 (at 10,000 games) to about 1.5 (at 20,000) games.

Once again, we believe the reason for this unpredicted behavior is associated to the algorithm for selecting a node to split. As shown in Section 2.5, evasion depends on E getting inside P's radius of curvature. According to Basar and Olsder [39], this requirement is further complicated by a fact the BUP (boundary of the usable part) of the Homicidal Chauffeur game has a "leaky corner." A leaky corner is a characteristic in the surface between terminal conditions of the game in which performance cannot be forced by either player. The leaky corner is even more problematic in our game in which *both* P and E have limited mobility (rather than just P).

To be able to learn this surface may require many more splits in the decision tree. Further, because our method for selecting a node to split is biased towards nodes that are frequently updated, and we hope that we are in the region with the leaky corner relatively

infrequently, our method is probably not well suited for learning these characteristics. Thus, a substantially larger number of node splits may be required with our method.

7.4 Comparing *MBCL* and *TBCL*

The results of both *MBCL* and *TBCL* are both interesting and encouraging. Nevertheless, their respective performances are (not surprisingly) different. In this section, we attempt to capture the similarities and differences in the performance of these two algorithms on the fours games and attempt to explain the reasons for these differences.

To facilitate such a comparison, we need to make the comparison as fair as possible. Specifically, we would like to compare based on similar knowledge complexity and similar learning complexity. We are faced, immediately, with an issue to be resolved. For *MBCL*, the knowledge is captured in a fixed-size memory base, and learning consists of modifying Q values associated with the examples in the memory base. For *TBCL*, on the other hand, the knowledge is captured in a decision tree that partitions the space. The partitioning is representative of the examples in *MBCL*'s memory base, except that each partition contains a complete game matrix, and an instance in the memory base only contains a single action. Thus, given ten strategies per player, one partition in *TBCL* is comparable to 100 examples, all centered in the middle of the partition, for *MBCL*.

Comparing the learning complexity is a bit tricky. In *MBCL*, Q-learning is associated with the nearest neighbors used to generating the game matrices in each state of play. Assume a game lasts 10 steps. Because all experiments set $k_s = 30$, a single game would

result in 300 Q-updates. In *TBCL*, Q-learning is associated with the cells in the game matrix used to select the actions for the two players. Thus, for that game with 10 steps, there would only be 10 Q-updates in a single game. The comparison is further complicated by the fact the tree is not fixed in size but grows throughout learning.

7.4.1 A Game of Force

We begin by considering the relative performance of the two algorithms on our simple game of force. Recall that the optimal strategy for this game was independent of state and was constant. Given the issues presented above, we would like to consider the two algorithms at comparable points. The size of *MBCL*'s memory base is fixed and has 50,000 points. For a comparable point in *TBCL*, we need a tree with 500 leaves. None of our experiments permitted the trees to grow this large. In fact, we limited this game such that only one node existed in the tree.

In comparing the number of updates, we find that in the worst case, *MBCL* would update 750 examples per game. Thus, after 10,000 games, at most 7,500,000 updates would have been made. Few games required 25 steps, so the actual number is much lower than this. For *TBCL*, after 100,000 games, we would have at most 2,500,000 updates. Of course, this would be over a significantly smaller number of possible stored examples.

Clearly, we cannot make a direct comparison between these two algorithms under the current conditions and be fair. Even so, we can point out some interesting differences. First, we would expect the state of convergence for *TBCL* to be beyond the corresponding

state for *MBCL* because *TBCL* averaged the equivalent of 25,000 updates per cell and *MBCL* averaged 750 updates per example. This implies *TBCL* should perform closer to optimal that *MBCL*. This was not the case. One possible reason for this is that *MBCL* updates a set of points associated with each stage of the game, thus the effects of updating are "smoothed" over a region in the state-action space. *TBCL*, on the other hand, updates only a single cell with each stage of the game; such smoothing is not experienced.

In spite of such difference in playing ability, it is significant to note that *TBCL* reached its level of performance in one-third of the time as *MBCL*. This is a graphic illustration of the computational burden associated with the memory-based approach when no optimization of the memory based has been done.

7.4.2 Pursuit with Simple Motion

In considering the pursuit game with simple motion, we also compare the relative complexity of the knowledge and the Q-update process. Once again, the size of the *MBCL* memory base was fixed. This time it had 400,000 points which was significantly larger than the memory base in the game of force. To be comparable, we would expect the corresponding tree in *TBCL* to have 4,000 leaves. Recall we limited the size of the trees to 20 leaves, so again we are not able to provide a fair comparison based on size.

Considering the number of updates, we find that in the worst case, *MBCL* would update 750 examples per game. After 5000 games, at most 3,750,000 updates would have been made. For *TBCL*, after 100,000 games, we would have at most 2,500,000 updates.

Once again, this is over a significantly smaller number of possible stored examples. This time, updating amounts to 25,000 updates per cell for *TBCL* and less than 10 updates per example for *MBCL*!

Given the diverse results with the simple game of force, we might be surprised to see any improvement at all in *MBCL*. Further, we would expect *TBCL* to far surpass the performance of *MBCL*. In fact, *MBCL* still performed better, coming to within 0.02 of optimal for both players while *TBCL* only came within 0.1 of optimal for P and within 0.02 of optimal with E.

In spite of the apparent superior performance of *MBCL* over *TBCL*, we need to consider the relative difference in computational burden. The 100,000 games of *TBCL* required approximately $\frac{1}{20}$th of the time for the 5000 games of *MBCL*. Given *TBCL* still showed signs of improving, it is possible that making the learning times equivalent and equalizing the complexity of the stored knowledge would lead to comparable playing ability. Unfortunately, preliminary experiments expanding the tree to 250 leaves did not support this hypothesis, further indicating problems in the node splitting and node selection process.

7.4.3 Pursuit with Simple Motion in a Half Plane

Recall that the pursuit game in the half plane was a simple extension of the pursuit game with simple motion. In fact, all of the parameters relative to the two pursuit games are identical, except for the size of the state space. Therefore, we can conclude, once again, that the two algorithms cannot be consistently compared. Nevertheless, we can make some

interesting observations.

First, we find that *MBCL* apparently continues to surpass *TBCL* in play. However, it is interesting that both this and the previous game show a greater improvement in performance in *TBCL* than in *MBCL*. Specifically, when learning began, *MBCL*'s deviation from optimal for the simple pursuit game was limited to 0.2 and 0.02 for P and E respectively. *TBCL*'s deviation from optimal, on the other hand, was at 0.45 and 0.025 respectively. For the pursuit game in the half plane, *MBCL*'s initial deviation from optimal was 0.14 and 0.015 for P and E respectively where *TBCL*'s initial deviation from optimal was 0.43 and 0.05 for P and E respectively. The performance for E was comparable for both algorithms, but *TBCL* played twice as "badly" as *MBCL* before any learning occurred. This shows the advantage of the memory-based approach in having information about the problem from the initial experiences that seeded the memory base.

7.4.4 Pursuit with Limited Mobility

It is with the pursuit game with limited mobility where we see comparable, if not superior, performance by *TBCL*. As in the previous experiments, the memory and update burdens are not comparable because *MBCL* has 560,000 examples and requires 7,500,000 updates through 10,000 games. *TBCL*, on the other hand only has 20 leaves and only makes 500,000 updates over 20,000 games. But with significantly fewer possible examples and significantly fewer updates, we find *TBCL* yields performance within 0.6 and 0.4 of the heuristic for P and E respectively. *MBCL*, on the other hand, yields performance of 0.9 and 0.3 of the heuristic

for P and E respectively. Further, we note that, at one point, E is performing within 0.2 of optimal for *TBCL* but then degrades. We conjecture that an improved algorithm for selecting a node to split could enable *TBCL* to maintain E's performance or even surpass it. In addition, making the computation times and structures comparable in size could also enable *TBCL* to further surpass *MBCL*.

7.5 Discussion

We were pleased with the results of *TBCL*, especially when compared to the performance of *MBCL*. First, we found the overall learning performance to be quite good. In fact, we feel the performance was clearly "comparable" to the performance of *MBCL*—even with several known deficiencies in *TBCL*. Further, as the games became more complex, the experimental results seemed to indicate that the tree-based approach could ultimately adapt better to the underlying state-action space; however, we did not try any variable-resolution memory-based strategies to compare. It is possible that the advantages of the tree-based strategy can be attributed to the variable resolution which can be replicated in memory-based approaches.

In addition to the comparable performance of *TBCL* relative to *MBCL*, we also found a substantial improvement in computational and memory burden. As before, all of the experiments for *TBCL* were run on either Sun Sparc 2 or Sun Sparc 10 processors. To give an idea of the improvement in computational burden, Table 7.2 shows the time required for playing the large number of games (in the best case) and the size of the associated trees

Table 7.2: Relative computational burdens for solving games with *TBCL*.

Game	Games Played	Leaves	Minutes
Force	100,000	1	31
Simple	100,000	20	49
Half	100,000	20	43
Limit	20,000	20	22

Table 7.3: Relative computational burdens for solving games with *MBCL*.

Game	Games Stored	Examples Stored	Minutes
Force	7500	50,000	96
Simple	2000	400,000	1376
Half	2000	400,000	1372
Limit	2800	560,000	3193

(in number of leaves—because all of the trees are binary, we know the number of nodes in the tree is twice the number of leaves minus 1).

As we see, the times required to learn these games were substantially less than the times required for *MBCL* (Table 7.3). Given a more clever approach to splitting nodes, we may have been able to yield even stronger performance with the same, or possibly fewer, numbers of nodes. Further, we could have increased the search space by providing a finer resolution on the strategy space and still been able to learn in a reasonable period of time.

7.6 Summary

In this chapter, we provided a second novel algorithm for co-learning based on dynamically partitioning the state space of the game. The focus of the approach was on reducing memory and computation requirements while maintaining or improving upon the performance

obtained by *MBCL*. The resulting algorithm, *TBCL*, accomplished these goals by not requiring explicit storage of examples in a memory base and by keeping game matrices at each of the leaves of the tree. Examples were replaced by state space partitions covering a region of the space. Since the regions can be partitioned to any required resolution, this approach can maintain the level of performance of the memory-based approach without explicitly storing examples.

In addition, since game matrices are kept with each partition, several computational advantages are obtained. First, the game matrix does not need to be regenerated at every step of the game. Second, if multiple steps in the game take place within the same partition of the state space, only one linear program needs to be solved (rather than one for each visit to the partition). Finally, following learning, the game matrix can be thrown away (thus further reducing memory requirements), and the current strategies stored with the partition used for play—there is no need to solve *any* linear programs during actual use. The result is an approach to co-learning that is faster than memory-based learning during both training and actual use.

Chapter 8

Conclusions

8.1 Summarizing the Results

In this dissertation, we provide a new focus for research in multi-agent reinforcement learning. Our focus on problems of game playing is from two perspectives. First, we focused on problems of differential games in which games take place through continuous time and have continuous state spaces and continuous action spaces. Second, our two novel approaches involved multi-agent learning in which both players in a two-player game are learning together.

This dissertation provides a coordinated view of the problem of learning solutions to differential games. Our introduction to the field of differential game theory (Chapter 2) includes some basic definitions from game theory, properties of the most commonly studied games, and a solution concept for games based on Nash equilibria.

We next introduce Markov decision processes and Markov games. Because our

experiments focus on discrete simulations of differential games, we can approximate the games by implementing them as comparable Markov games. We discuss two standard algorithms for solving Markov decision process—value iteration and policy iteration—and explain how these can be applied to solve Markov games.

Next we focus on differential games themselves. We introduce the basic parts of a differential game, including the kinematic equations that describe the dynamics of the game, the termination criteria (also called the target set) for the game, and different types of payoff functions. We highlight the pursuit game as an interesting class of differential game and introduce the most famous pursuit game in the field—the Homicidal Chauffeur.

Finally, we introduce two relatively simple differential games and provide closed formed solutions for each. These games, the simple game of force and the pursuit game with simple motion, are simplified forms of two classic games in differential game theory—the *dolichobrachistochrone* game and the Homicidal Chauffeur game. These games are then used in experiments testing two new algorithms for co-learning.

After introducing the field of differential game theory and solving two simple games, we provide a review of recent research in reinforcement learning and game playing. The literature on machine learning and game playing is extensive, and with recent interest in reinforcement learning, the literature describing applications of reinforcement learning to game playing is growing at a rapid rate. Consequently, we concentrate on a few representative approaches and highlight the results and applications to co-learning in games.

In spite of the growing literature in reinforcement learning and game playing, relatively little work exists in co-learning and game playing. With the exception of some work in artificial life, results in co-learning have been limited to relatively simple games, often with the intent to provide an interesting diversion rather than to focus on the problem of co-learning. In fact, only one instance of co-learning and differential games was found. This work involved using temporal difference learning coupled with residual, advantage updating in a neural network to learn a single linear-quadratic differential game. The work yielded interesting results but was limited to providing pure-strategy results rather than the more general mixed-strategy results.

Following these reviews, we focus on the experimental work of this dissertation. The experiments focus on four areas to further develop a coordinated view of research in learning solutions to differential games. We compare three distinct learning algorithms in solving strategies for one player in two different differential games of pursuit. The three algorithms studied include a genetic algorithm based on the Navy's SAMUEL architecture, nearest neighbor classification, and Q-learning on a simple memory base. The two differential games include a two-player game in which a single pursuer attempts to capture a single evader and a three-player game in which two pursuers attempt to capture a single evader. Learning focuses on identifying strategies for the evader in the presence of fixed pursuers. In the two-player game, the evader was able to control only the direction in which it turned. In the three-player game, the evader was able to control the turn angle, the speed, and the release of countermeasures.

The results of these experiments demonstrate superior performance for the genetic algorithm, strong performance for Q-learning, and mixed performance for 1-NN. In particular, 1-NN was able to learn quite well in the two-player game but quickly got bogged down with bad examples in the three-player game. These results motivated the experiments that followed.

Given the strong performance of the GA on the three-player game and the weak performance of 1-NN on the three-player game, an experiment was performed in which the GA acted as a teacher to 1-NN. This experiment was also motivated by recent work in creating teachers and advisors in machine learning. Most of the results in teaching either focus on external oracles or focus on using one learning algorithm to encode knowledge for another learning algorithm to use. In this experiment, we provided a method whereby one learning algorithm was coupled with a second learning algorithm, and the resulting performance of the combined strategy exceeded the performance of either strategy alone.

These experiments combining several algorithms and coupling two algorithms together set the stage for the final and most significant part of the dissertation. First, we introduce a new memory-based reinforcement learning algorithm, called *MBCL* (Memory-Based Co-Learning), which provides a strategy to learn solutions to differential games from examples. Examples are stored in the memory base with associated Q-values predicting future discounted reward. Based on these predictions, game matrices are constructed on the fly and solved to provide optimized behavioral strategies based on the current experience. The Q-values are updated using Q-learning with the experience playing games.

MBCL was applied to four differential games that gradually increased in complexity. The first two games were the simple game of force and the simple pursuit game introduced earlier. The third game extended the pursuit game by introducing a barrier in the playing field. This barrier caused an interesting discontinuity in the strategy for the evader which complicated learning. The fourth game also extended the simple pursuit game by limiting the mobility of the two players. The result was that a singularity was introduced into the surface that characterized the termination conditions called a "leaky corner." Although requiring a large amount of training time and a large amount of memory for the memory base, *MBCL* was able to learn reasonable strategies for all four games.

Finally, we introduce a new tree-based reinforcement learning algorithm, called *TBCL* (Tree-Based Co-Learning). This learning algorithm also learns to solve differential games from examples but does not explicitly store examples in a memory base. Instead, *TBCL* constructs a decision tree that partitions that state space for the game. Associated with each partition in the state space is a game matrix that can be used to derive behavioral strategies for the game. Learning takes place in two parts. First the cells in the game matrix are updated through Q-learning whenever a partition is visited and a pair of actions is selected. Second, the tree is grown by further partitioning the space to better sample the state space.

TBCL was applied to the same four games as was *MBCL*. Playing ability following learning was comparable between the two strategies. More significant, *TBCL* was able to obtain this comparable level of performance with a significantly smaller knowledge base and

in significantly less time.

8.2 Contributions of this Dissertation

This dissertation reports on several major contributions to the field of multi-agent reinforcement learning and learning in game playing. Two different views of multi-agent reinforcement learning where considered. The first assumed a single agent learning but performing in an environment with other, fixed agents. The second assumed two agents interacting in the same environment and both agents learning simultaneously.

The specific contributions of this dissertation include the following. First, we provide a detailed review of the field of machine learning and game playing, with an emphasis on co-learning, reinforcement learning, and Markov games. We discuss the results and relative merits of several approaches to game learning and suggested approaches for extending work in single-agent learning to the multi-agent, co-learning problem.

Second, we provide a direct comparison of three distinct learning algorithms on two difficult reinforcement learning tasks. These tasks are single-agent pursuit games in which one player applies a fixed strategy and the other player attempts to learn a strong opposing strategy. The process of performing this comparison resulted in adapting two traditional learning algorithms—nearest neigbor classification and Q-learning—to work on tasks with large state and action spaces.

The results of these experiments motivated the third significant contribution: a novel bootstrapping algorithm in which one learning algorithm learns a task and provides

the results of its experience to a memory-based learning algorithm (i.e., 1-NN) on a difficult pursuit game. This experiment also focused on a single player learning against a fixed opponent but focused on applying the experience from the "best" algorithm in the previous comparison to the "worst" algorithm. The result was a team approach to learning in which the final results exceeded the results of either algorithm alone. In addition, an editing algorithm was applied to the learned memory base to determine the robustness of the examples. For this experiment, we used the genetic algorithm to provide the examples and found that with a memory base comparable in size to a rule base generated by the GA, similar performance could be obtained, thus graphically illustrating the ability to effectively transfer knowledge between the two algorithms.

Fourth, we introduce a novel memory-based learning algorithm to find approximate solutions to differential games. This algorithm permits the opposing players in a two-player game to learn together. The algorithm assumes a shared memory base and applies Q-learning to the memory base to learn expected discounted rewards. During play, game matrices that characterize behavioral strategies are constructed and solved using linear programming to determine optimal mixed strategies for each player.

Finally, we introduce a novel tree-based learning algorithm to find approximate solutions to differential games. As with the memory-based algorithm, this algorithm permits the opposing players in a two-player differential game to learn together. This algorithm also assumes a shared knowledge base, but the knowledge base consists of a decision tree that partitions the state space. Associated with each partition is a game matrix that is

modified through Q-learning and solved through linear programming. This algorithm offers tremendous promise over the memory-based approach in that learning requires significantly less time and game matrices need not be evaluated during play. In addition, the size of the decision tree is significantly smaller than the size of the memory base.

8.3 Areas for Future Research

This dissertation represents the beginning of the work to be done in the area of reinforcement learning and differential games. Several areas of future research can be pursued to expand upon the results reported here.

Because the games reported in this dissertation are limited to two dimensions, work exploring games of higher dimensionality (e.g., x, y, z) is necessary. Further, in many ways, the games described do not correspond to similar games in the real world; therefore, games characterizing more realistic capabilities (e.g., noisy sensors, imperfect controllers) should be encouraged.

Comparative research is an important component of machine learning studies. The task of comparing algorithms on problems such that the comparisons are fair and meaningful is difficult. In this dissertation, we compared three single-agent learning algorithms on two single-agent learning tasks, and we compared two multi-agent learning algorithms on four multi-agent learning tasks. Although the results suggest likely differences in the algorithms' performance, they were in no way conclusive. In fact, we explaining why the comparisons in the multi-agent case were not completely fair.

We note that all the algorithms discussed in this dissertation are limited to symbolic reasoning systems. Specifically lacking are any algorithms derived from the connectionist (i.e., neural network) community, despite the fact that much of the successful research in reinforcement learning has been applied to connectionist systems, with symbolic systems largely limited to lookup tables. Additional work that applies the ideas in this dissertation to connectionist systems would be warranted. Of particular interest would be work integrating connectionist and symbolic systems into a cohesive multi-agent learner. For example, an interesting architecture might include an artificial neural network to provide a fitness function for filling out a game matrix. Harmon and Baird's approach [161] to using a neural network could easily be extended to include a full evaluation of the linear program and its dual.

Several variations of *MBCL* should be considered. For example, limited seeding followed by variable resolution memory-based learning (such as the parti-game algorithm) would provide a potential solution to the problem of appropriate sampling and excessive memory-base size. In addition, using a data structure such as the *kd*-tree to store the memory-base could significant speed up learning and testing.

Throughout this dissertation, no concerted effort was made to identify "optimal" parameters for the learning algorithms. Both *MBCL* and *TBCL* have a relatively large number of parameters that need to be set. Evaluating the effects of various parameter settings, for example through a factorial study, would provide considerable insight into the power of the algorithms and their ability to find reasonable parameters in other games.

In a similar issue, more work is needed in providing exploration strategies during learning. In Chapter 6, we point out that the mixed strategies resulting from evaluating game matrices provide an implicit approach to handling the k-armed bandit problem. This is only true if the resulting mixed strategies provide non-zero probabilities on all of the pure strategies. To address this problem during learning, we assigned a minimal probability to all strategies to ensure exploration. Other exploration strategies should be considered to further address the problem.

As with *MBCL*, several variations on *TBCL* could be explored. For example, we discuss at length in Chapter 7 that the method for selecting a node to split is naive at best. It may be possible to apply selection techniques such as those used in traditional classification decision trees to characterize potential improvement (e.g. entropy reduction, minimum description length, minority measures). As an alternative, it may be worthwhile to explore techniques of k-step look-ahead to evaluate a node for splitting. Under such a method, a small number of splits are selected and evaluated. The best split is then selected, and learning continues from that point.

Related to the problem of selecting a node for splitting is the problem of selecting an attribute and associated value for splitting. In selecting an attribute, we selected the attribute that maximized the difference between resultant submatrices. Once again, principles such as entropy reduction or minimum description length may be appropriate.

In all cases, we assumed that the split value would be the midpoint of the region. Again, this may not be appropriate. Because we are not splitting examples, we cannot select

regions between neighboring examples; however, a similar quantization of the attribute space may be appropriate.

Finally, it may be appropriate to consider non-axis-parallel trees in growing trees for *TBCL*. Work by Heath and Murthy has pointed out several issues and offered several suggestions for constructing oblique decision trees and addressing concerns such as look-ahead and splitting criteria [166, 247].

In Section 7.4 we point out that one of the significant differences between *MBCL* and *TBCL* is that Q-updates in *MBCL* occur over a region in the instance space where Q-updates in *TBCL* apply only to individual cells in the game matrix. An interesting variation to *TBCL* would apply a weighted update, such as

$$Q(s, a_p, a_e) = (1 - w\eta_i)Q(s, a_p, a_e) + w\eta_i[\rho + \gamma Q(s', \pi(s'))]$$

where w is a weight based on proximity to the target cell. Thus, cells in a region around the target cell to be updated could also be updated. This approach is motivated by the fact that the game matrix for a differential game would, frequently, be smooth.

It should be evident from this discussion that considerable work can be done to advance the results reported in this dissertation. We believe that the area of reinforcement learning is exciting and that considerable promise exists in the approach to solving several significant problems in multi-agent learning. It is our hope that the work reported here will motivate others to take the next step and provide even better algorithms to solve complex problems such as differential games.

Bibliography

[1] E. Aboaf, S. Drucker, and C. Atkeson. Task-level robot learning: Juggling a tennis ball more accurately. In *Proceedings of the IEEE Conference on Robotics and Automation*. IEEE, 1989.

[2] D. Aha. Incremental, instance-based learning of independent and graded concept description. In *Proceedings of the Machine Learning Workshop*, 1989.

[3] D. Aha. *A study of instance-based algorithms for supervised learning: Mathematical, empirical and psychological evaluations*. PhD thesis, Department of Information and Computer Science, University of California, Irvine, 1990.

[4] D. Aha. Case-based learning algorithms. In *Proceedings of the 1991 DARPA Case-Based Reasoning Workshop*. Morgan Kaufmann, Publishers, 1991.

[5] D. Aha. Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, **16**:267–287, 1992.

[6] D. Aha and D. Kibler. Noise-tolerant instance-based learning algorithms. In *Proceedings of IJCAI-89*, pages 794–799, Detroit, MI, 1989. Morgan Kaufmann.

[7] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, **6**(1), 1991.

[8] D. Aha and S. Salzberg. Learning to catch: Applying nearest neighbor algorithms to dynamic control tasks. In *Proceedings of the Fourth International Workshop on AI and Statistics*, pages 363–368, Ft. Lauderdale, 1993.

[9] S. Aihara and A. Bagchi. Linear-quadratic stochastic differential games for distributed parameter systems. *Computers and Mathematics with Applications*, **13**(1–3):247–259, 1987.

[10] M. Albert and D. Aha. Analyses of instance-based learning algorithms. In *Proceedings of the Ninth National Conference on Artificial Intelligence*. AAAI, 1991.

[11] L. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, 1994.

[12] H. Almuallim and T. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547–552. AAAI, 1991.

[13] H. Almuallim and T. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, **69**:279–305, 1994.

[14] E. Alpaydin. Voting over multiple condensed nearest neighbors. Technical Report TR-80816, Bogazici University, 1995.

[15] E. Altman and G. Koole. Stochastic scheduling games with markov decision arrival processes. *Computers and Mathematics with Applications*, **26**(6):141–148, 1993.

[16] C. Anderson and S. Crawford-Hines. Multigrid Q-learning. Technical Report CS-94-121, Department of Computer Science, Colorado State University, 1994.

[17] M. Ardema and N. Rajan. An approach to three-dimensional aircraft pursuit-evasion. *Computers and Mathematics with Applications*, **13**(1–3):97–110, 1987.

[18] S. Arya and D. Mount. Asymptotically efficient randomized algorithm for nearest neighbor searching. Technical Report CS-TR-3011 or UMIACS-TR-92-135, Computer Science, University of Maryland, College Park, MD, December 1992.

[19] S. Arya and D. Mount. Efficient heuristic for nearest neighbor searching. Technical Report CS-TR-3012 or UMIACS-TR-92-136, Computer Science, University of Maryland, College Park, MD, December 1992.

[20] M. Asada, E. Uchibe, and K. Hosoda. Agents that learn from other competitive agents. In *Proceedings of the Workshop on Agents that Learn from Other Agents*, 1995.

[21] H. Atir. Nonlinear effects in a variable speed pursuit-evasion game. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 58–66. Springer-Verlag, 1991.

[22] C. Atkeson. Model-based robot learning. Technical Report AI Memo No. 1024, Massachusetts Institute of Technology, 1988.

[23] C. Atkeson. Using local models to control movement. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 316–323, San Mateo, CA, November 1990. Morgan Kaufman.

[24] C. Atkeson. Using locally weighted regression for robot learning. In *Proceedings of the IEEE International Conferences on Robotics and Automation*, pages 958–963. IEEE, 1991.

[25] C. Atkeson. Memory-based approaches to approximating continuous functions. In M. Casdagli and S. Eubanks, editors, *Nonlinear Modeling and Forecasting*, pages 503–521. Addison Wesley, 1992.

[26] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 1995. To appear.

[27] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 1996. To appear.

[28] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[29] L. Baird. Advantage updating. Technical Report WL-TR-93-1146, Wright Laboratory, 1993.

[30] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufmann, 1995.

[31] L. Baird and A. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, 1993.

[32] M. Bardi and P. Soravia. Approximation of differential games of pursuit-evasion by discrete-time games. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 131–143. Springer-Verlag, 1991.

[33] E. Bareiss, B. Porter, and C. Wier. PROTOS- an exemplar based learning apprentice. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 12–23, Irvine, CA, 1987. Morgan Kaufmann.

[34] A. Barto, S. Bradtke, and S. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57, COINS, University of Massachusetts, 1991.

[35] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1993.

[36] A. Barto, R. Sutton, and C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, **13**:835–846, 1983.

[37] A. Barto, R. Sutton, and C. Watkins. Learning and sequential decision making. In Gabriel and Moore, editors, *Learning and Computational Neuroscience*, pages 539–602, Cambridge, 1990. MIT Press.

[38] T. Basar and P. Kumar. On worst case design strategies. *Computers and Mathematics with Applications*, **13**(1–3):239–245, 1987.

[39] T. Basar and G. Olsder. *Dynamic Noncooperative Game Theory.* Academic Press, London, 1982.

[40] K. Basye, T. Dean, and L. Kaelbling. Learning dynamics: System identification for perceptually challenged agents. *Artificial Intelligence*, **72**, 1995.

[41] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[42] D. Belsley. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* Wiley & Sons, New York, 1980.

[43] S. Benson. Action model learning and action execution in a reactive agent. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[44] S. Benson. Inductive learning of reactive action models. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[45] S. Benson and N. Nilsson. Reacting, planning, and learning in an autonomous agent. *Machine Intelligence 14*, 1995.

[46] J. Bentley. Multidimensional divide and conquer. *Communications of the ACM*, **23**(4):214–229, 1980.

[47] P. Bernhard, A. Colomb, and G. Papavassilopoulos. Rabbit and hunter game: Two discrete stochastic formulations. *Computers and Mathematics with Applications*, **13**(1–3):205–225, 1987.

[48] D. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Inc., 1987.

[49] D. Blackwell and M. Girshick. *Theory of Games and Statistical Decisions.* Dover Publications, Inc., 1954.

[50] L. Booker, D. Goldberg, and J. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, **40**:235–282, 1989.

[51] D. Borrajo and M. Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review*, 1996. To appear.

[52] J. Boyan. Modular neural networks for learning context-dependent game strategies. Master's thesis, University of Cambridge, 1992.

[53] J. Boyan and A. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

[54] S. Bradtke. Reinforcement learning applied to linear quadratic regulation. In *Neural Information Processing Systems 5*, pages 295–302, 1993.

[55] S. Bradtke and S Duff. Reinforcement learning methods for continuous time markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 393–400. The MIT Press, 1995.

[56] R. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. *Journal of Artificial Intelligence Research*, **3**:477–507, 1996.

[57] S. Brams. *Theory of Moves*. Cambridge University Press, 1994.

[58] J. Breese and D. Heckerman. Decision-theoretic case-based reasoning. Technical Report MSR-TR-95-03, Microsoft Research, 1995.

[59] L. Breiman. The π method for estimating multivariate functions from noisy data. *Technometrics*, **33**(2):125–143, 1991.

[60] L. Breiman. Stacked regressions. Technical Report TR-367, Department of Statistics, University of California at Berkeley, 1992.

[61] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[62] H. Bremermann and J. Pickering. A game-theoretical model of parasite virulence. *Journal of Theoretical Biology*, **100**:411–426, 1983.

[63] M. Breton. Approximate solutions to continuous stochastic games. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 257–264. Springer-Verlag, 1991.

[64] M. Buro. Statistical feature combination for the evaluation of game positions. *Journal of Artificial Intelligence Research*, **3**:373–382, 1995.

[65] C. Byrne and P. Edwards. Collaborating to refine knowledge. In *Proceedings of the ML95 Workshop on Agents that Learn from Other Agents*, 1995.

[66] C. Cardie. Using decision trees to improve case-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 25–32, University of Massachusetts, Amherst, 1993.

[67] D. Carmel and S. Markovich. The M^* algorithm: Incorporating opponent models into adversary search. Technical Report CIS-9402, Technion–Israel Institute of Technology, March 1994.

[68] D. Carmel and S. Markovitch. Learning models of opponent's strategy in game playing. Technical Report CIS-9318, Technion–Israel Institute of Technology, June 1993.

[69] D. Carmel and S. Markovitch. Opponent modeling in an multi-agent system. In *Proceedings of the IJCAI '95 Workshop on Adaptation and Learning in Multiagent Systems*, 1995.

[70] A. Cassandra. Optimal policies for partially observable markov decision processes. Technical Report CS-94-14, Department of Computer Science, Brown, University, 1994.

[71] A. Cassandra, L. Kaelbling, and M. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1023–1028. AAAI, 1994.

[72] W. Chan and S. Ng. Partially observable linear-quadratic stochastic pursuit-evasion games. *Computers and Mathematics with Applications*, **13**(1–3):181–189, 1987.

[73] C. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, **23**(11):1179–1184, November 1974.

[74] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, **32**:333–377, 1987.

[75] D. Chapman and P. Agre. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*, 1987.

[76] P. Cichosz. Reinforcement learning algorithms based on the methods of temporal differences. Master's thesis, Warsaw University of Technology Institute of Computer Science, 1994.

[77] P. Cichosz. Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal of Artificial Intelligence Research*, **2**:287–318, 1995.

[78] P. Cichosz and J. Mulawka. Fast and efficient reinforcement learning with truncated temporal differences. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[79] P. Clark and T. Niblett. Induction in noisy domains. In *Proceedings of the second European workshop on Machine Learning*, 1987.

[80] W. Cleveland, S. Devlin, and E. Grosse. Regression by local fitting: Methods, properties, and computational algorithms. *Journal of Econometrics*, **37**:87–114, 1987.

[81] J. Clouse and P. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 92–101, Aberdeen, Scotland, 1992. Morgan Kaufman.

[82] D. Cohn, Z. Ghahramani, and M. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, **4**:129–145, 1996.

[83] R. Collins. *Studies in Artificial Evolution.* PhD thesis, University of California at Los Angeles, Los Angeles, California, 1992.

[84] M. Colombetti and M. Dorigo. Training agents to perform sequential behavior. *Adaptive Behavior*, **2**(3):247–275, 1994.

[85] M. Colombetti and M. Dorigo. Behavior analysis and training: A methodology for behavior engineering. *IEEE Transactions on Systems, Man, and Cybernetics*, **26**(6), 1996.

[86] D. Comer and R. Sethi. The complexity of trie index construction. *Journal of the ACM*, **24**(3):428–440, July 1977.

[87] S. Conry, R. Meyer, and V. Lesser. Multistage negotiation in distributed planning. Technical Report COINS Technical Report 86-67, University of Massachusetts, Amherst, Massachusetts, 1986.

[88] M. Corless, G. Leitmann, and J. Skowronski. Adaptive control for avoidance or evasion in an uncertain environment. *Computers and Mathematics with Applications*, **13**(1–3):1–11, 1987.

[89] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* The MIT Press and McGraw-Hill Book Company, 1990.

[90] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, **10**(1):57–78, 1993.

[91] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, **13**:21–27, 1967.

[92] T. Cover and J. van Campenhout. On the possible orderings in the measurement selection problems. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-7**(9), 1977.

[93] M. Cox and A. Ram. Using introspective reasoning to select learning strategies. In *Proceedings of the First International Workshop on Multistrategy Learning*, 1991.

[94] M. Cox and A. Ram. Multistrategy learning with introspective meta-explanations. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 123–128, 1992.

[95] M. Cox and A. Ram. Choosing learning strategies to achieve learning goals. In *Proceedings of the AAAI Spring Symposium on Goal-Driven Learning*, 1994.

[96] R. Crites and A. Barto. An actor/critic algorithm that is equivalent to Q-learning. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

[97] R. Crites and A. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*. MIT Press, 1996.

[98] B. Dasarathy, editor. *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[99] B. Dasarathy. Minimal consistent set (MCS) identification for optimal nearest neighbor systems design. *IEEE transactions on systems, man and cybernetics*, **24**(3):511–517, 1994.

[100] D Dasgupta and D. McGregor. Evolving neurocontrollers for pole balancing. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer Verlag, Berlin, 1993.

[101] D. Dasgupta and D. McGregor. Genetically designing neuro-controllers for a dynamic system. In *Proceedings of the International Joint Conference on Neural Networks, Nagoya (Japan)*, pages 2869–2872. IEEE, 1993.

[102] Y Davidor. A naturally occurring niche and species phenomenon: The model and first results. In *Proceedings of the 1991 International Conference on Genetic Algorithms*, pages 257–263, 1991.

[103] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, **8**:341–362, 1992.

[104] P. Dayan and G. Hinton. Feudal reinforcement learning. In *Neural Information Processing Systems 5*, pages 271–278, 1993.

[105] P. Dayan and T. Sejnowski. TD(λ) converges with probability 1. *Machine Learning*, **14**:295–301, 1994.

[106] T. Dean, K. Basye, and J. Shewchuk. Reinforcement learning for planning and control. In S. Minton, editor, *Machine Learning Methods for Planning and Scheduling*. Morgan Kaufmann, 1992.

[107] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, **76**, 1995.

[108] K. DeJong. Genetic algorithm based learning. In Y. Kodratoff and R. Michalski, editors, *Machine learning, and Artificial Intelligence approach*, volume 3, pages 611–638. Morgan Kaufmann, San Mateo, CA, 1990.

[109] K. DeJong and A. Schultz. Using experience-based learning in game playing. In *Proceedings of the Fifth International Machine Learning Conference*, pages 284–290. Morgan Kaufmann, Publishers, 1988.

[110] K. Deng and A. Moore. Multiresolution instance-based learning. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, 1995.

[111] P. Devijver. On the editing rate of the multiedit algorithm. In *Pattern Recognition Letters*, volume 4, pages 9–12, 1986.

[112] P. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[113] B. Dike and R. Smith. Application of genetic algorithms to air combat maneuvering. Technical Report TCGA Report No. 93002, University of Alabama, Tuscaloosa, Alabama, 1993.

[114] D. Dobkin and S. Reiss. The complexity of linear programming. *Theoretical Computer Science*, **11**:1–18, 1980.

[115] M. Dorigo and M. Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, **71**(2):321–370, 1994.

[116] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimizing process. *IEEE Transactions on Systems, Man and Cybernetics*, 1994.

[117] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics*, **26**(1):1–13, 1996.

[118] M. Dorigo and U. Schnepf. Genetics-based machine learning and behavior based robotics: A new synthesis. *IEEE Transactions on Systems, Man, and Cybernetics*, **23**(1):141–154, 1993.

[119] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[120] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, **90**:161–175, 1990.

[121] M. Duff. Q-learning for bandit problems. Technical Report 95-26, Department of Computer Science, University of Massachusetts at Amherst, 1995.

[122] E. Durfee and V. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceedings of the 1987 International Joint Conference on Artificial Intelligence*, pages 875–883, 1987.

[123] I. Erev and A. Roth. On the need for low rationality, cognitive game theory: Reinforcement learning in experimental games with unique mixed strategy equilibria. Unpublished manuscript, August 1995.

[124] B. Everitt. *Cluster Analysis - 3rd Edition*. E. Arnold Press, London., 1993.

[125] S. Fortune and J. Hopcroft. A note on rabin's nearest neighbor algorithm. *Information processing Letters*, **8**(1):20–23, January 1979.

[126] A. Friedman. *Differential Games*. Wiley Interscience, New York, 1971.

[127] J. Friedman. Flexible metric nearest neighbor classification. Technical report, Department of Statistics, Stanford University, 1994.

[128] J. Friedman, F. Baskett, and L. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, pages 1000–1006, October 1975.

[129] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, **3**(3):209–226, September 1977.

[130] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, 1992.

[131] K. Fujimura. A model of reactive planning for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1503–1509. IEEE, 1991.

[132] K. Fukanaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[133] J. Gallagher and R. Beer. A qualitative dynamical analysis of evolved locomotion control. In H. Roitblat, J-A. Meyer, and S. Wilson, editors, *From Animals to Animats, Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB 92)*. The MIT Press, Cambridge, MA, 1992.

[134] E. Galperin and J. Skowronski. Pursuit-evasion differential games with uncertainties in dynamics. *Computers and Mathematics with Applications*, **13**(1–3):13–35, 1987.

[135] G. Game and C. James. The application of genetic algorithms to the optimal selection of parameter values in neural networks for attitude control systems. In *IEE Colloquium on 'High Accuracy Platform Control in Space'*, volume Digest No. 1993/148, pages 3/1–3/3. IEE, London, 1993.

[136] M. Garey and D. Johnson. *Computers and Intractability: a Guide to the theory of NP-Completeness*. Freeman and Co., San Francisco, CA, 1979.

[137] G. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, **18**:431–433, May 1972.

[138] M. Genesereth, M. Ginsberg, and J. Rosenschein. Cooperation without communication. In *Proceedings of the National Conference on Artificial Intelligence*, pages 51–57, San Jose CA, 1986. AAAI Press.

[139] M. Georgeff. A theory of action for multi-agent planning. In *Proceedings of National Conference on Artificial Intelligence*, pages 121–125, San Jose CA, 1984. AAAI Press.

[140] M. Gherrity. *A Game-Learning Machine*. PhD thesis, University of California, San Diego, 1993.

[141] D. Ghose and U. Prasad. Determination of rational strategies for players in two-target games. *Computers and Mathematics with Applications*, **26**(6):1–11, 1993.

[142] D. Goldberg. *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Machine Learning*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1983.

[143] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[144] C. Goldman and J. Rosenschein. Mutually supervised learning in multiagent systems. In *Proceedings of the ICJAI Workshop on Adaptation and Learning in Multiagent Systems*, 1996.

[145] S. Goldman and M. Kearns. On the complexity of teaching. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 303–314, Santa Cruz, CA, August 1991. Morgan Kaufmann.

[146] D. Gordon and D. Subramanian. A multistrategy learning scheme for agent knowledge acquisition. *Informatica*, **17**:331–346, 1993.

[147] D. Gordon and D. Subramanian. A multistrategy learning scheme for assimilating advice in embedded agents. In *Proceedings of the Second International Workshop on Multistrategy Learning*, pages 218–233. George Mason University, 1993.

[148] G. Gordon. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, Publishers, 1995.

[149] J. Grefenstette. Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, **3**:225–245, 1988.

[150] J. Grefenstette. Lamarkian learning in multi-agent environments. In *Proceedings of the Fourth International Conference of Genetic Algorithms*, pages 303–310. Morgan Kaufmann, 1991.

[151] J. Grefenstette. The evolution of strategies for multi-agent environments. *Adaptive Behavior*, **1**(1):65–89, 1992.

[152] J. Grefenstette and R. Daley. Methods for competitive and cooperative coevolution. In *Adaptation, Coevolution, and Learning in Multiagent Systems (ICMAS '95)*, pages 276–282. AAAI Press, 1995.

[153] J. Grefenstette, C. Ramsey, and A. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, **5**:355–381, 1990.

[154] N. Grigorenko. The problem of pursuit by several objects. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 71–80. Springer-Verlag, 1991.

[155] V Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, **3**:671–692, 1990.

[156] V. Gullapalli. *Reinforcement Learning and its Application to Control.* PhD thesis, University of Massachusetts, 1992.

[157] V Gullapalli. Learning control under extreme uncertainty. In *Neural Information Processing Systems 5*, pages 271–278, 1993.

[158] S. Gutman, M. Esh, and M. Gefen. Simple linear pursuit-evasion games. *Computers and Mathematics with Applications*, **13**(1–3):83–95, 1987.

[159] R. Hamalainen and H. Ehtamo, editors. *Differential Games—Developments in Modeling and Computation.* Springer-Verlag, New York, 1991.

[160] W. Hamilton. Instability and cycling of two competing hosts with two parasites. In S. Karlin and E. Nevo, editors, *Evolutionary Process Theory.* Academic Press, 1986.

[161] M. Harmon and L. Baird. Residual advantage learning applied to a differential game. In *Neural Information Processing Systems 7*, 1995.

[162] M. Harmon, L. Baird, and A. Klopf. Advantage updating applied to a differential game. In *Advances in Neural Information Processing Systems 7*, pages 353–360, 1994.

[163] M. Harmon, L. Baird, and A. Klopf. Reinforcement learning applied to a differential game. *Adaptive Behavior*, 1995. To appear.

[164] P. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, **14**(3):515–516, May 1968.

[165] T. Haynes and S. Sen. Evolving behavioral strategies in predators and prey. In S. Weiss and S. Sen, editors, *Adaptation and Learning in Multiagent Systems*, pages 113–126. Springer-Verlag, 1996.

[166] D. Heath. *A Geometric Framework for Machine Learning*. PhD thesis, Department of Computer Science, The Johns Hopkins University, 1992.

[167] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[168] J. Holland. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. *Machine Learning*, **2**:593–623, 1986.

[169] R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, **11**(1):63–90, 1993.

[170] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, MD, 1984.

[171] R. Huang. Systems control with the genetic algorithm and the nearest neighbor classification. *CC-AI*, **9**((2-3)):225–236, 1992.

[172] B. Huberman and N. Glance. Evolutionary games and computer simulations. *Proceedings of the National Academy of Sciences*, 1995. in press.

[173] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, **5**(1):15–17, 1976.

[174] Y. Ichikawa. Evolution of neural networks and application to motion control. In *Proceedings of the IEEE International Conference on Intelligent Motion Control*, pages 239–245. IEEE, 1990.

[175] F. Imado and T. Ishihara. Pursuit-evasion geometry analysis between two missiles and an aircraft. *Computers and Mathematics with Applications*, **26**(3):125–139, 1993.

[176] R. Isaacs. Differential games: A mathematical theory with applications to warfare and other topics. Technical Report Research Contribution No. 1, Center for Naval Analysis, Washington, D.C., 1963.

[177] R. Isaacs. *Differential Games*. Robert E. Krieger, New York, 1975.

[178] C. Isbell. Explorations of the practical issues of learning prediction-control tasks using temporal difference learning methods. Master's thesis, Massachusetts Institute of Technology, 1992.

[179] T. Jaakkola, M. Jordan, and S. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 1994.

[180] T. Jaakkola, S. Singh, and M Jordan. Reinforcement learning algorithm for partially observable markovian decision problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352. The MIT Press, 1995.

[181] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural Computation*, **3**:79–87, 1991.

[182] M. James. *Classification Algorithms*. Wiley-Interscience Publications, 1985.

[183] B. Järmark. On closed-loop controls in pursuit-evasion. *Computers and Mathematics with Applications*, **13**(1–3):157–166, 1987.

[184] T. Jervis and F. Fallside. Pole balancing on a real rig using a reinforcement learning controller. Technical Report CUED/F-INFENG/TR 115, Cambridge University Engineering Department, 1992.

[185] S. Johnson. Hierarchical clustering schemes. *Psychometrika*, **32**(3), September 1967.

[186] M. Jordan and R. Jacobs. Hierarchical mixtures of experts and the EM algorithm. Technical Report AI Memo 1440, Massachusetts Institute of Technology, 1993.

[187] M. Jordan and L. Xu. Convergence results for the EM approach to mixtures of experts architectures. Technical Report AI Memo 1458, Massachusetts Institute of Technology, 1993.

[188] L. Kaelbling. Associative reinforcement learning: A generate and test algorithm. *Machine Learning*, **15**(3), 1994.

[189] D. Kibler and D. Aha. Comparing instance-averaging with instance-filtering learning algorithms. In *Proceedings of the Third European Working Session on Learning*, pages 68–80, 1988.

[190] D. Kibler and D. Aha. Comparing instance-saving with instance-averaging learning algorithms. In D.P.Benjamin, editor, *Change of Representation and Inductive Bias*. Kluwer Academic Publisher, Norwell, MA, 1989.

[191] H. Kimura, M. Yamamura, and S. Kobayashi. Reinforcement learning by stochastic hill climbing on discounted reward. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 295–303, 1995.

[192] K. Kira and L. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 249–256, Aberdeen, Scotland, 1992. Morgan Kaufmann.

[193] S. Koenig and R. Simmons. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Technical Report CMU-CS-93-106, Carnegie Mellon University, 1992.

[194] D. Koller and N. Megiddo. Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory*, **25**:73–92, 1996.

[195] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on the Theory of Computing*, pages 750–759, 1994.

[196] D. Koller, N. Megiddo, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 1995. To appear.

[197] D. Koller and A. Pfeffer. Generating and solving imperfect information games. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, 1995.

[198] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.

[199] H. Kuhn. Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematical Studies*, pages 193–216. Princeton University Press, 1953.

[200] H. Lai and K. Tanaka. An n-person noncooperative discounted vector valued dynamic game with a stopped set. *Computers and Mathematics with Applications*, **13**(1–3):227–237, 1987.

[201] P. Langley and S. Sage. Scaling to domains with many irrelevant features. Learning Systems Department, Siemens Corporate Research, Princeton, NJ, 1993.

[202] V. Laporte, J. Nicolas, and P. Bernhard. About the resolution of discrete pursuit games and its application to naval warfare. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 151–163. Springer-Verlag, 1991.

[203] A. Levchenkov, A. Pashkov, and S. Terekhov. A construction of the value function in some differential games of approach with two pursuers and one evader. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 38–47. Springer-Verlag, 1991.

[204] R. Levinson. General game-playing and reinforcement learning. Technical Report UCSC-CRL-95-06, Department of Computer Science, University of California, Santa Cruz, May 1995.

[205] J. Lewin. *Differential Games*. Springer-Verlag, New York, 1994.

[206] M. Lewis, A. Fagg, and A. Solidum. Genetic programming approach to the construction of a neural network for control of a walking robot. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, volume 3, pages 2618–2623. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[207] J. Lin and J. Vitter. A theory for memory-based learning. Technical Report CS-92-53, Brown University, 1992.

[208] L. Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 781–786, 1991.

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

[210] L. Lin and T. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science, 1992.

[211] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Machine Learning Conference*, pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann.

[212] M. Littman. Memoryless policies: Theoretical limitations and practical results. In *Proceedings of the Third International Conference on the Simulation of Adaptive Behavior*, 1994.

[213] M. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, Department of Computer Science, 1996.

[214] M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. Technical Report CMU-CS-93-165, School of Computer Science, Carnegie-Mellon University, 1993.

[215] M. Littman, A. Cassandra, and L. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Machine Learning Conference*. Morgan Kaufmann, Publishers, 1995.

[216] M. Littman, T. Dean, and L. Kaelbling. On the complexity of solving markov decision problems. In *Proceedings of the Eleventh International Machine Learning Conference*. Morgan Kaufmann, Publishers, 1994.

[217] R. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey.* John Wiley and Sons, 1957.

[218] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, **22**:251–282, 1996.

[219] S. Mahadevan. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 768–773, 1991.

[220] S. Mahadevan. To discount or not to discount in reinforcement learning: A case study comparing R learning and Q learning. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, Publishers, 1994.

[221] S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 1996. To appear.

[222] K. Markey. Efficient learning of multiple degree-of-freedom control problems with quasi-independent Q-agents. In *Proceedings of 1993 Connectionist Models Summer School*, 1993.

[223] J. Maynard Smith. *Evolution and the Theory of Games.* Cambridge University Press, 1982.

[224] R. McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, 1993.

[225] R. McCallum. First results with instance-based state identification for reinforcement learning. Technical Report TR 502, University of Rochester, 1994.

[226] R. McCallum. Reduced training time for reinforcement learning with hidden state. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

[227] R. McCallum. Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems 7*, pages 377–384, 1995.

[228] R. McCallum. Instance-based utile distinction for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[229] D. McGregor, M. Odetayo, and D Dasgupta. Adaptive-control of a dynamic system using genetic-based methods. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 521–525. IEEE, 1992.

[230] A. Merz. Stochastic guidance laws in satellite pursuit-evasion. *Computers and Mathematics with Applications*, **13**(1–3):151–156, 1987.

[231] S. Mikami, H. Tano, and Y. Kakazu. An autonomous legged robot that learns to walk through simulated evolution. In *Self-organization and life, from simple rules to global complexity, Proceedings of the Second European Conference on Artificial Life*, pages 758–767. MIT Press, Cambridge, 1993.

[232] J. Millan and C. Torras. A reinforcement connectionist approach to robot path finding in non-maze-like environments. *Machine Learning*, **8**:363–395, 1992.

[233] G. Miller and D. Cliff. Co-evolution of pursuit and evasion I: Biological and game-theoretic foundations. Technical Report CSRP311, School of Cognitive and Computing Sciences, University of Sussex, August 1994.

[234] J. Miller. The coevolution of automata in the repeated prisoner's dilemma. Technical Report No. 8903, Santa Fe Institute, 1989.

[235] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, **4**(2):227–243, 1989.

[236] J. Mingers. An empirical comparison of selection measures for decision tree induction. *Machine Learning*, **3**:319–342, 1989.

[237] A Moore. *Efficient Memory-Based Learning for Robot Control.* PhD thesis, Computer Laboratory, Cambridge University, 1990.

[238] A. Moore. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In *Neural Information Processing Systems 6*, 1994.

[239] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, **13**:103–130, 1993.

[240] A. Moore and C. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 1995. To appear.

[241] A. Moore, C. Atkeson, and S. Schaal. Memory-based learning for control. Technical Report CMU-RI-TR-95-18, Carnegie-Mellon University, April 1995.

[242] A. Moore and J. Schneider. Memory-based stochastic optimization. In *Neural Information Processing Systems 7*, 1995.

[243] Y. Mor, C. Goldman, and J. Rosenschein. Learn your opponent's strategy (in polynomial time)! In *Proceedings of the IJCAI Workshop on Adaptation and Learning in Multagent Systems*, 1995.

[244] D. Moriarty and R. Miikkulainen. Learning sequential decision tasks. Technical Report AI95-229, Department of Computer Science, The University of Texas at Austin, January 1995.

[245] K. Moritz, R. Polis, and K. Well. Pursuit-evasion in medium-range air-combat scenarios. *Computers and Mathematics with Applications*, **13**(1–3):167–180, 1987.

[246] P. Morris. *Introduction to Game Theory.* Springer-Verlag, 1994.

[247] S. Murthy. *On Growing Better Decision Trees from Data.* PhD thesis, Department of Computer Science, The Johns Hopkins University, 1995.

[248] D. Mutchler. The multi-player version of minimax displays game pathology. *Artificial Intelligence*, **64**(2):323–336, December 1993.

[249] T. Nagao, T. Agui, and H. Nagahashi. A genetic method for optimization of asynchronous random neural networks and its application to action control. In *Proceedings of the 1993 International Joint Conference on Neural Networks, Nagoya (Japan)*, pages 2869–2872. IEEE, 1993.

[250] J. Nash. Non-cooperative games. *Annals of Mathematics*, **54**(2):286–295, 1951.

[251] D. Nau. Decision quality as a function of search depth on game trees. *Journal of the Association of Computing Machinery*, **30**(4):687–708, October 1983.

[252] D. Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 357–363, 1989.

[253] N. Nilsson. Teloe-reactive programs for agent control. *Journal of Artificial Intelligence Research*, **1**:139–158, 1994.

[254] G. Owen. *Game Theory.* Academic Press, 1982.

[255] M. Pachter. Simple-motion pursuit-evasion in the half plane. *Computers and Mathematics with Applications*, **13**(1–3):69–82, 1987.

[256] M. Pachter and T. Miloh. The geometric approach to the construction of the barrier surface in differential games. *Computers and Mathematics with Applications*, **13**(1–3):47–67, 1987.

[257] L. Paletta. Temporal difference learning in RISK-like environments. Master's thesis, Technical University Graz and The Johns Hopkins University, 1996.

[258] B. Pell. Exploratory learning in the game of GO. Technical Report TR 275, University of Cambridge, Computer Laboratory, 1992.

[259] B. Pell. Metagame: A new challenge for games and learning. Technical Report TR 276, University of Cambridge, Computer Laboratory, 1992.

[260] B. Pell. *Strategy Generation and Evaluation for Meta-Game Playing.* PhD thesis, University of Cambridge, Cambridge, England, 1993.

[261] B. Pell. A strategic metagame player for general chess-like games. In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

[262] M. Pendrith. On reinforcement learning of control actions in noise and non-markovian domains. Technical Report UNSW-CSE-TR-9410, The University of New South Wales, 1994.

[263] J. Peng. *Efficient Dynamic Programming-Based Learning for Control.* PhD thesis, Northeastern University, 1993.

[264] J. Peng. Efficient memory-based dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[265] J. Peng and R. Williams. Incremental multi-step Q-learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

[266] M. Potter, K. De Jong, and J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In *Proceedings of the International Conference on Genetic Algorithms*, pages 366–372, 1995.

[267] U. Prasad and N. Rajan. Aircraft pursuit-evasion problems with variable speeds. *Computers and Mathematics with Applications*, **13**(1–3):111–121, 1987.

[268] T. Prescott and J. Mayhew. Obstacle avoidance through reinforcement learning. In *Advances in Neural Information Processing Systems 4*, pages 523–530, San Mateo, CA, 1992. Morgan Kaufmann.

[269] M. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming.* John Wiley and Sons, New York, 1994.

[270] J. Quinlan. Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, San Mateo, CA, 1983.

[271] J. Quinlan. Induction of decision trees. *Machine Learning*, **1**:81–106, 1986.

[272] J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, Publishers, San Mateo, California, 1993.

[273] N. Rajan, U. Prasad, and N. Rao. Pursuit-evasion of two aircraft in a horizontal plane. *Journal of Guidance and Control*, **3**(3):261–267, 1980.

[274] A. Ram, R. Arkin, G. Boone, and M. Pearce. Using genetic algorithms to learn reactive control parameters for autonomous robot navigation. *Adaptive Behavior*, **2**(3), 1994.

[275] A. Ram and J. Santamaria. Continuous case-based reasoning. In *Proceedings of the AAAI Workshop on Case-Based Reasoning*, 1993.

[276] A. Ram and J. Santamaria. Multistrategy learning in reactive control systems for autonomous robot navigation. *Informatica*, **17**(4):347–369, 1993.

[277] A. Ram and J. Santamaria. Introspective reasoning using meta-explanations for multistrategy learning. *Machine Learning: A Multistrategy Approach*, **4**, 1994.

[278] C. Ramsey and J. Grefenstette. Case-based anytime learning. In D. Aha, editor, *Case Based Reasoning: Papers from the 1994 Workshop*, pages 91–95, Menlo Park, California, 1994. AAAI Press.

[279] C. Reynolds. Competition, coevolution, and the game of tag. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 59–69, Cambridge, Massachusetts, 1994. MIT Press.

[280] M. Ring. *Continual Learning in Reinforcement Environments*. PhD thesis, Department of Computer Science, The University of Texas at Austin, 1994.

[281] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, **21**(6):665–669, 1975.

[282] E. Rodin, Y. Lirov, S. Mittnik, B. McElhaney, and L. Wilbur. Artificial intelligence in air combat games. *Computers and Mathematics with Applications*, **13**(1–3):261–274, 1987.

[283] J. Rosenschein and M. Genesereth. Deals among rational agents. In *Proceedings of the 1985 International Joint Conference on Artificial Intelligence*, pages 91–99, 1985.

[284] S. Rosenschein and L. Kaelbling. A situated view of representation and control. *Artificial Intelligence*, **73**, 1995.

[285] C. Rosin and R. Belew. A competitive approach to game learning. In *Proceedings of the Ninth Annual ACM Conference on Computational Learning Theory*, 1996.

[286] A. Roth and I. Erev. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, **8**:164–212, 1995.

[287] S. Russel and D. Subramanian. Probably bounded-optimal agents. *Journal of Artificial Intelligence Research*, **2**:575–609, 1995.

[288] S. Safra and M. Tennenholtz. On planning while learning. *Journal of Artificial Intelligence Research*, **2**:111–129, 1994.

[289] M. Salganicoff. Density-adaptive learning and forgetting. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 276–283. Morgan Kaufmann, 1993.

[290] M. Salganicoff. Improved learning of time-varying mappings with performance-error based forgetting. Technical Report MS-CIS-93-80, University of Pennsylvania, 1993.

[291] M. Salganicoff and L. Ungar. Active exploration and learning in real-valued spaces using multi-armed bandit allocation indices. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[292] S. Salzberg. *Learning with Nested Generalized Exemplars*. Kluwer Academic Publishers, Norwell, MA, 1990.

[293] S. Salzberg. Distance metrics for instance-based learning. In *Methodologies for Intelligent Systems: 6th International Symposium*, pages 399–408, 1991.

[294] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, **6**:251–276, 1991.

[295] S. Salzberg. Combining learning and search to create good classifiers. Technical Report JHU-92/12, Johns Hopkins University, Baltimore MD, 1992.

[296] S. Salzberg, A. Delcher, D. Heath, and S. Kasif. Learning with a helpful teacher. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 705–711, Sydney, Australia, August 1991. Morgan Kaufmann.

[297] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**(3):211–229, 1959.

[298] A. Samuel. Some studies in machine learning using the game of checkers II–recent progress. *IBM Journal of Research and Development*, **11**(6):601–617, 1967.

[299] T. Sandholm and R. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, **37**:147–166, 1995.

[300] L. Saul and S. Singh. Markov decision processes in large state spaces. In *Proceedings of COLT '95*. Morgan Kaufmann, Publishers, 1995.

[301] S. Schaal. Nonparametric regression for learning. In *Proceedings of the Conference on Prerational Intelligence–Adaptive and Learning Behavior*, 1994.

[302] A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, **2**:475–500, 1995.

[303] J. Schmidhuber. A general method for incremental self-improvement and multi-agent learning in unrestricted environments. *Evolutionary Computation: Theory and Applications*, 1996. To appear.

[304] J. Sheppard and S. Salzberg. Memory-based learning of pursuit games. Technical Report JHU-94-02, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, January 1993. revised May, 1995.

[305] J. Sheppard and S. Salzberg. Bootstrapping memory-based learning with genetic algorithms. In *1994 Workshop on Case Based Reasoning*. AAAI, August 1994.

[306] J. Sheppard and S. Salzberg. Combining memory based reasoning with genetic algorithms. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 452–459. Morgan Kaufmann, 1995.

[307] J. Sheppard and S. Salzberg. A teaching method for memory-based control. *Artificial Intelligence Review*, 1997. To appear.

[308] T. Shibata, T. Fukada, and K. Tanie. Fuzzy critic for robotic motion planning by genetic algorithm in hierarchical intelligent control. In *Proceedings of the 1993 International Joint Conference on Neural Networks, Nagoya (Japan)*, pages 770–773. IEEE, 1993.

[309] T. Shibata, T. Fukada, and K. Tanie. Nonlinear backlash compensation using recurrent neural network - unsupervised learning by genetic algorithm. In *Proceedings of the 1993 International Joint Conference on Neural Networks, Nagoya (Japan)*, pages 742–745. IEEE, 1993.

[310] T. Shibata, T. Fukada, and K. Tanie. Synthesis of fuzzy artificial intelligence, neural networks, and genetic algorithms for hierarchical intelligent control. In *Proceedings of the 1993 International Joint Conference on Neural Networks, Nagoya (Japan)*, pages 2869–2872. IEEE, 1993.

[311] J. Shinar and A. Davidovitz. A two-target game analysis in line-of-sight coordinates. *Computers and Mathematics with Applications*, **13**(1–3):123–140, 1987.

[312] J. Simons, H. van Brussel, J. DeSchutter, and J. Verhaert. A self-learning automaton with variable resolution for high precision assembly by industrial robots. *IEEE Transactions on Automatic Control*, **27**(5):1109–1113, October 1982.

[313] W. Simpson. The sensitivity of air combat maneuvering engagements to the initial conditions in the near-neighborhood of a neutral start. Technical Report 78-0276, Center for Naval Analyses, Arlington, Virginia, 1978.

[314] S. Singh. The efficient learning of multiple task sequences. In *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA, 1992. Morgan Kaufmann.

[315] S. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, **8**:323–340, 1992.

[316] S. Singh. *Learning to Solve Markovian Decision Processes.* PhD thesis, University of Massachusetts, 1994.

[317] S. Singh. Reinforcement learning algorithms for average-payoff markovian decision processes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 700–706. AAAI, 1994.

[318] S. Singh, T. Jaakola, and M. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning.* Morgan Kaufmann, Publishers, 1994.

[319] S. Singh, T. Jaakola, and M. Jordan. Reinforcement learning with soft state aggregation. In *Neural Information Processing Systems 7*, 1995.

[320] S. Singh and R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 1996. To appear.

[321] D. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Machine Learning Conference*, pages 293–301, New Brunswick, NJ, 1994. Morgan Kaufman.

[322] J. Skowronski and R. Stonier. The barrier in a pursuit-evasion game with two targets. *Computers and Mathematics with Applications*, **13**(1–3):37–45, 1987.

[323] R. Smith and B. Gray. Co-adaptive genetic algorithms: An example in othello strategy. Technical Report TCGA Report No. 94002, University of Alabama, Tuscaloosa, Alabama, 1993.

[324] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, **29**(12):1213–1228, 1986.

[325] E. Stanley, D. Ashlock, and L. Tesfatsion. Iterated prisoner's dilemma with choice and refusal of partners. In *Proceedings of ALife III.* Sante Fe Institute, 1993.

[326] B. Steer and M. Larcombe. A goal seeking and obstacle avoiding algorithm for autonomous mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1518–1528. IEEE, 1991.

[327] P. Stone and M. Veloso. Beating a defender in robotic soccer: Memory-based learning of a continuous function. In *Proceedings of Neural Information Processing Systems*, 1995.

[328] P. Stone and M. Veloso. Broad learning from narrow training: A case study in robotic soccer. Technical Report CMU-CS-95-207, Carnegie Mellon University, School of Computer Science, 1995.

[329] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *IEEE Transactions on Knowledge and Data Engineering*, 1996. submitted.

[330] P. Stone and M. Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. In *1996 AAAI Spring Symposium on Adaptation, Co-Evolution, and Learning in Multiagent Systems*, 1996. submitted.

[331] T. Suguwara and V. Lesser. On-line learning of coordination plans. Technical Report COINS TR 93-27, University of Massachusetts, 1993.

[332] R. Sutton. Learning to predict by methods of temporal differences. *Machine Learning*, **3**:9–44, 1988.

[333] R. Sutton. TD models: Modeling the world at a mixture of time scales. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, Publishers, 1995.

[334] R. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing 8*. MIT Press, 1996.

[335] C. Swonger. Sample set condensation for a condensed nearest neighbor decision rule for pattern recognition. In S. Watanabe, editor, *Frontiers of Pattern Recognition*, pages 511–519. Academic Press, 1972.

[336] M. Tambe. Recursive agent and agent-group tracking in a real-time, dynamic environment. In *Adaptation, Coevolution, and learning in Multiagent Systems (ICMAS '95)*. AAAI Press, 1995.

[337] M. Tambe. Teamwork in real-world, dynamic environments. In *1996 International Conference on Multiagent Systems*. AAAI Press, 1996.

[338] M. Tambe. Tracking dynamic team activity. In *Proceedings of the 13th National Conference on Artificial Intelligence*. AAAI Press, 1996.

[339] M. Tambe, L. Johnson, and W. Shen. Adaptive agent tracking in real-world multi-agent domains: A preliminary report. *International Journal of Human Computer Studies*, 1996. To appear.

[340] M. Tambe and P. Rosenbloom. Architectures for agents that track other agents in multi-agent worlds. *Intelligent Agents: Lecture Notes in Artificial Intelligence*, **II**(1037), 1996.

[341] M. Tan. Multi-agent reinforcement learning: independent vs. cooperative agents. In *Machine Learning: Proceedings of the Tenth International Conference*, San Mateo, CA, 1993. Morgan Kaufmann.

[342] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, **8**:257–277, 1992.

[343] G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, pages 58–67, March 1995.

[344] G. Tesauro and T. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, **39**:357–390, 1989.

[345] S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, 1992.

[346] S. Thrun and A. Schwartz. Issues in using function approximation in reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*. Lawrence Erlbaum Publisher, 1993.

[347] S. Thrun and S. Schwartz. Finding structure in reinforcement learning. In *Advanced in Neural Information Processing Systems 7*, pages 385–392. The MIT Press, 1995.

[348] B. Tolwinski. Solving dynamic games via markov game approximations. In R. Hamalainen and H. Ehtamo, editors, *Differential Games—Developments in Modeling and Computation*, pages 265–274. Springer-Verlag, 1991.

[349] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-6**(6):448–452, June 1976.

[350] J. Tsitsiklis and B. van Roy. An analysis of temporal difference learning with function approximation. Technical Report LIDS-P-2322, Massachusetts Institute of Technology, 1994.

[351] P. Turney. Theoretical analysis of cross-validation error and voting in instance-based learning. *Journal of Experimental and Theoretical Artificial Intelligence*, 1993. submitted.

[352] P. Turney. A theory of cross-validation error. *Journal of Experimental and Theoretical Artificial Intelligence*, 1993. submitted.

[353] L. Valiant. A theory of the learnable. *Communications of the ACM*, **27**:1134–1142, 1984.

[354] L. VALIANT. Learning disjunctions of conjunctions. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 560–566, Los Altos, California, 1985. Morgan Kaufmann.

[355] J. VAN DER WAL. *Stochastic Dynamic Programming.* Morgan Kaufmann, Amsterdam, 1981.

[356] M. VELOSO AND P. STONE. FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, **3**:25–52, 1995.

[357] D. VOLPER AND S. HAMPSON. Learning and using specific instances. *Biological Cybernetics*, **57**:57–71, 1987.

[358] J. VON NEUMANN AND O. MORGENSTERN. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, New Jersey, 1947.

[359] B. VON STENGEL AND D. KOLLER. Minmax equilibria in team games. *Games and Economic Behavior*, 1996. To appear.

[360] N. VORB'EV. *Game Theory, Lectures for Economists and Systems Scientists.* Springer-Verlag, 1977.

[361] T. WAGNER. Convergence of the edited nearest neighbor. *IEEE Transactions on Information Theory*, **19**:696–697, September 1973. Correspondence.

[362] J WANG. *The Theory of Games.* Oxford Science Publications, 1988.

[363] C. WATKINS. *Learning with Delayed Rewards.* PhD thesis, Cambridge University, Department of Computer Science, Cambridge, England, 1989.

[364] C. WATKINS AND P. DAYAN. Q-learning. *Machine Learning*, **8**:279–292, 1992.

[365] G. WERNER AND M. DYER. Evolution of communication in artificial organisms. In *Artificial life II*, pages 659–687. Addison-Wesley, 1991.

[366] D. WETTSCHERECK. *A Study of Distance-Based Machine Learning Algorithms.* PhD thesis, Oregon State University, 1994.

[367] D. WETTSCHERECK, D. AHA, AND T. MOHRI. A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *Artificial Intelligence Review*, 1996. To appear.

[368] S. WHITEHEAD. *Reinforcement Learning for the Adaptive Control of Perception and Action.* PhD thesis, Department of Computer Science, University of Rochester, 1992.

[369] D. Whitley, S. Dominic, and R. Das. Genetic reinforcement learning with multilayer neural networks. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1991.

[370] B. Widrow. The original adaptive neural net broom-balancer. In *International Symposium on Circuits and Systems*, pages 351–357, 1987.

[371] R. Williams and L. Baird. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. Technical Report NU-CCS-93-11, Northeastern University, 1993.

[372] R. Williams and L. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University, 1993.

[373] D. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, **2**(3):408–421, July 1972.

[374] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 1996. submitted.

[375] Y. Yavin. Pursuit-evasion differential games with deception or interrupted observation. *Computers and Mathematics with Applications*, **13**(1–3):191–203, 1987.

[376] Y. Yavin. A stochastic two-target pursuit-evasion differential game with three players moving in a plane. *Computers and Mathematics with Applications*, **13**(1–3):141–149, 1987.

[377] J. Zhang. Selecting typical instances in instance-based learning. In *Proceedings of the Ninth International Machine Learning Conferences*, pages 470–479, Aberdeen, Scotland, 1992. Morgan Kaufman.

[378] J. Zhao and J. Schmidhuber. Incremental self-improvement for life-time multi-agent reinforcement learning. In *Proceedings of SAB '96*, 1996. To appear.

Vita

John Wilbur Sheppard was born in Pittsburgh, Pennsylvania on 21 August 1961. He graduated, salutatorian, from Hickory Senior High School in Hermitage, Pennsylvania in 1979. After studying at Allegheny College, the Pennsylvania State University, and Thiel College, he graduated, *magna cum laude* from Southern Methodist University in 1983 with a Bachelor of Science degree in Computer Science. He then attended the Lutheran Theological Seminary in Gettysburg, Pennsylvania where he studied for a Master of Divinity degree. In 1989, he received his Master of Science in Computer Science from the Johns Hopkins University while working full time at ARINC Research Corporation in Annapolis, Maryland. During his time working on the masters, he married Justina Anne Pape in 1988. In 1990, he began his doctoral program at Johns Hopkins, still employed by ARINC. In 1992, he became a father with the birth of his son, Jesse Carl on 23 August.