A Framework for Aggregation of Multiple Reinforcement Learning Algorithms

by

Ju Jiang

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2007

Abstract

Aggregation of multiple Reinforcement Learning (RL) algorithms is a new and effective technique to improve the quality of Sequential Decision Making (SDM). SDM is very common and important in various realistic applications, especially in automatic control problems. The quality of a SDM depends on (discounted) long-term rewards rather than the instant rewards. Due to delayed feedback, SDM tasks are much more difficult to handle than classification problems. Meanwhile, in many SDM tasks, the feedback about a decision is often in the form of evaluation rather than instruction. Therefore, supervised learning techniques are not suitable in these tasks. To tackle these difficulties, RL methods are investigated.

Although many RL algorithms have been developed, none is consistently better than the others. In addition, the parameters of RL algorithms significantly influence learning performances. Successful RL applications depend on suitable learning algorithms and elaborately selected learning parameters, but there is no universal rule to guide the choice of algorithms and the setting of parameters.

To handle this difficulty, a new multiple RL system - the Aggregated Multiple Reinforcement Learning System (AMRLS) is developed. In this proposed system, each RL algorithm (learner) learns individually in a learning module and provides its output to an intelligent aggregation module. The aggregation module dynamically aggregates these outputs by using some intelligent aggregation methods and provides a decision of action. Then, all learners take the action and update their policies individually. The two processes are performed alternatively in each learning episode. Because of the intelligent and dynamic aggregation, AMRLS has the ability to deal with dynamic learning problems without the need to search for the optimal learning algorithm or the optimal values of learning parameters. It is claimed that several complementary learning algorithms can be integrated in the AMRLS to improve the learning performance in terms of success rate, robustness, confidence, redundance, and complementariness.

There are two strategies for learning an optimal policy by using RL methods. One is based on the Value Function Learning (VFL) strategy, which learns an optimal policy expressed as a value function. The Temporal Difference (TD) methods are examples of this strategy and they are called TDRL in this dissertation. The other strategy is based on the Direct Policy Search (DPS), which directly searches for the optimal policy in the potential policy space. The Genetic Algorithms (GAs)-based search algorithms are instances of this strategy and they are named GARL. Both of the strategies exhibit advantages and disadvantages. A hybrid learning architecture of GARL and TDRL, HGATDRL, is proposed to combine them together. HGATDRL uses an off-line GARL approach to learn an initial policy first, and then updates the policy on-line by using a TDRL approach. This new learning method enhances the learning ability of RL learners in AMRLS.

The AMRLS framework and HGATDRL method are tested on several SDM problems, including the maze world problem, pursuit domain problem, cart-pole balancing system, mountain car problem, and flight control system. The experimental results show that the proposed framework and method can enhance the learning ability and improve learning performance of a multiple RL system.

Acknowledgments

I wish to express sincere thanks to all the people who have provided invaluable help and support in my thesis research.

I would like to thank my supervisor, Professor Mohamed S. Kamel. He has generously helped me in my studying and research. I have learned much from him both in knowledge and in research attitude. Without his encouragement, guidance, and support, I would not have accomplished my thesis research. I owe a great deal of gratitude to him.

I also want to express my appreciation to Professor Mohamed A. Zohdy of Oakland University for serving as my external examiner. Special thanks are due to Professor Jiahua Chen, Professor Otman Basir, and Professor Hamid R. Tizhoosh for serving in the exam committee. Their comments and suggestions have greatly improved my work.

I am greatly indebted to my parents, my wife, Haiyan Xu, and my daughter, Yangzi Jiang. They have encouraged me, believed in me, helped me, and supported me. Without their love and support, I would not have been able to complete my work.

Dedication

This is dedicated to Ms. Haiyan Xu, my beloved wife.

Contents

List of Tables

List of Figures

AI	Artificial Intelligence
AMRLS	Aggregated Multiple Reinforcement Learning System
CMAC	Cerebellar Model Articulation Controller
DPS	Direct Policy Search
FA	Function Approximation
GARL	Genetic Algorithms-based Reinforcement Learning
GAs	Genetic Algorithms
HGATDRL	Hybrid architecture of GARL and TDRL
MASs	Multiagent Systems
MCSs	Multiple Classifier Systems
MDP	Markov Decision Process
RL	Reinforcement Learning
SARSA	State Action Reward State Action
SDM	Sequential Decision Making
TD	Temporal Difference
TDRL	TD-based Reinforcement Learning
VFL	Value Function Learning

Chapter 1

Introduction

Learning is a principal characteristic of an intelligent agent because learning makes the agent establish its knowledge by itself rather than relying on the designers. Developing new learning methods and improving existing learning methods by integrating several techniques are the goals of machine learning research. This chapter describes the motivation and methods for improving the learning abilities of Reinforcement Learning (RL) algorithms by integrating aggregation technique with RL.

1.1 Preface

Reinforcement Learning (RL), a type of machine learning, is an efficient method to make agents learn. Typically, all RL methods are described as procedures for learning a policy based on numerical reinforcement (rewards or punishments) obtained through trial-and-error interactions with environments. In RL, an environment is defined as all the objects outside an agent that is performing the learning, but interacting with the agent. Usually, an environment contains a lot of information, such as the current state of the agent, the state transitions of the agent when it executes an action in a state, the instant reward, and the conditions of the learning procedure. The goal of RL is to learn an optimal policy that makes sequential decisions to maximize a long term discounted reward. Here, a policy is defined as a mapping from a state set to an action set of a practical problem. In this dissertation, an RL agent is also called a learner or RL algorithm.

RL learns a policy by interacting with an environment without the need to know the dynamic model of the environment that is taken under consideration. Furthermore, RL does not need the precise target values for inputs to establish training data. For many complicated Sequential Decision Making (SDM) tasks in which neither the model of the environment nor desired actions are clearly described, RL technique is an efficient method for training an agent to perform the tasks with good qualities. As a result, RL is often applied to situations in which the knowledge of the environments is either insufficient or too costly to obtain.

Two different strategies, Value Function Learning (VFL) [45, 84, 92] and Direct Policy Search (DPS) [11, 60, 96], have been proposed to learn an optimal policy using reinforcement information. Although most papers about RL adopt the VFL strategy to learn an optimal policy, in this dissertation both VFL and DPS strategies are discussed and combined to form a new hybrid RL architecture. For convenience, sometimes, RL is used to represent the VFL-based RL algorithm in this dissertation.

Temporal Difference (TD)-learning method is a well-known and efficient RL method based on VFL strategy. A value function is designed to express the expected discounted accumulated reward of a state or state-action pair, therefore, an optimal policy can be easily obtained from the value function. By interacting with an environment, TD methods receive the instant reward, update the value function based on the TD-error, a combination of the instant reward and estimated value of the next state, and construct a policy without needing the dynamic model of the environment. There is a family of TD-learning algorithms, Actor Critic (AC) method [9], Q-learning [92], SARSA(λ)-learning [84], to name a few. These RL algorithms are called TD-based RL (TDRL) algorithms, which are the main RL algorithms used in this dissertation, and will be explained in Chapter 3.

For DPS, a policy is represented by a set of parameters that are optimized by searching for the policy space directly, and a structure that combines the parameters. There are two key elements for DPS. One is the method employed to parameterize a policy, i.e., determining the explicit representation of a policy. The other is the strategy used for optimizing the parameters. Various optimization methods, including value iteration [84], simulated annealing [20], and evolutionary algorithms [59] have been proposed to optimize the parameters. Among these methods, the evolutionary algorithms are the most widely applied [47, 59, 60, 64, 86], while Genetic Algorithms (GAs), a type of evolutionary algorithms, are particularly promising [62]. GA-based RL approach updates the fitness function of each policy with reinforcement information and searches for the optimal policy using evolution operators such as mutation and crossover. This GA-based RL method, abbreviated as GARL, is discussed in detail in Chapter 4.

1.2 Motivation

A number of RL algorithms, such as Dyna method [4], Real-time Dynamic Programming [10], Actor Critic (AC) [9], SARSA-learning [84], and Q-learning [92], have been developed. These algorithms are based on different theories and learning methods and have been successfully applied to various tasks, such as the elevator dispatching [18], dynamic channel allocation [77], robotic soccer game [69], multi-link robot [14, 15], mobile robot [25, 45] and helicopter flight control [63]. However, there is no algorithm that is always superior to other algorithms in handling all kinds of tasks. Different algorithm possesses diverse advantages and disadvantages. Empirical results and specific applications confirm that a given learning algorithm can outperform other algorithms for a certain task or in a certain environment, but it is impossible to develop one algorithm to achieve the best results in all the problem domains. One algorithm that performs very well in a few specific applications may present unacceptable performances in other applications. There is no universal methodology to guide the choice of learning algorithms.

Even if an algorithm is deliberately selected based on experiments or experience, the learning performance is significantly influenced by the values of the learning parameters. There are several important learning parameters, such as the learning rate, α, eligibility traces factor, λ, and discount factor, γ, in TD-based RL algorithms, and the population size, chromosome length, and crossover rate in GA-based RL algorithms. It may take a long time and large computation to find the optimal values for the parameters. Meanwhile, the optimal values may lack in the adaptability to a dynamic environment.

Aggregation of different learning algorithms provides an approach for combining multiple algorithms to eliminate the weaknesses of them, avoid searching for the optimal learning algorithm or the optimal values of the learning parameters, and make use of information more efficiently. A new learning system, the Aggregated Multiple Reinforcement Learning System (AMRLS), is proposed to aggregate multiple RL algorithms to improve learning performances. In the AMRLS, a group of RL algorithms learn to perform a common task. To coordinate these algorithms and make them work more efficiently, aggregation techniques are introduced to the system to perform dynamic aggregation.

Although there are convincing evidences that an improved performance can be achieved by aggregation in a static environment, such as in Multiple Classifier Systems (MCSs), aggregation in RL field has not been well explored yet. When aggregation is integrated with RL, it faces several difficulties, including inadequate history data, limited learning time, and dynamic environment. By adopting some techniques, for example, the on-line weights learning and weighted aggregation, AMRLS successfully combines the dynamic RL methods and aggregation techniques together to improve the learning performances in both dynamic learning procedure and steady state.

Usually, AMRLS adopts TDRL methods to learn a policy. TDRL methods can learn a policy on-line to maximize the long term discounted reward. However, these methods explore the solution space only in one direction at a time and have difficulty solving RL problems with large solution spaces. In some complicated problems, it is difficult to learn a policy efficiently with TDRL when the initial values of the value function are set randomly. If some initial knowledge can be provided to establish the initial values, learning efficiency can be significantly improved.

GARL, on the other hand, is particularly suitable for solving problems with a large state and solution space because of its parallel search scheme. Therefore, GARL can quickly establish a preliminary policy. However, in general, GARL is an off-line learning scheme as it evolves policies based on large population size and great generation iterations. The learning procedure of GARL is computationally complex; hence, GARL is difficult to apply to real-time control problems.

Obviously, the two RL strategies are complementary rather than exclusive. The advantages and disadvantages of them motivate people to develop a hybrid learning architecture to complement the two strategies. A new hybrid RL architecture, which combines GARL and TDRL, is proposed in this dissertation to join the strengths of the two strategies together to enhance the learning ability of individual learners working in the AMRLS. This hybrid architecture of GARL and TDRL is called HGATDRL which learns a preliminary policy easily using the off-line GARL scheme and then modifies the policy on-line with TDRL methods. HGATDRL improves the searching ability in a large state space for the learners that perform in the AMRLS, so AMRLS can deal with some complicated tasks that are difficult to handle by TDRL.

1.3 Objective

The objective of this dissertation is to develop a general multiple learning framework for RL to improve learning performance by alternatively performing RL and aggregation during a learning procedure.

In the proposed multiple learning framework, AMRLS, a number of learning algorithms (learners), working in an artificial or real environment, learn together and attempt to perform a common task with a better performance. Aggregation techniques provide cooperation and coordination among the learning algorithms. At each time step, the outputs of individual algorithms, which can be in the form of knowledge, statistical probability, rank of preference, or decision, are aggregated dynamically to produce a more reasonable, efficient, and reliable decision. It is anticipated that through the use of the dynamic aggregation technique, this new learning system will improve the overall learning quality.

Combining the on-line RL algorithms with some sort of improved aggregation techniques is one of the principal contributions of this dissertation. To achieve aggregation, diversity and similarity of different learning methods must be preserved. In the proposed AMRLS, each learner adopts one of the RL algorithms described in Section 3.3 or the same algorithm with different values of a learning parameter to learn its policy for achieving the common goal, and then takes the action determined by an aggregation method. The common goal of the different learning algorithms guarantees similarity, and the different learning scheme of each learner ensures diversity. Both diversity and similarity establish the foundation for the aggregation in the AMRLS.

By introducing aggregation to RL, it is expected that AMRLS can work well without the need of the designers to finetune the learning parameters or choose the optimal learning algorithm. The AMRLS will be tested in several tasks: the maze world problem, pursuit domain problem, cart-pole balancing system, mountain car problem, and flight control system. In order to apply this framework to more complex tasks, generalization techniques are introduced to the AMRLS.

HGATDRL combines the advantages of GARL and TDRL to improve the learning ability for RL in a large state space. An experience cloning module is proposed to connect GARL with TDRL and combine their strengths by converting the learning results of GARL to the initial values for TDRL. By adopting the experience cloning module and improving some genetic operations, HGATDRL is expected to handle some complicated tasks, which are difficult to accomplish with conventional TDRL methods. The proposed hybrid RL architecture is tested in the flight control system.

1.4 Contributions

One principal contribution of this dissertation is the development of AMRLS to enhance the learning ability and improve the performance of RL algorithms. Another main contribution is the proposal of a hybrid RL architecture, HGATDRL, to combine the strengths of GARL and TDRL. HGATDRL improves the learning qualities of the individual learners performing in the AMRLS. The two contributions have been realized through the following aspects.

- Propose an on-line aggregated multiple reinforcement learning framework, AMRLS.

- Develop two aggregation schemes for the AMRLS.
  - Aggregation based on heterogeneous learners, that is, different learning algorithms.
  - Aggregation based on homogeneous learners, that is, the same learning algorithm with different values of the learning parameters.
- Devise the different aggregation styles.
  - Aggregation based on knowledge, for example, Q-values
  - Aggregation based on statistical probabilities, for example, the Boltzmann probability
  - Aggregation based on the ranks (preferences) of candidates
  - Aggregation based on decisions, for example, actions
  - Aggregation based on state information, for example, features
- Improve aggregation effects by introducing the weights, which are obtained during the dynamic learning procedure, to aggregation methods.
- Improve GARL approaches by modified genetic operations such as crossover and mutation.
- Propose a hybrid reinforcement learning architecture, HGATDRL, to combine the strengths of GARL and TDRL.
- Develop an experience cloning module to convert knowledge from GARL to TDRL.
- Control the complicated altitude system of a Boeing 747 aircraft successfully by adopting the AMRLS and HGATDRL architectures.

Based on both analyses and experiments of several on-line test environments, the proposed AMRLS shows a great potential as a new methodology for the research of multiple learning systems and aggregation. Experimental results also demonstrate that the proposed HGATDRL provides a new technique to integrate the two RL strategies efficiently.

1.5 Organization

The dissertation consists of six chapters and is organized as follows.

- Chapter 1 presents the introduction of this dissertation;
- Chapter 2 describes the background about this research work, including RL, generalization, aggregation, and GAs;
- Chapter 3 focuses on the architecture, working procedure, learning algorithms, and aggregation methods of the proposed AMRLS;
- Chapter 4 discusses the GA-based direct policy search RL strategy (GARL) and proposes a hybrid RL architecture, HGATDRL;
- Chapter 5 applies the proposed framework to several realistic examples to illustrate the feasibility of the framework;
- Chapter 6 provides several conclusions obtained from experiments and analysis and suggests the possible directions for future work.

Chapter 2

Background

In this chapter the background about Reinforcement Learning (RL), generalization, aggregation, and Genetic Algorithms (GAs) is provided. This knowledge is closely related to the research topic of this dissertation.

2.1 Reinforcement Learning

Sequential Decision Making (SDM) is common and important in the areas of process control, industrial manufacturing, robot soccer, financial trading, and game-playing. In SDM problems, a set of decisions should be made in dynamic environments to realize some pre-defined objectives. Establishing a SDM strategy by learning rather than by the instructions from designers is a difficult and promising technique. Many techniques and algorithms, such as recurrent neural network, search-based models, evolutionary computational models, and RL, have been applied to sequential learning problems [81].

RL [45, 76, 84, 92] is a learning method that can learn an optimal policy to maximize a type of long term reward without needing to know the dynamics of the environment with which it interacts. Unlike mathematics-based methods such as the optimal control, which explicitly solves a set of differential equations, RL does not rely on the exact dynamic model of a system. Unlike the well-known supervised

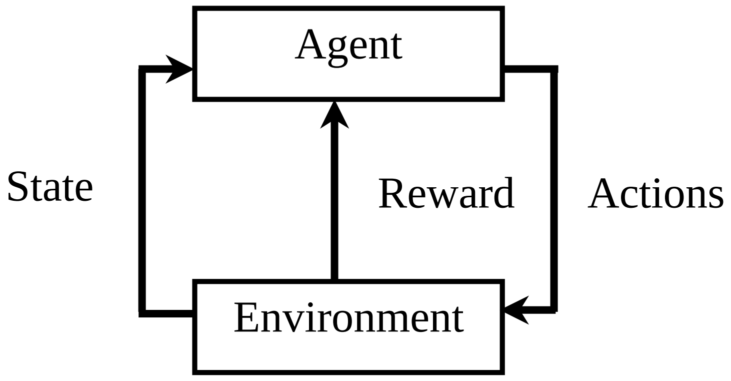


Figure 2.1: Standard model of RL

learning technique, RL does not need the precise training data for learning. Figure 2.1 represents a standard RL model, which describes the process of RL. At each discrete time step, the RL agent observes its current state, takes an action in the environment based on its current policy and state, and receives a numeric reward or punishment when the agent transits to a new state after it takes the action. If an action results in a higher reward, that is, a positive reinforcement, the agent will tend to take the action more in the future, but if an action causes a punishment, that is, a negative reinforcement, the agent will avoid taking the action in future. Therefore, RL uses a trial-and-error learning scheme to learn an optimal control policy that will maximize the discounted sum of expected future rewards (r_t). The reward or punishment is the only scale evaluative feedback given, internally or externally, to the agent when the agent transits from one state to another.

Most RL approaches are studied within the condition of Markov property, which is defined as

$$P_r(s_{t+1} = s', r_{t+1} = r' | s_t, a_t) = P_r(s_{t+1} = s', r_{t+1} = r' | s_0, a_0, s_1, a_1, \ldots, s_t, a_t).$$

Any RL task that satisfies this property is called the Markov Decision Process (MDP) [12]. If the condition of Markov property is met, the optimal policy for a given state is the optimal policy for the entire history before the state. In other words, the Markov property ensures that the choice of action is based only on the current state.

There are two great challenges for RL. First, there is no explicit teacher instruction that indicates the correct output at each time step. Therefore, an RL agent should learn its knowledge by itself. Second, the reinforcement information for RL is delayed, so RL algorithms must solve the temporal credit assignment problem, that is, must assign credit or blame to all of the states and actions that resulted in the final outcome of the sequence. To deal with the two challenges, the trial-and-error strategy and temporal difference technique are developed for RL.

2.1.1 Dynamic Programming, Monte Carlo, and Temporal Difference Methods

Value function-based RL can be divided into three categories, Dynamic Programming (DP), Monte Carlo (MC) methods, and Temporal Difference (TD) methods, according to the schemes for updating the value functions [84]. A brief introduction about the differences among the three categories is given as follow.

Dynamic Programming (DP)

DP is a collection of algorithms that are employed to compute the optimal policy given a perfect model of the environment as a MDP. DP is a bootstrapping method that estimates the value of a state based on the estimation of the value of the successor state. The values of states are updated with Bellman's optimality equation,

$$V(s) = \sum_{s'} P^a_{ss'}[r^a_{ss'} + \gamma V(s')]. \tag{2.1}$$

$V(s)$ is a value function (V-value) that predicts the long-term (discounted) reward that an agent can receive when the agent starts from state s and then follows a given policy. Value iteration or policy iteration algorithms can be adopted to update the V-value when $P^a_{ss'}$ and $r^a_{ss'}$ are given [84].

DP algorithms update the value function based only on the environment model rather than the interactions between the agent and the environment; therefore, DP algorithms are more like a planning approach than a learning approach. Because classical DP algorithms require an exact model of the environment in terms of the transition probability, $P^a_{ss'}$, reward distribution, $r^a_{ss'}$, and need to update all the values over the entire state (or state-action) space, they are unsuitable for realistic tasks. However, DP provides an essential and theoretical foundation for the understanding of RL.

Monte Carlo (MC) Methods

Unlike the DP algorithms that demand complete knowledge about the environment, MC algorithms estimate the value functions directly from the experience of an agent. These algorithms acquire experience, including sample sequences of the states, actions, and rewards, from real or simulated interactions with the environment rather than from the dynamic model of the environment.

MC algorithms do not bootstrap. They learn incrementally in an episode-by-episode sense. Thus, it is only on the completion of an episode that the value estimates and policies are updated, because only at the terminal state can an agent receive a reward. This drawback makes it difficult to use MC for practical applications because if a process is not an episode case, that is, there is no terminal state, the reward cannot be assigned.

Temporal Difference (TD) Methods

TD learning [83] is a combination of MC and DP. Like MC algorithms, TD algorithms can learn directly from raw experience without any prior knowledge about the model of an environment. Like DP algorithms, TD algorithms update their estimates (value function) based, in part, on the learned estimations of the successor state or state-action pair, without waiting for the final outcome. Therefore, TD algorithms rely on bootstrapping to estimate the value functions. For example, the simplest TD algorithm, TD(0), updates its estimated state value V as

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \tag{2.2}$$

where $V(s_t)$ is the value of state s_t, and state s_{t+1} is the successor state of s_t after taking action a_t in s_t. The reinforcement information, r_{t+1}, represents the immediate reward (or punishment) that the agent receives after taking action a_t.

Although both DP and TD(0) bootstrap, TD(0) updates the value of only one state rather than the values of all the states, so TD(0) saves a lot of computation. Although neither MC nor TD methods need the model of the environment, MC methods update the policy only at the end of an episode, whereas TD methods update their policies at each time step by using the instant reward and the expected value of the next state. Consequently, TD methods work more efficiently than DP or MC methods.

A TD algorithm can improve its policy in two approaches: on-policy and off-policy. For the former, the policy learned is the same as the one that the agent is using. SARSA-learning is an example of on-policy learning. For the latter, the policy used to generate the behavior (the behavior policy) might not be related to the estimation policy that is evaluated and improved. Q-learning adopts this approach. In this dissertation, only TD methods are employed to update the value function of RL, and they are discussed in detail in Section 3.3.

2.1.2 Direct Policy Search (DPS)-based RL Methods

DPS approaches indicate that a policy can be represented by some parameters and a structure for using the parameters, and the parameters are optimized by searching a policy space directly. DPS-based RL constructs new policy candidates from those with the highest fitness values observed. Various optimization algorithms can be used for searching, such as stochastic hill-climbing, Levin search, genetic algorithms, genetic programming, and success-story algorithm [72]. In this dissertation, the focus is on genetic algorithms (GAs). The DPS-based RL that involves GAs technique is called GARL, which is discussed in Section 4.1.

2.1.3 Model-based and Model-free RL

According to the objectives that RL wants to learn, RL can also be divided into two categories: model-based RL (e.g., DP method) and model-free RL (e.g., MC and TD). Model-based RL learns a model of the environment, i.e., the transition probability, $P^a_{ss'}$, and the reward distribution, $r^a_{ss'}$, by interacting with the environment and makes decisions based on the model. Model-free RL learns an optimal policy directly from interacting with the environment without model and makes decisions based on the policy.

Model-based RL

Model-based RL, called indirect RL, learns a model from experience by computing the transition probabilities as

$$P^a_{ss'} = \frac{\texttt{The number of transitions in state-action pair } (s, a) \rightarrow s'}{\texttt{Total number of the transitions in state-action pair } (s, a)}.$$

This information can be used to form a model of an environment. Several model-based RL algorithms, such as Dyna method [4], Real-time Dynamic Programming [10], Fuzzy Prioritized Sweeping technique [7], and Bayesian approach [16], have been proposed to improve the learning efficiency of RL. These algorithms are particularly useful in problems where computation and memory are not issues, but the real-world experience is very costly. The advantages of model-based RL are the combination of learning and planning together and saving the experience for replaying. However, the computational load is heavy for this method, and the model might misguide the learning process if the experience obtained to establish the model is not sufficient or correct enough.

Model-free RL

Model-free RL, on the other hand, learns the values of state-action pair Q(s, a) directly. This method adapts to the change of environments faster and more easily than the model-based RL. Nevertheless, this method wastes much real experience. After learning, an agent knows how to act at any state, but it cannot provide any other information about the environment. In this dissertation only model-free RL algorithms are investigated because in simulation environments experience is obtained easily.

Model-free RL presents two challenges. One is the "temporal-credit assignment dilemma", that is, how to assign the credit to each decision in the history according to the evaluation of the whole sequence of decisions. Another is the "tradeoff between exploration and exploitation." TD-learning and eligibility trace techniques [84] are used to meet the first challenge, and several soft-max techniques, such as "ϵ-greedy" and "Boltzmann exploration" techniques, are adopted to deal with the second challenge (discussed in Section 2.1.6).

2.1.4 Eligibility Trace and λ

An important improvement of RL is the adoption of the eligibility trace, which is used as a temporary record of the history of a learning procedure. It is more reasonable that an action should be rewarded or punished not only with its instant reward, but also with the future rewards, because the influence of an action may be delayed for several steps. Surely, the future rewards should be discounted. This is the view of the "forward eligibility trace" [84]. However, it is very difficult to compute the "forward eligibility trace" because it needs the rewards in the far future. For dealing with this problem, a "backward eligibility trace" approach is proposed. In this approach, when a learner receives a reward, the reward should not only be assigned to the nearest previous state-action pair, but also to all the previous state-action pairs by some discounting. With an eligibility trace, a learner can distribute a reward to previous state-action pairs without remembering the learning procedure explicitly.

A vector or matrix $e(t)$ is used to express the eligibility trace and is initialized as zeros. An eligibility trace can be updated in two methods: accumulating and replacing, as follows:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if} \quad s = s_t,\ a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \tag{2.3}$$

and

$$e_t(s,a) = \begin{cases} 1 & \text{if} \quad s = s_t,\ a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \tag{2.4}$$

The eligibility trace factor, λ $(0 \leq \lambda \leq 1)$, controls the temporal credit assignment to the state-action pairs in the history. For TD(λ), if $\lambda = 0$, it is one step TD learning, that is, TD(0). If $\lambda = 1$, TD methods are equivalent to MC methods. The parameter λ provides a smooth heuristic interpolation between these two limits. A family of TD(λ) algorithms can be formed by combining TD methods with the eligibility trace technique (Section 3.3).

2.1.5 Reward Function

The choice of reward functions significantly affects the learning performance of an RL system. A reward function should clearly describe the objective to be optimized and correctly map the behavior of an agent to what the agent is assigned to do [63]. An incorrect defined reward function may stimulate an agent to obtain higher rewards rather than to achieve the desired goal. In some cases, the definition of a reward function is direct and clear. For example, if successful, a positive reward is given, or if failed, a punishment (negative reward) is provided, and the rewards are zeros in the other situations. However, in many cases, it is not very clear how to choose a reward function to meet the requirements of a system's performances, especially for dynamic performance.

2.1.6 Action Selection

One of the interesting problems of RL is the tradeoff between exploration and exploitation during learning. For exploitation, RL algorithms behave optimally according to the current knowledge, whereas for exploration, RL algorithms acquire new knowledge by taking random actions. Exploitation can avoid the risk of obtaining very low or even negative rewards, but may miss the opportunity to obtain larger rewards. On the other hand, exploration can not only produce a more valuable reward, but also a less valuable reward. Without a balance of exploration and exploitation, an RL agent cannot learn successfully. As a result, three popular techniques are often used to select actions in order to balance exploration and exploitation.

1. $\epsilon-$greedy Technique

- With probability ϵ, pick an action uniformly at random;
- With probability 1 - ϵ, choose the greedy action.

However, in stationary problems (that is, a problem in which the parameters do not change with time), continual exploration can lead to suboptimal results, even if the optimal solution has already been learned. Therefore, ϵ should be decreased during the learning.

2. Boltzmann Exploration Technique

   This policy chooses action a in state s_t with a probability based on the state-action values, that is, the Q-values:

$$P(s_t, a) = \frac{e^{\frac{Q(s_t,a)}{T}}}{\sum_{b \in A(s_t)} e^{\frac{Q(s_t,b)}{T}}} \tag{2.5}$$

   Temperature parameter, T, determines the degree of randomness for action selecting, thereby balancing exploration and exploitation. The greater T is, the greater the chance to explore. During the learning procedure, T should be decreased gradually.

3. Semi-uniform Distribution Technique

   This policy chooses action a in state s_t with the probability,

$$P(s_t, a) = \begin{cases} P_{best} + \frac{1-P_{best}}{\text{Number of actions}} & \text{if action } a \text{ maximizes Q-value;} \\ \frac{1-P_{best}}{\text{Number of actions}} & \text{otherwise.} \end{cases} \tag{2.6}$$

   P_{best} balances exploration and exploitation. If $P_{best} = 0$, it is pure random exploration. If $P_{best} = 1$, it is pure exploitation.

2.1.7 Difficulties of the Current RL Algorithms

Although RL has been applied to various domains successfully, there are still some difficulties that restrict the applications of RL. The following presents some difficulties related to this research.

Parameter Setting

There are several adjustable parameters for TD(λ) algorithms. The main parameters are the learning rate, α, eligibility trace decay factor, λ, and discount rate, γ. Many RL algorithms work well only under the correctly set parameters. Normally, these parameters are heuristically tuned for a given task. However, tuning is a time and computation cost job, and the fixed parameters cannot be adapted to changes of the environment or different learning procedures. More important, many parameters are related, i.e., the change of one parameter affects the others. Therefore, parameter setting and tuning is a difficulty for the application of RL.

The Curse of Dimensionality

If a state space contains m state variables, and each variable is discretized into n values, then the number of states is n^m, which is exponentially large in m. To apply RL algorithms to large-dimensional or continuous problems, several generalization techniques such as value gradient generalization methods, value function estimation, function approximation with neural network, and support vector machine are proposed [22]. However, the universal theoretical proofs for the convergence of RL algorithms with generalization are not available.

Efficiency of Learning

Although TD methods are regarded as on-line learning methods, significant time is required to establish the value function. It is very risky to apply TD methods to a real-time control problem if there is no pre-experience about the environment. How to improve the learning efficiency of TD methods by using experience cloning or other methods is a research topic to be attended to.

2.2 Generalization and Function Approximation

The conventional RL algorithms express their value function in a table. Each entry of the table stores the expected return of a state-action pair. One of the disadvantages of tabular-based RL algorithms is the “curse of dimensionality,” which means that the size of the Q-value table grows dramatically as the environment becomes more complicated. Curse of dimensionality makes tabular-based RL algorithms unsuitable for many practical problems, because not only is much time needed to calculate the value function, but also much data (information) is required to fill the table. Another disadvantage of tabular-based RL algorithms is that they do not permit generalization. Thus, each state or state-action pair must be learned because there is no way to generalize between states. However, in real applications, the size of state space is very large and even continuous, and many states are not trained forever. In order to make RL applicable, generalization is necessary. Generalization is a technique that enables an agent (learner, algorithm) to give an approximately correct output for an input state that has not been trained before, as long as some neighborhoods of the input state have been trained; that is, generalization can use a finite number of samples from a desired function to construct an approximation structure or function to represent the desired function as accurately as possible.

Generalization is performed in different ways; fuzzy logic, neural network approximation, nearest neighbor method, and statistical pattern recognition are several efficient methods. In this dissertation, the focus is on two Function Approximation (FA) methods, linear FA [73] and Cerebellar Model Articulation Controller (CMAC) [2] methods, which are discussed in detail in following sections.

In FA, the values of states or state-action pairs are not given by a table. Instead, they are represented as a function of a parameter vector $\overrightarrow{\Theta}$ and the structure of the approximation function or the feature vector $\overrightarrow{F}$. $\overrightarrow{F}$ is constructed from the states in various approaches. The goal of FA is to learn the parameter vector $\overrightarrow{\Theta}$ and the structure based on a set of training data to express the computational relationship between the input and output variables. Once such a relationship is learned from the training data, it can be used for predicting the output values of the previously untrained states. Typically, the dimension of $\overrightarrow{F}$ is larger than the dimension of the state variables, but much smaller than the number of state points. Often, the Mean Squared Error (MSE) for the approximation $\widehat{V}$ and the desired function value V is used as the measurement of the approximation performance by calculating

$$MSE(\overrightarrow{\Theta}) = \sum_{s \in S} W(s)(V(s) - \widehat{V}(s))^2, \tag{2.7}$$

where $W(s)$ is utilized to weight the errors of the different states.

2.2.1 Learning Methods for Function Approximation

To construct an approximation function is to learn a parameter vector to minimize the error in Equation (2.7). The following three methods are used in this dissertation to learn the parameter vectors.

Gradient-descent Methods

FA is regarded as an example of supervised learning. If the desired output of an approximation function can be given by TD methods, FA approaches can be used in RL. As with conventional supervised learning situation, the parameter vector of an approximation function in RL can be updated by the gradient-descent method. The information about the gradient of a function specifies a descent direction, so that the parameters are updated along this direction to minimize the error in Equation (2.7). Although the gradient-descent method cannot guarantee convergence to the global optimal point, the results of a FA are often good enough if a reasonable initial value is chosen. Linear FA (in Section 2.2.2) is an examples of this method.

Memory-based Methods

The approximation functions can also be learned by using memory-based learning methods [6], or exemplar-based methods [37]. CMAC (in Section 2.2.3) is an examples of this method. In these methods, the training data is expressed as exemplars and stored in memories. The output value corresponding to an input state is determined by interpolating the stored exemplars close to the input and combining them using different weights, which can be obtained by some operations. K-nearest neighbor algorithm is a popular algorithm to achieve generalization in this case.

Evolution Methods

If the architecture of the approximation function is determined, the parameters of the function can be learned by evolution methods, for example, GAs. This method is very powerful because the architecture of the approximation function can assume in any form.

2.2.2 Linear Function Approximation

An approximation function can be expressed by the feature vector in forms such as exponential forms, polynomial expression, or linear combination. The most popular FA expression is the linear FA, which indicates that the approximation function is defined as a linear combination of features [84], expressed as

$$\widehat{V} = \overrightarrow{\Theta}^T \cdot \overrightarrow{F_s}, \tag{2.8}$$

where $\overrightarrow{\Theta}$ is the parameter vector, which can be optimized by RL, GAs, or other optimization methods. For the linear FA, the vector $\overrightarrow{\Theta}$ can converge to its global optimal value.

When the linear FA method is used to approximate the Q-value, the approximation is

$$\widehat{Q}(s, a_j) = \overrightarrow{\Theta_j}^T \cdot \overrightarrow{F_s} = \sum_{i=1}^{n} \theta_j(i) \cdot f_s(i). \quad j = 1, \ldots, m, \tag{2.9}$$

where $j = 1, \ldots, m$ denotes that there are m actions to be selected in each state. Thus, the total number of parameters to be learned should be $n \cdot m$ in this case, and the parameters are expressed as an $n \cdot m$ matrix. The parameter matrix $\overrightarrow{\Theta}$ should be learned to

$$\min \sum_{s \in S} P_j(s)(Q(s, a_j) - \widehat{Q}(s, a_j))^2, \quad j = 1, \ldots, m. \tag{2.10}$$

Using the gradient-descent optimization method for example, parameter matrix Θ is calculated as

$$\overrightarrow{\Theta}_{j,t+1} = \overrightarrow{\Theta}_{j,t} + \alpha \delta_t \overrightarrow{e}_t, \tag{2.11}$$

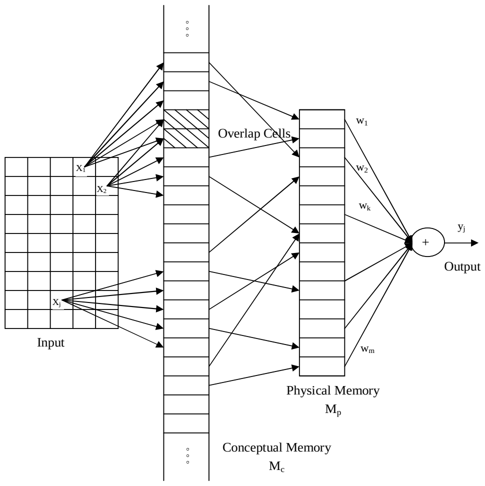


Figure 2.2: The architecture of CMAC

where

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_k) - Q(s_t, a_j), \tag{2.12}$$

and

$$\overrightarrow{e}_t = \gamma\lambda\overrightarrow{e}_{t-1} + F(s); \qquad \overrightarrow{e}_0 = \overrightarrow{0}, \tag{2.13}$$

where, parameters α, γ, λ are the same as those explained in Section 2.1, and $Q(s_t, a_j)$ is calculated by Equation (2.9), rather than given by a table.

After the parameter vector or matrix $\overrightarrow{\Theta} = [\overrightarrow{\Theta}_1; \overrightarrow{\Theta}_2; \ldots; \overrightarrow{\Theta}_n]$ is learned, the outputs of states or state-action pairs can be calculated by the corresponding input values and the parameter vector or matrix, and then an optimal control policy is produced based on the outputs.

2.2.3 Cerebellar Model Articulation Controller (CMAC)

Cerebellar Model Articulation Controller (CMAC) method, first introduced by Albus [2], is another popular generalization method and has been proven to be feasible for RL [17, 51, 82]. Unlike the tile coding method (explained in Section 3.4.9), which generates generalization by the overlap of tiles, CMAC uses neighbors to generalize. It is expected that similar states will generate similar optimal actions. In CMAC, an input state stimulates not only the Q-value exactly corresponding to the state, but also all the Q-values of the neighbors of the state. CMAC uses state variables as an index to store the information into a set of memories (neighbors) rather than only its own location memory. At the same time, RL learner with CMAC makes its decision based on the summed Q-values of all the neighbors. With CMAC technique, a state can share some information with its neighbors, and knowledge can be obtained even for states that have not been visited. As a result, CMAC possesses the ability to generalize. At the same time, because multiple states can be mapped to the same memory by using some hash-coding methods, the number of required memory cells is smaller than the number of states.

Albus's CMAC method maps each input (vector) to a set of m_n points in conceptual memory, M_c. Where m_n is the number of the neighbors of an input. Normally, the size of the conceptual memory, M_c, is very large. A hash-coding [50] method is adopted to map conceptual memory M_c into physical memory M_p, which is a one-dimensional memory with size m (shown in Figure 2.2.) Usually, m is much smaller than the size of the input space, but should be set carefully. An m that is too small causes a collision that maps more than one input into the same physical memory and lowers the accuracy of mapping. Therefore, this method is efficient only when fewer states store their information at each time step. It is helpful that most control problems belong to this category.

Figure 2.2 depicts the architecture of CMAC. This generalization method is appropriated for the pitch control system, and the experimental results are discussed in Section 5.5.1.

2.3 Aggregation

Aggregation is a technique for combining information from multiple sources. This technique is widely applied in the areas of election, decision-making, operations research, image processing, classification, and automatic control. Several aggregation methods have been proposed and successfully applied to various areas to improve the learning qualities such as accuracy, confidence, redundance, complement, robustness and fault tolerance. In this section, a brief overview of aggregation is presented.

2.3.1 Overview of Aggregation

Several aggregation methods, including fuzzy integral [27], Behavior Knowledge Space (BKS) [36], Decision Templates (DT) [48], mixture of experts [44], and the Bayes approach [89] have been proposed for Multiple Classifier Systems (MCSs). However, most of them are data-dependent aggregation approaches [46] and work well in off-line learning.

MCS is the most active area that adopts aggregation techniques to improve classification performances. However, MCSs are static decision making processes and can reuse the training data to analyze the statistical property of data to enhance aggregation ability. For a dynamic on-line RL algorithm, the method to combine aggregation techniques poses a new research problem.

To combine aggregation techniques with RL algorithms, some data-independent decision-based methods are taken into account. The most common decision-based aggregation techniques are voting and ranking. There are several voting methods such as maximum, minimum, median, and plurality voting, but the most popular method is the Majority Voting (MV). When two alternatives are considered, the one preferred by most voters is selected. The majority rule with two alternatives possesses the most desirable properties of voting systems. In fact, it is often considered the best method for preference aggregation [70]. Ranking methods, on the other hand, sort the decisions of different learners and then choose the decision with the highest ranking. The Borda Count [32] (BC) method and Instant Runoff Voting (IRV) method are two examples of the ranking method.

2.3.2 Aggregation Algorithms

Several popular aggregation algorithms, which are examined in this dissertation are introduced as below.

1. Average Probability Algorithm

   Different algorithms yield different probabilities for an action. The aggregation algorithm averages the probabilities as follows:

   $$D_j(s) = \frac{\sum_{i=1}^{m} P_{i,j}(s)}{m}, \qquad j = 1, ..., n, \tag{2.14}$$

   where n is the number of actions and m is the number of learners (algorithms). Then, the aggregation algorithm chooses the action based on the probabilities.

2. Maximal Probability Algorithm

   The only difference between this algorithm and the average probability algorithm is the use of the highest probability values, rather than average values, to obtain the probabilities of the different actions. Therefore,

   $$D_j(s) = \max_{i=1}^{m} P_{i,j}(s), \qquad j = 1, ..., n, \tag{2.15}$$

   Then, the aggregation algorithm chooses the action with the $\epsilon - greedy$ or Boltzmann exploration strategy to maintain the balance of exploration and exploitation.

3. Majority Voting (MV) Algorithm

   Majority voting is one of the oldest strategies for decision making. Different RL algorithms choose their best actions according to their Q-values or the probabilities, respectively. Then, this strategy chooses the best action based on the majority, which means the best action is the one most often chosen by the different RL algorithms.

   If there are m learning algorithms, the majority voting method will give a correct decision if at least $floor(m/2) + 1$ learning algorithms give correct outputs. If each learning algorithm has probability p to make a correct decision, then the aggregated learning system will have the following probability P to make a correct decision [49].

   $$P = \sum_{i=floor(m/2)+1}^{m} \begin{pmatrix} m \\ i \end{pmatrix} p^i (1-p)^{m-i}. \tag{2.16}$$

4. Plurality Voting [67]
   It is a winner-take-all voting method for decision making. The difference between this method and the majority voting is that this method chooses an action based on the most voting, regardless of whether or not the voting is majority.

5. Borda Count (BC) Algorithm [32]
   For any action, a, the Borda count is the sum of the number of actions ranked below a by each RL algorithm. If $B_i(a)$ is the number of actions ranked below action a that is chosen by the i^{th} RL algorithm, then the Borda count for action a is
   $$B(a) = \sum_{i=1}^{m} B_i(a). \tag{2.17}$$
   Then, the action with the largest Borda count will be selected as the aggregated output.

6. Instant Runoff Voting (IRV) Algorithm
   IRV is a combination of majority voting and Borda count. If no action is a majority in the first run of selection, the action with the lowest rank (Borda count) is eliminated and the second selection of the RL algorithm, for which the first selection is eliminated, is used for voting. This procedure continues until one action becomes a majority. Then, this action is selected as the aggregated output.

7. Weighted Average Probability Algorithm
   The use of the weighted average algorithm, rather than a simple average algorithm, should produce better results. This algorithm is expressed as
   $$D_j(s) = \frac{\sum_{i=1}^{m} w_i P_{i,j}(s)}{\sum_{i=1}^{m} w_i}, \tag{2.18}$$
   where w_i is the weight for algorithm i at state s.

8. Weighted Majority Voting (WMV) Algorithm
   The decision of each learner is weighted by the weights w_i, which can be

obtained by some on-line information. Then, the aggregation algorithm takes majority voting based on the weighted decisions.

9. Weighted Borda Count (WBC) Algorithm
   This algorithm is similar to the WMV, but it makes decision based on the weighted Borda count.

Determining the weights for aggregation is not very easy. Several researchers [32, 46, 80, 90] have proposed different methods to automatically determine the weights. However, these methods depended too much on the designer's experience and the characteristics of the objects considered. In RL problems, the weights should be adapted according to the number of times that one algorithm is selected previously or according to the number of times that one algorithm succeeds. A more attractive method is to learn the weights on-line.

2.4 Genetic Algorithms (GAs)

Genetic algorithms (GAs), which belong to Evolutionary Algorithms (EAs), are very popular and efficient for searching optimal solutions in a large solution space. In this dissertation, GAs are the key methods used for DPS. Some background about GAs is provided as follows.

2.4.1 Overview

GAs are evolution-based optimization algorithms whose operations are inspired by the biological process of evolution [55]. The first GAs appeared in the late 1950s and early 1960s. These algorithms had been studied by evolutionary biologists who wanted to use computers to simulate some aspects of natural evolution. Fraser was one of the researchers who worked most closely with the current concepts of GAs. The work of Holland and his students, done in the early 1960s, significantly influenced the field of GAs. Holland first explicitly proposed crossover and other recombination operators. In 1967, Holland's student, Bagley, first used the term

Genetic Algorithm in his dissertation [8]. His algorithm resembles many of the evolution operations used today, such as selection, crossover, and mutation. In 1975, Holland published his book, *Adaptation in Natural and Artificial systems* [33], which is one of the most important books in the area of GAs. Since then, GAs have become well established soft-computing techniques and have been applied to a number of problems, and researchers still continue to look for new applications of GAs.

2.4.2 Working Procedures and Operations of GAs

GAs use a chromosome, an individual of a population pool, to represent a potential solution of the studied problem. In each episode or generation, each chromosome in a population pool is updated by genetic operations such as selection, crossover, and mutation. Finally, some chromosomes are chosen to form a new generation based on a fitness function, which evaluates the fitness of each chromosome (solution). The larger the fitness value of a chromosome, the higher the chance the chromosome is chosen. Typically, there are eight steps in applying GAs.

1. Define a fitness function
2. Represent the potential solutions as chromosomes
3. Initialize the population
4. Produce offspring
    - Crossover parents to create offspring
    - Mutation of offspring
5. Evaluate the fitness value of each chromosome
6. Select chromosomes with probabilities derived from the fitness value
7. Form a new population from the selected chromosomes
8. Repeat steps 4 to 7 until the termination criterion is satisfied

Chapter 3

Proposed Learning Framework: an Aggregated Multiple Reinforcement Learning System (AMRLS)

A single RL algorithm is often inadequate to deal with the increasing complexity of realistic tasks and to fulfill the specific requirements for control problems. Each learning algorithm is established based on certain assumptions; therefore if some assumptions are not satisfied for a learning algorithm, its learning performance may be poor or unacceptable. Aggregation of several learning algorithms that are based on different learning methods can improve learning performances by enhancing their advantages and eliminating their disadvantages. In this chapter, an Aggregated Multiple Reinforcement Learning System (AMRLS) is proposed to enhance learning ability and improve learning quality. Some TD-based RL algorithms and aggregation methods used in this dissertation are also introduced in this chapter.

3.1 Multiple Reinforcement Learning Systems

The architecture of multiple RL systems can be roughly classified into two types [31], RL Individually (RLI) and RL in Group (RLG). For RLI, each learner learns its policy individually with RL algorithms such as Q-learning, SARSA-learning, or AC method, and takes other learners as a part of its environment. Although this architecture is often used in multiagent systems [5, 85, 96], there are two difficulties. One is computational complexity; due to the increasing number of learners, the computation load will increase dramatically. The other is the problem of convergence. Since the environment is no longer stationary, convergence cannot be guaranteed.

For RLG [39, 40, 41, 93], learners learn together and take actions based on strategies such as aggregation, coordination, or cooperation. This approach addresses the convergence problem, "since changes to a learner's policy are only in the context of groups and therefore must be coordinated with those of other learners" [31]. In this case, aggregation or cooperation is necessary for obtaining a more reasonable output. This is the motivation for proposing the new multiple learning algorithm architecture described in the next section.

3.2 The Aggregated Multiple Reinforcement Learning System (AMRLS)

AMRLS is a learning system designed to on-line combine multiple RL algorithms with aggregation techniques in order to improve learning abilities. AMRLS can be used as a controller, for example, in the mountain car problem and cart-pole system [40, 41], or as a solver, in the maze environment [39]. In more general cases, AMRLS can be used in Multiagent Systems (MASs). In this situation, AMRLS can enhance the learning ability of each agent, and therefore, improve the performance of overall system.

3.2.1 Motivation for Combining Aggregation Techniques with RL

Although many RL algorithms are successful in various applications, none of these algorithms is always consistently superior to the others. Thus, it is not easy to determine which algorithm should be adopted for a given application. Even when a learning algorithm is chosen based on experience or experimental results, its learning quality is substantially influenced by the values of its parameters. Ideally, these parameters should be adaptable to changes in the environment to perform optimally. However, the parameters of all existing RL algorithms are set by designers based on their experience and experiments. Such a practice is time and effort consuming and might not guarantee obtaining optimal values. Therefore, in this dissertation, a novel multiple learning system, AMRLS, is developed to avoid the search for the best learning algorithm or the optimal learning parameters.

The successful applications of aggregation and fusion techniques in MCSs encourage the integration of aggregation methods with dynamic RL procedures. These applications establish the theoretical and practical motivations for AMRLS. Since AMRLS is a dynamic learning system, aggregation should be made on-line rather than at the end of experiments. Moreover, there is not enough memory to record all the historical data during the learning process; therefore, aggregation should be made based mainly on the current information rather than the history information. This dissertation focuses on ways to handle the two problems in order to integrate aggregation techniques with RL algorithms efficiently.

3.2.2 Architecture and Working Procedures of AMRLS

Figure 3.1 demonstrates the architecture of the proposed AMRLS. Typically, aggregation is reasonable only when the basic elements, for example, the learning algorithms in AMRLS, are sufficiently diverse but similar. The different learning algorithms ensure diversity, while the shared goal of the different algorithms provides similarity; both factors form the foundation for aggregation.

The proposed AMRLS consists of two major modules, a learning module and

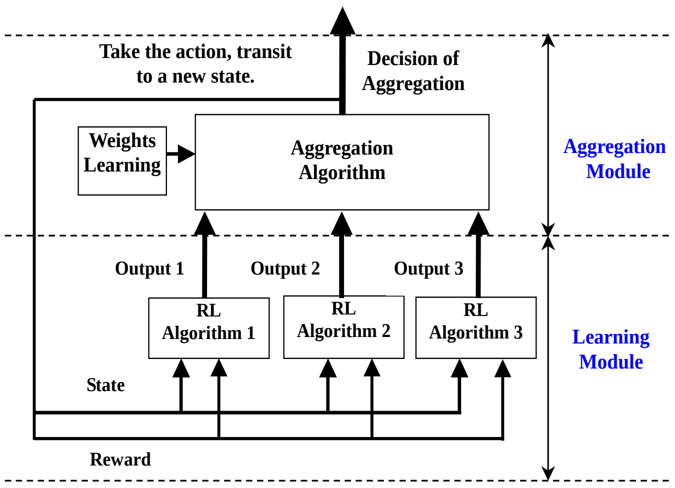


Figure 3.1: AMRLS architecture

an aggregation module. These two modules interact with each other and engage alternately in a SDM process.

The learning module is in the first level, in which different learners learn their control policies individually, synchronously or asynchronously, in parallel or in serial. These learners can be heterogeneous, which means that learners adopt different RL algorithms, or homogeneous, which means that learners use the same RL algorithm but with different values of a certain learning parameter, such as λ or α. At each time step, every learner makes its decision independently, based on its current policy and state. Then, each learner submits its decision of the selected action or the preference of actions to the aggregation module.

Figure 3.2 is the architecture of an RL learner shown in Figure 3.1. In this architecture, the learning scheme updates the value function based on the input state, instant reward, new state, and RL method. Basically, the value-function is represented in a tabular form, but for complicated applications, some generalization methods are adopted to generalize the value function. The action selector chooses an action based on the value function and some exploration and exploitation strategies.

The aggregation module is at the second level, where the input information

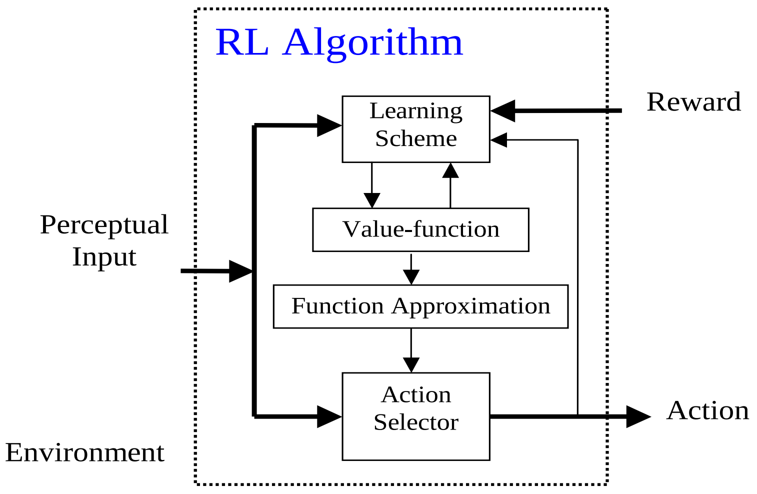


Figure 3.2: Architecture of value function-based RL

(knowledge, probabilities, rank, or decisions about the action candidates) from the learning module is dynamically aggregated with the weights that are learned online. Some information, such as the number of times that each algorithm (learner) has been selected by the AMRLS, is recorded for weighting the different learners. According to the types of information provided by the learning module, aggregation can be performed based on knowledge, such as the Q-values; on statistic information, for example, the probabilities for selecting different actions; on some numerical information, for instance, the ranks of different actions; or on boolean information, such as the decisions of the selected actions.

After that, the aggregation module sends a final decision of action back to the learning module. Then, every learner in the learning module takes the action, transits to a new state, and obtains an instant reward about the action. Subsequently, each learner updates its policy based on the new state, instant reward, and its learning algorithm. With the new state and new policy, new outputs of different learners are provided to the aggregation module again, and a new cycle, called a step, starts.

In the AMRLS, the two dynamic processes, learning and aggregating, perform alternatively and are repeated for a number of steps or until several criteria are satisfied. The working procedure of AMRLS can be described as follows.

```
Learning Procedure of AMRLS
Let $m$ = the number of algorithms;
    $n$ = the number of action candidates;
Initialize all $Q_i(s, a_j),\ i = 1, ..., m,\ \ j = 1, ..., n$;
Repeat (for each episode);
  Choose an initial state $s_0$ and a random action $a_0$;
    Repeat (for each step of an episode);
      (1) In state $s_t$, decide action $a_t = AggFunction(Q_i(s_t, a_t)),\ i = 1, ..., m$;
      (2) Take action $a_t$ and transfer to a new state $s_{t+1}$;
      (3) Obtain an instant reward $r_{t+1}$;
      (4) Update $Q_i(s_t, a_t),\ i = 1, ..., m$ individually;
      (5) $s_t \leftarrow s_{t+1}$.
    End
End
```

Here, *AggFunction()* is one of the aggregation methods that are described in Section 3.4.4.

3.3 TD-based RL Algorithms Used for AMRLS

TD method is an efficient and popular learning approach for RL. This section provides a description of several TD-based RL (TDRL) algorithms adopted in this dissertation.

TD methods estimate, rather than exactly calculate, the value function and use the TD-error to update the knowledge of learning algorithms. Both the state value, V-value $V(s_t)$, and state-action pairs value, Q-value $Q(s_t, a_t)$, can be updated with TD methods. For problems of automatic control, $Q(s_t, a_t)$ is preferred to $V(s_t)$. Several TD-based RL algorithms adopted in this dissertation are introduced as follows.

Actor Critic (AC) Algorithm

The Actor Critic (AC) algorithm [9] is a human-like learning algorithm and is derived from the policy iteration algorithm, where the policy evaluation acts as a critic of the actor's behavior. In AC algorithms, there are two separate components, the policy structure, called the *actor*, and the value function estimator, called the *critic*. *Actor* is used to select actions, and *critic* is used to evaluate the selected actions. The output of the *critic* is normally a TD value, which evaluates the action selected by the *actor*. Then, the *actor* updates its policy according to the TD evaluation.

SARSA-learning Algorithms

SARSA-learning [71], is an on-policy learning algorithm, which attempts to improve its learning policy (value function) based on the real state-action pair it takes in the next step. The value function is updated as

$$Q_k(s_t, a_t) = Q_{k-1}(s_t, a_t) + \alpha[r_{t+1} + \gamma Q_{k-1}(s_{t+1}, a) - Q_{k-1}(s_t, a_t)]. \tag{3.1}$$

From Equations (3.1) to (3.7), $Q(s_t, a_t)$ implies the Q-value for action a_t at state s_t and is called Q-value function, which is the total discounted reward that the learning algorithm expects to accumulate when starting in state s_t, taking action a_t, and following the current policy. The parameter α, which varies between 0 and 1, is known as the step-size or learning rate. During learning, α should be decreased. The parameter γ, which measures how the instant reward r_{t+1} is weighted compared with the prediction of future rewards, is the discount factor. State s_{t+1} is the new state to which the algorithm will transit if it takes action a_t at state s_t. The value function $Q_k()$ is the new Q-value after updating, and $Q_{k-1}()$ is the old Q-value before updating.

Q-learning Algorithm

Q-learning [92], on the other hand, is an off-policy learning algorithm. Off-policy scheme improves its learning policy not based on the real state-action pair it will take in the next step. Q-learning updates its value function by the following computation:

$$Q_k(s_t, a_t) = Q_{k-1}(s_t, a_t) + \alpha[r_{t+1} + \gamma(\max_{a \in A(s_{t+1})} Q_{k-1}(s_{t+1}, a)) - Q_{k-1}(s_t, a_t)]. \quad (3.2)$$

The reason for combining the instant reward r_{t+1} and the estimated value of the next state-action pair is to maximize the long term discounted accumulated rewards. Otherwise, a learner tends to choose an action that produces a high instant reward but transits the system to a new state with lower Q-value, which is highly undesirable.

SARSA(λ)-learning Algorithm

SARSA(λ) is also an on-line, on-policy learning algorithm and updates its Q-values by calculating the following:

$$Q_k(s_t, a_t) \ = \ Q_{k-1}(s_t, a_t) + \ \alpha_t(r_{t+1} + \gamma Q_{k-1}(s_{t+1}, a_{t+1}) - Q_{k-1}(s_t, a_t))e_t(s, a), \quad (3.3)$$

where $e_t(s, a)$ is a type of eligibility trace, which can be by

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & \text{if} \quad s = s_t, \ a = a_t; \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise,} \end{cases} \quad (3.4)$$

if the accumulating scheme is adopted.

All the symbols and parameters are the same as those of the SARSA-learning algorithm except that parameter λ is added, which is a factor of eligibility trace, as aforementioned in the Section 2.1.4. Obviously, the eligibility value of a state increases each time when the state is visited and decreases exponentially otherwise. The states that are visited earlier are assigned less credit for the current temporal difference.

Q(λ)-learning

The Q-values of a Q(λ)-learning algorithm are updated using the eligibility trace technique. The difference between SARSA(λ)-learning and Q(λ)-learning is that

Q(λ) adopts an off-policy learning scheme. The Q-value is updated as

$$Q_k(s_t, a_t) \; = \; Q_{k-1}(s_t, a_t) + \alpha_t(r_{t+1} + \gamma(\max_{a \in A(s_{t+1})} Q(s_{t+1}, a)) - Q(s_t, a_t))e_t(s, a), \quad (3.5)$$

and

$$e_t(s, a) = \begin{cases} \gamma\lambda e_{t-1}(s, a) + 1 & \text{if} \quad s = s_t, \; a = a_t; \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise;} \\ 0 & \text{if non-greedy action taken.} \end{cases} \quad (3.6)$$

All the symbols and parameters are the same as those in the SARSA(λ)-learning algorithm. However, the resetting of the eligibility trace $e(s, a)$ is different from that used in the SARSA(λ)-learning algorithm. At each time step, if a non-greedy action is really taken, all the $e(s, a)$ values should be reset to zero. The reason for the resetting scheme for $e(s, a)$ is that Q(λ)-learning updates the Q-value based on the greedy action, and if the real action is not the greedy one, the history of past state-action pairs, i.e., the eligibility trace $e(s, a)$, has no influence on further TD updates.

R-learning Algorithm

Another RL algorithm used in this dissertation is the R-learning algorithm [52, 74, 84], which is an off-policy learning scheme. Unlike the Q(λ)-learning algorithm, which is also an off-policy learning scheme but it accumulates the discounted rewards, the R-learning algorithm calculates the average of the expected reward per time step under the policy. The Q-values are updated by

$$Q_k(s_t, a_t) = Q_{k-1}(s_t, a_t) + \alpha_t \delta_t e_t(s, a), \quad (3.7)$$

where

$$\delta_t = r_{t+1} - \rho \; + \max_{a \in A(s_{t+1})} Q_{k-1}(s_{t+1}, a) - Q_{k-1}(s_t, a_t). \quad (3.8)$$

If

$$Q_k(s_t, a_t) = \max_{a \in A(s_t)} Q_k(s_t, a),$$

then

$$\rho = \rho + \beta[r_{t+1} - \rho + \max_{a \in A(s_{t+1})} Q_{k-1}(s_{t+1}, a) - \max_{a \in A(s_t)} Q_k(s_t, a)],$$

where, ρ is the average reward because it approximates the average expected reward, and $\alpha_t,\ \beta$ are scalar step-size parameters. This RL algorithm is called R-learning “because the state-action values are relative to the average reward under the current policy” [84]. Obviously, R-learning maximizes the average reward ρ, which is independent with any special state.

TD(λ) with Function Approximation

The procedure of TD(λ) with linear FA is summarized as follows:

```
Algorithm of TD($\lambda$) with linear FA
Initialize: Let parameter matrix $\Theta(i,j) = 0, \overrightarrow{e}(i,j) = 0,\ i = 1, ..., k,\ j = 1, ..., n$,
where $k$ is the number of candidate actions, and $n$ is the size of feature vector.
Repeat (for each episode);
 Choose an initial state $s_t$ and a random action $a_t$.
Transfer state $s_t$ to feature space $F_t$.
  Repeat (for each step of an episode);
    (1) Take action $a_t$ and transit to a new state $s_{t+1}$;
        Transfer state $s_{t+1}$ to feature space $F_{t+1}$;
    (2) Obtain an instant reward $r_{t+1}$;
    (3) Determine the action $a_{t+1}$ in state $s_{t+1}$;
    (4) Calculate the TD error $\delta = r_{t+1} + \gamma V(\Theta, F_{t+1}) - V(\Theta, F_t)$;
    (5) Update eligibility traces value $\overrightarrow{e} = \lambda\gamma\overrightarrow{e} + F_t$;
    (6) Update parameter matrix $\Theta = \Theta + \eta\delta\overrightarrow{e}$;
    (7) $s_t = s_{t+1}$.
  End
End
```

3.4 Aggregation Strategies for AMRLS

Aggregation is a key procedure in the AMRLS. In a single RL algorithm system, only the information of the selected action is used for learning. Some information, such as the magnitudes of Q-values or the probabilities of different actions, are discarded. However, discarding of the information may cause poor learning performance in some cases, for example, in the case when the values of the highest probability and the second highest probability of actions are very close. In an AMRLS, by aggregating multiple learners, various information is used to improve learning performance. In this section, some aggregation strategies used for the AMRLS in this dissertation are explained.

3.4.1 Information Types Provided by Individual Learning Algorithm

Individual learning algorithm in an AMRLS can provide several types of output, which can be divided into four types.

- Type 1: Decision of the Selected Action
  Based on the Q-value of each state-action pair (s_t, a_i) in state s_t, each individual learner in the learning module provides its decision of the selected action to the aggregation module, where a type of aggregation method, for example, the majority vote method, is adopted to make final decision. A certain exploration strategy is used to balance exploration and exploitation.

- Type 2: A Vector of Ranks of the Actions
  Each individual learner in the learning module provides a vector of ranks of all action candidates based on its Q-values to the aggregation module, where a type of aggregation method, for example, the Borda count method, is adopted to make final decision.

- Type 3: A Vector of Probabilities of the Actions
  Each individual learner in the learning module provides a vector of probabilities of all action candidates based on its Q-values and a certain exploration

strategy to the aggregation module, where a type of aggregation method, for example, the weighted average method, is adopted to make the final decision.

- Type 4: A Vector of Q-values of the Actions
  Each individual learner in the learning module provides a vector of Q-values of all action candidates to the aggregation module, where a type of aggregation method, for example, the weighted average method, and a certain exploration strategy are adopted to make final decision.

Obviously, Type 4 provides more information than Types 1, 2, and 3 for aggregating, and better learning performance is expected in Type 4. However, more constraints are required for Type 4, for example, homogeneous learning scheme is necessary for using information of Type 4. The details about the corresponding aggregation algorithms are described in following sections.

3.4.2 On-line Weights Learning

Although several simple data independent aggregation methods, such as majority voting, Borda count, instant runoff voting, and average, can be utilized in AMRLS, some information obtained during the learning should be used to improve the effects of on-line aggregation. This procedure is called weights learning.

A credit recording algorithm (weights learning) records the strength or weight of each algorithm. The strength or weight, which measures the effectiveness of a learning algorithm, is updated incrementally by recording the number of times that each algorithm has been selected, or by summing up the instant rewards distributed to each algorithm previously. Here, an algorithm is selected if the decision of this algorithm is the same as the decision made by AMRLS. At each step, the reward received by AMRLS is distributed to each algorithm that gives the same decision as that of AMRLS.

The weights also reflect the degree of confidence for each algorithm. When voting techniques are used to aggregate different learners, these weights can be used to weight up different learners. A similar system is the Learning Classifier Systems (LCS) introduced by Dorigo and Colombetti [23]. In LCS, the strength of each classifier (decision rule) is calculated by a TD-like algorithm, the bucket brigade algorithm, and is used for bidding or resolving conflicts. In this dissertation weights are learned with the following methods.

- Accumulative method

  This method is based on the number of times each learner has be chosen before. Let $malg$ be the number of algorithms (learners) and $W = [w_1, w_2, \ldots, w_{malg}]$ be the weight vector.

  Initialize $W = [1, 1, \ldots, 1]$ and repeat the following updating at each time step.

  $$\begin{cases} w_i = w_i + 1 & \text{if algorithm } i \text{ is selected;} \\ w_i = w_i & \text{otherwise.} \end{cases}$$

- Hill climber method

  This method is similar to the accumulative method, but it updates the weights as

  $$\begin{cases} w_i = w_i + \delta \ (0 < \delta < 1) & \text{if algorithm } i \text{ is selected;} \\ w_i = w_i - \frac{\delta}{malg-1} & \text{otherwise.} \end{cases}$$

- Performance method

  The performance of different algorithm in different learning stages is different. The weights of each algorithm are changed by simulations.

After the weights $W = [w_1, \ldots, w_{malg}]$ are learned, they are distributed to each algorithm to weight its decision or knowledge. For example, when an aggregation strategy, *AggFunctionProbability()* (Section 3.4.4), is used, the probability $p_{i,j}$ of learning algorithm i for action j is weighted and becomes

$$pw_{i,j} = w_i \times p_{i,j}, \quad i = 1, \ldots, malg, \quad j = 1, \ldots, mact, \tag{3.9}$$

Where, $malg$ is the number of learning algorithms, $mact$ is the number of action candidates, and w_i is the weight for algorithm i.

In this dissertation, all the aggregation algorithms adopt the weighting scheme, unless indicated otherwise.

3.4.3 Improved Majority Voting Method

Majority voting is efficient in the case of two candidates. When three or more candidates are under consideration, some pre-processes are necessary to convert the multiple candidates into "two-pairs" at different levels to ensure that in all situations one of the candidates is preferred by the majority or plurality.

For example, in the flight control system, there are more than two actions (the angle values of the elevator). Therefore, in each state, the number of action candidates is more than two. The problem is coped with as follow. First, the actions are divided into two groups: positive value or negation value, and the majority voting method or plurality method is employed to choose the preferred group. Then, the preferred group is separated into two subgroups: large or small, based on the values of actions, and a preferred subgroup is chosen. The division is repeated until the preferred action is chosen.

3.4.4 Aggregation Based on Homogeneous Learning

In this dissertation, homogeneous learning algorithms are viewed as the same RL algorithms with different values for one kind of learning parameter, such as learning rate, α, or eligibility trace discount rate, λ. For FA, if the similar featurizing methods are used, they are treated as homogeneous.

For homogeneous learning algorithms, aggregation is performed, not only in terms of the action (decision), but also of the knowledge (Q-values). Therefore, there are three possible aggregation functions for homogeneous learning algorithms.

- *AggFunctionAction()*

  Different learning algorithms provide their selected actions. *AggFunction-Action()* aggregates these actions and selects the aggregated action. This function works like below.

```
Function of Aggregation Based on the Actions
Obtain the decisions of action from each learner in the learning module;
Calculate the weights of each learner, based on some on-line information;
Weight each action;
Make a decision based on the aggregation algorithms such as WMV and
an exploration strategy;
Return the action.
```

Although aggregation based on the actions, which belongs to the field of decision fusion and selection, is the strategy generally more robust to the failures of individual learners [19], it uses less information provided by RL algorithms. Usually, this method works well when the selection probabilities of different actions are similar. In this dissertation, "robustness of a control policy" is defined as the ability that a control policy learned in a given environment can still work well when unmeasured noises and disturbances are introduced to the environment or the environment changes a little. The following aggregation functions make decision based on richer information.

- *AggFunctionProbability()*

  Each learning algorithm in the learning module indicates the probability of each action. This information is expressed as a probability matrix

$$P(s) = \begin{bmatrix} p_{1,1}(s) & \cdots & p_{1,k}(s) & \cdots & p_{1,mact}(s) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{l,1}(s) & \cdots & p_{l,k}(s) & \cdots & p_{l,mact}(s) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{malg,1}(s) & \cdots & p_{malg,k}(s) & \cdots & p_{malg,mact}(s) \end{bmatrix}, \tag{3.10}$$

  where $p_{l,k}(s)$ means the probability that algorithm l selects action k in state s. $malg$ and $mact$ are the same as those in Equation (3.9).

  The weights of different algorithms are presented as

$$W = [w_1, w_2, \cdots, w_{malg}], \tag{3.11}$$

then the weighted aggregated probability $PW(s)$ is

$$PW(s) = \frac{W \times P(s)}{\sum_{i=1}^{malg} w_i}. \tag{3.12}$$

AggFunctionProbability() calculates these probabilities and chooses a preferred action based on the probabilities. This function is described as follows.

```
Function of Aggregation Based on the Probabilities
Obtain the probabilities of different actions provided by each learner in
the learning module;
Calculate the weights of each learner based on some on-line information;
Weight the probabilities;
Select an action based on the probabilities;
Return the action.
```

This aggregation strategy is normally more reasonable than *AggFunctionAction()*. For example, as shown in Table 3.1, there are three learning algorithms and two action candidates at each time step, and each algorithm gives its probability about the two actions. Obviously, the aggregation results are different for *AggFunctionAction()* and *AggFunctionProbability()*. If each algorithm is assigned the same weigh, *AggFunctionProbability()* seems providing more reasonable result than *AggFunctionAction()* does. *AggFunctionProbability()* is more efficient when the selected probabilities of different actions are significantly different.

- *AggFunctionRank()*

  Similar to the *AggFunctionProbability()*, *AggFunctionRank()* can select an action based on the vector of ranks provided by the learning module and the weights obtained on-line.

- *AggFunctionQvalue()*

Table 3.1: Comparing of *AggFunctionAction()* and *AggFunctionProbability()*

	Probability of action A	Probability of action B	Action selected
Learning algorithm 1	0.6	0.4	A
Learning algorithm 2	0.7	0.3	A
Learning algorithm 3	0.1	0.9	B
AggFunctionAction()			A
AggFunctionProbability()			B

Different learning algorithms own different Q-values at each state, s. *AggFunctionQvalue()* aggregates these Q-values. This is the most general form of AMRL because Q-values can be used in many ways.

– Produce Actions

  ∗ Each algorithm selects an action based on its Q-values. Then, the case is the same as that in the *AggFunctionAction()*;

  ∗ If the learners are homogeneous, their Q-values can be weighted, and the preferred action is selected by using the ϵ-greedy technique.

– Produce Ranks

  Each algorithm ranks its preference of each action based on its Q-values. Then, the case is the same as in the *AggFunctionRank()*;

– Produce Probabilities

  Each algorithm provides the probabilities about each action based on its Q-values. Then, the case is the same as in the *AggFunctionProbability()*;

```
Function of Aggregation Based on the Q-values
Obtain the Q-values of each learner in the learning module;
Determine how to use the Q-values;
Choose an aggregation function and choose the preferred action;
Return the action.
```

3.4.5 Aggregation Based on Heterogeneous Learning

Heterogeneous means that different learners adopt different learning schemes for learning. For example, AC, Q-learning, R-learning, and SARSA-learning are heterogeneous. Their Q-values are in different scales. When different featurization methods, for example, the titling methods and normalizing methods, are used, they are treated as heterogenous. In these cases, the Q-values of different learners cannot be simply combined to make decisions. Therefore, only the strategies of *AggFunctionAction()*, *AggFunctionRank()*, and *AggFunctionProbability()* can be used in this case.

3.4.6 Aggregation Based on Sharing Information

In MASs, learning performance and quality can be improved by cooperating multiple agents. As explained before, AMRLS is a special example of MASs, and it can improve learning performance and quality by promoting cooperative learning. Through cooperation, agents can not only improve their individual performance, such as, the quality of the solutions and the efficiency in achieving solutions, but also achieve tasks that cannot be solved by individual agents. Therefore, AMRLS improves the performance of the whole system.

In this dissertation, information sharing is performed based on following two aspects.

- Input Information

  Learners share their position information and the sensations of the environment. From this information, learners enlarge their visual field and enhance their sensor information.

- Knowledge

  Learners share their experience or value functions. In this case, knowledge is updated more than one times at each time step, and the knowledge can be shared by all learners. With this information, the learning quality and system performance can be significantly improved.

The first method is used in the pursuit domain problem (Section 5.2). The second method is adopted in the cases of aggregation based on different values of a learning parameter (homogeneous learning), such as the pursuit domain problem (Section 5.2), cart-pole balancing system (Section 5.3) and mountain car problem (Section 5.4).

3.4.7 Aggregation Based on Different Values of a Learning Parameter

This is an example of aggregation based on homogeneous learning. In this example, different learners use the algorithms with the same learning scheme but different values of a certain learning parameter.

Several learning parameters, including learning rate, α, eligibility traces factor, λ, discount rate, γ, and exploration rate, ϵ, significantly affect learning performance. These learning parameters are strongly task and environment dependent. Typically, these parameters are set based on experiments and experience, but they are time and computation costly. Many researchers suggest fixing the parameters or changing the parameters according to a predesigned scheme, for example, to decrease learning rate α as [28]

$$\alpha^{n+1} = \frac{T_1}{1 + \frac{[n]^2}{1+T_2}}, \quad T_1 = 0.1,\ T_2 = 10^6,$$

or

$$\alpha^{n+1} = \alpha^n \cdot r, \quad r < 1,$$

where, n is the number of steps in an episode. However, the experimental results in Section 5.4 indicate that these two schemes come with difficulties. First, a larger or a smaller α causes learning failures, which implies that the decreasing scheme does not work well. Secondly, the suitable values of α are greatly influenced by another parameter λ, which should also be optimized. Even if an optimal value of α is determined by experiment, when the other parameters change, the “optimal value” is not optimal anymore. Therefore, the fixing scheme is also not feasible.

The eligibility trace factor, λ, is another parameter that affects RL performances. A larger value of λ weights more on the actual return than on the expected return value predicted by Q-function. A smaller value of λ emphasizes the prediction of Q-function more. In general, for a stronger dynamic system, a larger value of λ is preferred, and for a weaker dynamic environment, a smaller value of λ is more suitable. However, in many practical applications, RL is a kind of "lifelong" learning. It is hard to predict the dynamic characteristics of the environment; therefore it is impossible to hold an optimal λ value during the entire process of learning.

Instead of off-line pre-searching for the optimal value of each parameter, AMRLS copes with the difficulty by on-line aggregating the outcomes of multiple RL algorithms. In this aggregation situation, each learning algorithm learns its value function by using the same RL algorithm, but with different values for one learning parameter. This is a type of homogeneous learning method, and the aggregation strategies proposed in Section 3.4.4 can be applied here.

3.4.8 Aggregation Based on Different Learning Algorithms

This is an example of aggregation based on heterogeneous learning. In this example, different learners use different learning algorithms for learning.

In Section 3.3, a number of RL algorithms are discussed. Although all of them have been successfully applied to realistic tasks, none have been compared for the strengths and weakness of each algorithm. In fact, different algorithms exhibit different learning performances at different learning stages or different tasks (the experimental results are given in Sections 5.1 and 5.3). Aggregation of different learning algorithms is based on the learning aspect, i.e., using different basic RL algorithms. By aggregating different learning algorithms, AMRLS avoids searching for the optimal learning method. At the same time, because the aggregation is performed during the whole learning procedure, the weaknesses of the different learning algorithms at different learning stages are compensated for by the other algorithms.

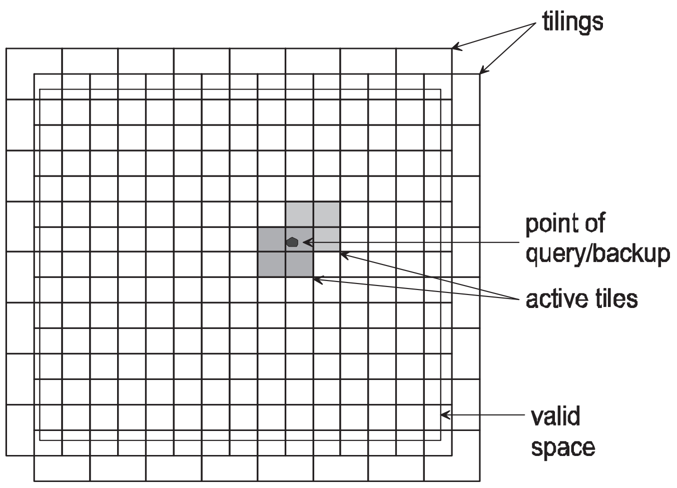


Figure 3.3: A two-dimension grid-like tiling

3.4.9 Aggregation Based on Featurization

Featurization is a critical step for FA. Here, each state, s, is translated into the set of features expressed in Equation (3.13). The features can be constructed from the state in many different ways: CMAC, RBF, tiling, and combination of the states, to name a few. The different featurization methods express the input state in different forms or with different values. These differences provide a foundation for AMRLS to aggregate. This aggregation procedure is demonstrated by using the tile coding method explained below.

Tile coding is a coding method that transfers a state from the input space to the feature space. Because of the lower computation load, tile coding is well suited for efficient on-line learning, such as RL. In tile coding, the feature value of an input state is expressed by the combination of a group of partitions, called tilings. The size and shape of partition can be arbitrary, large or small, in circles, grides, log strips, or even irregular. These factors affect the resolution of FA and determine the ability and nature of the generalization. The simple uniform grid-like tilings in Figure 3.3 are often used for parting to lower the computation cost. To produce the overlap, which produces the ability to generalize, each of the tilings is offset by a random value. The more the overlap of the two states, the stronger the generalization relation of the two states. To simplify the computation, binary expression is used to present the features, in which "1" signifies active and "0" denotes inactive. With this expression, if there are m tilings, then, at most, only m elements of the feature vector are active, or "1s"

There are two principle parameters for tile coding: the number of partition N_p, which determines the size of each tile, and the number of tiling N_t, which influences the overlapping of the tiles. If the number of the input variable is r, then the feature vector can be expressed as

$$\overrightarrow{F}_s = [f_s(1), f_s(2), \cdots, f_s(n)]^T, \tag{3.13}$$

where $n = N_t \cdot (N_p)^r$.

The combination of the two parameters determines the resolution of the FA. The higher the number of tilings, the denser the tiling are and the more accurate FA is, but the computation costs increase significantly. At the same time, a good generalization ability does not guarantee a good learning performance.

Different combinations of the two parameters present diverse learning performances. Therefore, AMRLS is applied again to improve the learning performance. In this case, each learner learns with the same learning algorithms and learning parameters but uses different combinations of parameters N_p and N_t. In turn, N_p and N_t present different feature values for the same input state and make different decisions on it. AMRLS aggregates the decisions and produces an aggregated action. The experimental results are presented in Table 5.11 in Section 5.4.

3.5 Summary of AMRLS

AMRLS is a new multiple learning framework based on the dynamic reinforcement learning, aggregation, and weights learning. By combining the advantages of individual learners and eliminating their disadvantages, AMRLS improves learning efficiency and qualities of the whole system.

3.5.1 Main Elements of AMRLS

AMRLS is summarized as follows.

- Two alternatively dynamic processes
  - dynamic reinforcement learning
  - dynamic aggregating and on-line weights learning
- Three sources for diversity
  - different RL algorithms
  - different values of learning parameters
  - different input information
- Four types of information for aggregation
  - decision
  - rank
  - probability
  - value function
- Five key components
  - a number of RL learners
  - a set of outputs from different RL learners
  - a weight-learning algorithm
  - a vector of weights for weighing different RL learners
  - a type of aggregation method

The above main elements form an aggregated multiple RL system, which can enhance the learning ability and improve learning performances of the overall system.

3.5.2 Advantages of AMRLS

The advantages of AMRLS are summarized as follows.

- Combine dynamic reinforcement learning and aggregation strategies together;
- Avoid searching for the optimal learning algorithm or the optimal value of some learning parameters;
- Develop the adaptive ability to a dynamic environment;
- Share information among different learners to improve learning qualities;
- Speed up the dynamic learning process by taking more efficient actions;
- Increase the steady-state learning qualities by smoothing the learning;
- Eliminate the risk of choosing bad actions;
- Provide fault tolerance ability;
- Improve the reliability of the overall system;
- Enhance the robustness of the system.

Chapter 4

An Improved Reinforcement Learning Method for AMRLS: a Hybrid RL Architecture

In Chapter 3, AMRLS and several aggregation strategies are proposed to improve RL performance. However, for many complicated tasks, the Value Function-based RL algorithms discussed in Chapter 3 do not work well. In order to enhance the learning ability of the individual learners in AMRLS, a new learning method is proposed in this chapter. First, an improved GARL algorithm is introduced. Then, it is applied to two control problems: the cart-pole system and the altitude control system, to show the advantages of this off-line RL approach. After that, a new hybrid RL method, HGATDRL, is proposed. By combining GARL with TDRL algorithms, HGATDRL improves the RL performance.

4.1 An Improved GA-based RL (GARL) Algorithm

GARL is an approach for searching the optimal control policy for an RL problem by using GAs. When GAs are used for RL problems, the potential solutions are the policies and are expressed as chromosomes, which can be modified by genetic operations such as mutation and crossover. Each element of a chromosome is called a gene and can be constructed in various forms, for example, a binary value, a real-value parameter of a function, a rule of a fuzzy system, or an exemplar expressed with a state-action pair [37]. Although binary chromosomes are the most popular, a new and more efficient coding method for GAs, real-value GAs (RGAs) coding method, has been proposed by Davis [21] and has received theoretical support from Muhlenbein and Schlierkamp-Voosen [61]. Real-value chromosomes have demonstrated many advantages compared with binary chromosomes. In RGAs, a chromosome is coded as a finite-length string of real values corresponding to the designed parameters. The real-valued representation is robust, accurate, and efficient because it is conceptually closest to the real design space. It has been reported that RGAs outperform binary-value coding GAs in many problems [38, 65]. In this dissertation, all GARL algorithms use the real-value coding method to construct a policy.

GAs are thought to be inherently an RL technique, according to the view of Whitley et al. [97]. They can directly learn decision policies without studying the model and state space of the environment in advance. The only feedback for GAs is the fitness values of different potential policies. In many cases, fitness function can be expressed as the sum of instant rewards, which are used to update the Q-values of value function-based RL algorithms.

As explained earlier, GARL is an example of DPS-based RL. Different from TDRL algorithms such as Q(λ)-learning and SARSA(λ)-learning, which try to obtain an approximate value function, for example Q-value function, GARL uses evolutionary operations to search for potential optimal policies directly without the need to construct a value function. TD-based RL assigns credit to a state-action pair (TD(0)) or all the state-action pairs visited (TD(λ)), according to the instant reward and the expectation value of the next state-action pair. GARL assigns credit to each policy by calculating the fitness values of each policy based on the entire experiment procedure.

Like conventional RL algorithms, GARL does not require prior knowledge and experience of the objective. The mathematical model is not important anymore.

The important work is how to define the fitness function and calculate the fitness values based on simulations. For example, in the altitude control system (in Section 5.5.2), the fitness function is defined as

$$fitness = \sum REWARD * (1 - abs(h/h_{limit})), \tag{4.1}$$

where, $REWARD$ is a positive instant reward; function abs gives out the absolute value of the input; h is the value of perturbation of the altitude, and h_{limit} is the limitation of h. Obviously, the smaller the altitude disturbance, the larger the fitness value. GARL searches for the optimal policy that maximizes the fitness value.

Although GARL methods have been successfully applied to many realistic tasks, few papers report the application of GARL in the flight control systems. The difficulty is the off-line property of GAs. In the rest of this chapter, GARL is discussed in detail and a hybrid RL method is proposed for building a cart-pole controller and an altitude controller for an aircraft.

4.1.1 Working Procedures of the Improved GARL Algorithm

The main procedures of GARL are summarized as follows.

```
GA-based Reinforcement Learning (GARL)
Initialize: Construct a population pool randomly with a given policy form.
            The size of population is $N_c$ (even number);
Repeat until the terminal conditions are satisfied
   (1) Couple: Couple the $N_c$ chromosomes randomly into $N_c/2$ groups.
               The two chromosomes in each group are called parents;
   (2) For each group
      (a) Form a family:
            Produce children from parents using crossover() and mutation().
            The number of children is $N_{children}$.
            Parents and children form a family;
      (b) Evaluate:
            Calculate the fitness value of each member of the family by evaluation();
      (c) Select:
            Select two chromosomes in the family with the highest fitness values;
      (d) Replace:
            Replace the old parents of the family in the population pool
            with the two chromosomes selected;
      End each group;
   (3) Update all the chromosomes in the population pool,
   (4) Start a new generation with the evolutive chromosomes.
End
```

Figure 4.1 presents the architecture of GARL and shows the working process of direct policy searching. The GARL algorithm is also called an off-line GARL module in the following hybrid RL method.

4.1.2 Construction of a Policy

One of the key works of GARL is the construction of a policy (chromosome). Several methods, such as neural network, fuzzy rules, or a table [60], can be used to form a

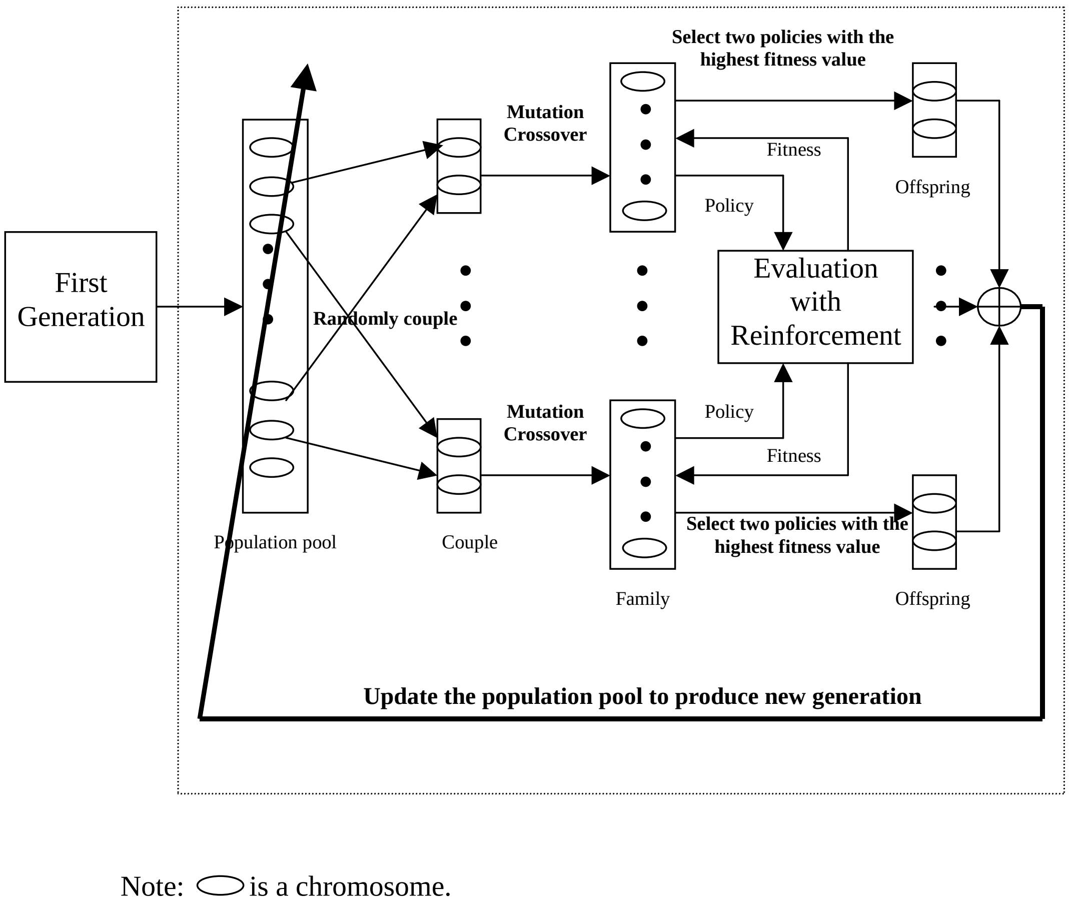

First Generation
Population pool
Randomly couple
Couple
Mutation Crossover
Mutation Crossover
Family
Select two policies with the highest fitness value
Select two policies with the highest fitness value
Fitness
Policy
Policy
Fitness
Evaluation with Reinforcement
Offspring
Offspring
Update the population pool to produce new generation
Note: is a chromosome.

Figure 4.1: Architecture of a GARL algorithm

chromosome. Two methods, the exemplar-based policy (EBP) and the coefficient-based policy (CBP), are used in this dissertation.

Exemplar-based Policy (EBP)

The exemplar-based policy (EBP) [37] optimization is introduced as a framework for DPS. In this framework, a policy has a case-based representation, that is, the exemplars of state-action pairs. The parameters (the real values of the state-action pairs) are directly optimized by RGAs.

For EBP, a chromosome is constructed in the following way.

```
Constructing the Chromosome for EBP-based GARL
Define the ranges, $R_{limit}$, for each state variable;
Determine the number of action candidates $N_a$;
Decide the length of a chromosome, i.e., the number of genes in a chromosome $N_g$;
Set the number of chromosomes $N_c$;
Repeat $N_c$ times
    Repeat $N_g$ times
        Select the state variables uniform randomly within the range $R_{limit}$;
        Pick an action randomly from the pool of action candidates and
        assign it to the selected state;
        Construct an exemplar $(s, a)$ with the values of the state-action pair.
    End
End
```

The number of genes, also called exemplars, N_g, in one chromosome (policy) can be set as 100, 200, or 400 in experiments. Although more exemplars give a more correct control policy, the searching requires too much time. In the experiments, all of these settings give good performances. N_g does not significantly influence the learning performance because of the construction of the exemplars. Even though the number of exemplars in one chromosome is not very large, the total number

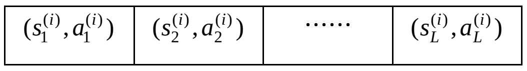


Figure 4.2: Chromosome

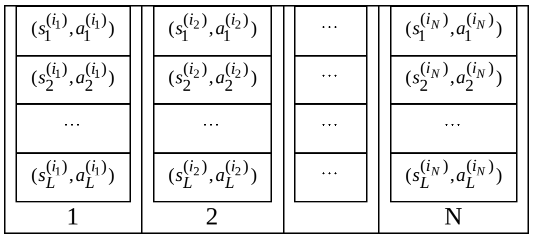

$(s_1^{(i_1)}, a_1^{(i_1)})$
$(s_2^{(i_1)}, a_2^{(i_1)})$
$\cdots$
$(s_L^{(i_1)}, a_L^{(i_1)})$
1
$(s_1^{(i_2)}, a_1^{(i_2)})$
$(s_2^{(i_2)}, a_2^{(i_2)})$
$\cdots$
$(s_L^{(i_2)}, a_L^{(i_2)})$
2
$(s_1^{(i_N)}, a_1^{(i_N)})$
$(s_2^{(i_N)}, a_2^{(i_N)})$
$\cdots$
$(s_L^{(i_N)}, a_L^{(i_N)})$
N

Figure 4.3: Representation of the population in the RGA

of exemplars in the population pool is large enough, and all the exemplars are evaluated and then selected with probability, which corresponds to the fitness value. Another parameter for GARL, the number of chromosomes, N_c, in the population pool, i.e., the population size, is set to be around 50 in experiments.

Figure 4.2 represents a chromosome with length L. It is obvious that the representation of a chromosome with real-value pairs outperforms that with binary-value because the real-value chromosome is shorter in length and is most similar to the real problems. Figure 4.3 is a representation of the population with the EBP method.

Coefficient-based Policy (CBP)

Besides the EBP method, it is natural to assume that a control policy can be constructed as a function of features (obtained from the state variables). If the coefficients of the function can be learned, the expression of the optimal control policy can be constructed. In this case, a policy (chromosome) is a set of coefficients of a function; therefore, this method is called a coefficient-based policy (CBP).

There are two kinds of CBPs used in the dissertation. One is the linear CBP (LCBP), in which a policy is expressed as the linear combination of features; the other is the non-linear (polynomial) CBP (PCBP), in which a policy is in the form of a high order polynomial of features. For example, in the cart-pole system (Section

5.3), there are four state variables: the position of the cart, x, velocity of the cart, $\dot{x}$, angle of the pole, θ, and angle rate of the pole, $\dot{\theta}$. If the state variables are used instead of features, the policy of LCBP is simply expressed as

$$sign(k_1 \cdot x + k_2 \cdot \dot{x} + k_3 \cdot \theta + k_4 \cdot \dot{\theta}), \tag{4.2}$$

because there are only two action candidates in each state. In this case, a chromosome is in the form of $[k_1, k_2, k_3, k_4]$.

For PCBP, if a third order polynomial is employed, a policy is expressed as

$$sign(k_0+k_1\cdot x+k_2\cdot x^2+k_3\cdot\dot{x}+k_4\cdot\dot{x}^2+k_5\cdot x\cdot\dot{x}+k_6\cdot x^2\cdot\dot{x}+k_7\cdot x\cdot\dot{x}^2+k_8\cdot\theta+k_9\cdot\theta^2+k_{10}\cdot\dot{\theta}$$

$$+k_{11} \cdot \dot{\theta}^2 + k_{12} \cdot \theta \cdot \dot{\theta} + k_{13} \cdot \theta^2 \cdot \dot{\theta} + k_{14} \cdot \theta \cdot \dot{\theta}^2). \tag{4.3}$$

The corresponding chromosome is $[k_0, k_1, \cdots, k_{13}, k_{14}]$.

4.1.3 Improved Crossover Algorithm

Crossover is a key operation of GAs. Crossover causes individuals to exchange genetic information. This EBP-based GARL has been inspired by Ikeda [37]. However, Ikeda used crossover only with a constant selection rate, r, which may cause some excellent genes to be destroyed by this crossover operation. Two approaches are proposed to deal with the problem.

1. Both parents and children consist of a family, and all the members of a family are evaluated and selected using the same criteria;

2. Children get the genes from their parents with different probability, r_c, which is proportional to the fitness values of the parents.

The first approach is applied by the architecture of GARL (Figure 4.1), and the second approach is adopted in the improved crossover algorithm. Consequently, the improved GARL method is developed for the sake of preventing excellent chromosomes from being destroyed and speeding up the convergence of GARL.

To explain the modified *crossover()* function and the new *mutation()* function (Section 4.1.4), the following notations are defined:

$$\overline{f} = \frac{f_f + f_m}{2}, \tag{4.4}$$

and

$$f^* = \max(f_f, f_m), \tag{4.5}$$

where, $\overline{f}$ is the average fitness value; f^* is the maximal fitness value; f_f and f_m are the fitness values of the parents, father and mother, respectively. Therefore, $\overline{f}$ is the average fitness of the parents, and f^* is the larger one of the parents'.

The selected rate, r_c, for the crossover is calculated according to the fitness value, f, of each chromosome. For example, the selected crossover rate, r_c, from the father is calculated as

$$r_c = \frac{f_f}{2\overline{f}}. \tag{4.6}$$

Clearly, if the selected crossover rate from the father is r_c, the selected crossover rate from the mother is $1 - r_c$,

The larger fitness value f implies that the policy owns more good exemplars. Equation (4.6) implies that the larger the fitness value of a parent, the higher the rate of the genes to be selected from the parent to form the genes of a child.

Figure 4.4 illustrates the operation of crossover. Here, P_i and P_j are the parents in one family, and P_c is a child produced by the two parents. If S is assumed to be a state set, and A is an action set, then a policy P consists of a set of exemplars (s_i, a_i), where $s_i \in S$ and $a_i \in A$.

Figure 4.4 is an example of a multiple-point crossover operation for EBP-based GARL. Each gene of a child is selected from its parents with the probabilities of r and $1 - r$, respectively, at a random position of the parents' chromosomes. If any two exemplars of a child are identical, one of them is removed, and a different one is chosen again. The algorithm of *crossover()* function is as follows.

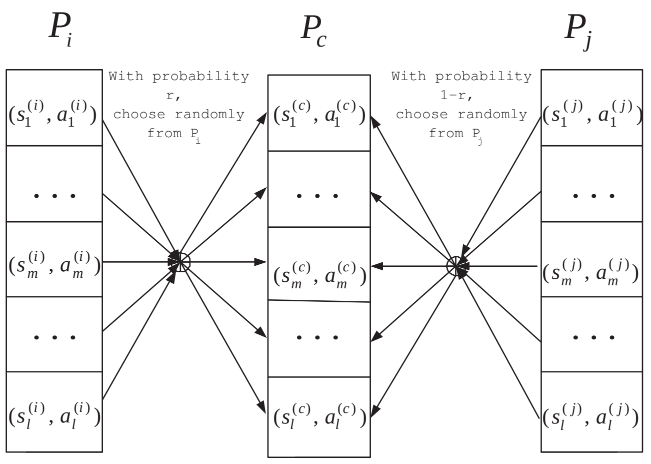


Figure 4.4: Crossover operation for two chromosomes using the EBP method

```
Crossover for the EBP-based GARL
Assign one chromosome as father and the other as mother.
Calculate the crossover rate, $r_c$, of the father using Equation (4.6).
Repeat for each child
    Repeat for each gene of a child
        Choose an exemplar from parents with probability $r_c$ and $(1 - r_c)$,
        respectively from a random position;
        If the exemplar has existed in the child's chromosome, re-choose
        another exemplar;
    End
End
```

If the CBP method is used for GARL, the crossover operation should be performed in the corresponding positions, indicating that the genes of a child should come either from the father or from the mother in the same position.

4.1.4 Improved Mutation Algorithm

The mutation operation encourages exploration of the search space to ensure that the algorithm escapes from the local optima. Ikeda [37] did not take the mutation operation. However, the experimental results prove that the mutation operation is crucial. A mutation algorithm is introduced into the GARL to improve the search ability.

The mutation rate, r_m, inspired by Wu [98] and Hu et al. [35], is defined as,

$$r_m = \begin{cases} 0.1(f_c^* - f)/(f_c^* - \bar{f}_c) & \text{if } f_c^* - \bar{f}_c \neq 0 \text{ and } f_c^* - f < f_c^* - \bar{f}_c \\ 0.1 & \text{otherwise,} \end{cases} \tag{4.7}$$

where f is the fitness value of a child, and f_c^*, $\overline{f_c}$ are the maximal and average fitness value of the whole children in the family. Equation (4.7) demonstrates that if the generation converges to a local optimum, that is, $f_c^* - \bar{f}_c$ is small, then the mutation selection rate, r_m, increases to avoid the local optimum in the following generation. At the same time, those chromosomes that have a larger fitness value will have smaller probabilities to mutate, so that the better chromosomes will be protected.

How to mutate a gene is an open question. For most binary chromosomes, mutation is performed by replacing the value in a randomly selected position with its complement. For real-value chromosomes, mutation can be performed in different ways. For example, in this improved EBP-based GARL, a TD-like method is applied for mutation. For the CBP-based GARL, the mutation is performed by changing the value of a randomly selected coefficient with an error of $\pm 5\%$. The *mutation()* function of the EBP-based GARL is expressed as follows.

```
Mutation for EBP-based GARL
Repeat for each child
    Calculate the mutation rate, $r_m$, using Equation (4.7).
    If (mutation should be performed)
        Choose an exemplar $(s_m, a_m)$ for mutation randomly;
        Take action $a_m$ in state $s_m$, and transit to a new state $s'_n$;
        Use the nearest neighbor method to find the exemplar $(s_n, a_n)$
        corresponding to the new state $s'_n$;
        Replace $a_m$ with $a_n$.
    End
End
```

4.1.5 Improved Evaluation Algorithm

The *evaluation()* function, one of the chief functions for GARL, evaluates the fitness value of each chromosome (policy) by testing the policy in a simulation/real environment. The only information provided by the environment is the evaluations of actions, i.e., rewards or punishments (reinforcement information). A fitness function is updated by using the reinforcement information, so this method is called GARL. The key part of *evaluation()* function is the definition of a fitness function. In real applications, it is not easy to define a fitness function. If a fitness function is chosen poorly or defined imprecisely, the GARL may be unable to find the optimal solution to a problem, or may end up providing a wrong solution. In this dissertation, the improved fitness function expressed in Equation (4.1) or (4.10) is applied. The procedure of *evaluation()* function is summarized as follows.

```
Evaluation Function for EBP-based GARL
Repeat for each family member (policy)
    Choose an initial state: a random start state is supplied to the system
    for evaluating a policy;
    Repeat until terminal conditions are satisfied
        Use the nearest neighbor method to find the exemplar $(s_t, a_t)$
        corresponding to the state;
        Take action $a_t$ in state $s_t$ and transit to a new state $s_{t+1}$.
        Replace the old state, $s_t$, with the new state, $s_{t+1}$;
        Obtain a reward and update the fitness value;
    End
    Return the fitness value of the policy.
End
```

4.2 Applications of the Improved GARL

In this section, the improved GARL algorithm is tested in two realistic systems: the cart-pole system and an altitude control system of an aircraft, by using the CBP method and EBP method, respectively, to demonstrate the efficiency of the GARL.

4.2.1 GARL with the Coefficient-based Policy (CBP) for the Cart-pole Balancing System

The cart-pole system is a famous test bed used in control fields. A detailed description of the cart-pole system is give in the Section 5.3. Here, only the application of the CBP-based GARL approach is described to demonstrate the design procedure and the effects of the CBP-based GARL approach.

A very important step in GARL is to design the fitness function. Usually, a fitness function should be related to the rewards (reinforcement information). In this example, a reward is set to be 0.1 for each successful step in a test episode. Therefore, the fitness function is

$$fitness = 0.1 \cdot i, \tag{4.8}$$

where i is the number of successful steps in an episode.

In order to keep the stable point close to zero, i.e., the position of the cart-pole system is in balance at approximately $x = 0$, a bonus for the stable state, x, is proposed to the fitness function. The bonus f_b is defined as

$$f_b = 50(1 - |\frac{x}{2.4}|), \tag{4.9}$$

where 2.4 is the limitation of variable x.

If the definition of successful control of the cart-pole system is to keep the system in balance for 100,000 continuous steps, then the total fitness value for the system is given as

$$fitness = \begin{cases} 0.1 \cdot 100,000 + 50(1 - |x/2.4|) & \text{if successful;} \\ 0.1 \cdot i & \text{if failed.} \end{cases} \tag{4.10}$$

Clearly, the longer the system remains in balance and the nearer the stable position stays approximately at zero, the larger the fitness value is.

Linear CBP-based GARL

In this case, a policy is defined by using the Linear CBP (LCBP), i.e., a linear combination of parameters $\theta(1), \theta(2), \cdots, \theta(n)$ and states. Actions are obtained from Equation (4.2). The control policy is trained by the improved GARL described in Section 4.1. The dynamic performance of the system is graphed in Figure 4.5. Obviously, the performance is very good because of the fast convergence speed and small stable values.

Polynomial CBP-based GARL

One of the issues of RL is that the learned policy is sensitive to the initial state of the experiments. Some of the good policies for a given initial state often do not

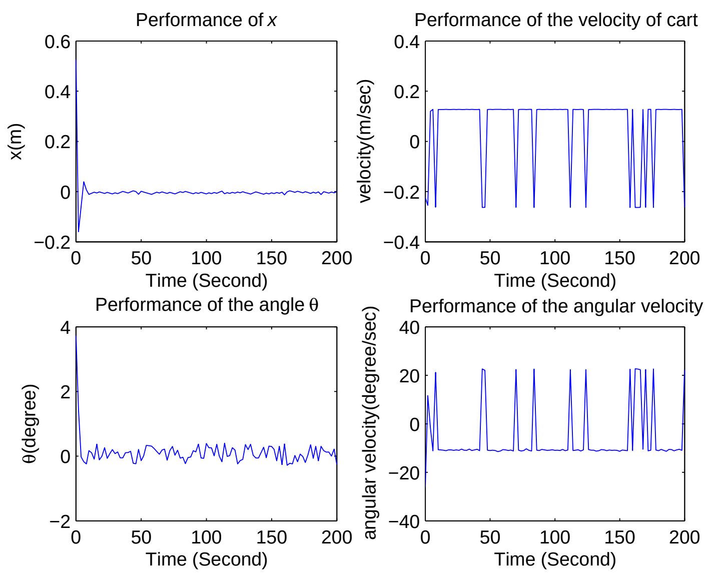


Figure 4.5: Performance of LCBP method for the cart-pole system

work when they are tested with different initial states. The most popular solution approach is to train a policy for many episodes with different initial states. However, more time and computation are required for this approach.

In experiments, the Polynomial CBP (PCBP) has been found to be much less sensitive to the initial states. In the case of PCBP, the method described in Section 4.1.2 is adopted to construct a policy. Therefore, the policy is expressed as a parameter vector $[k_0, k_1, \cdots, k_{13}, k_{14}]$, and actions are determined by Equation (4.3).

Figure 4.6 reflects the control performance of PCBP-based GARL. The result is exciting because of the excellent steady-state values. To evaluate the robustness of PCBP-based GARL to initial states, 100 initial states were randomly selected within the range of each state variable. The experimental results summarized in Table 4.1 are the average of 50 runs. Here, *Best* means the highest success rate in 50 runs with 100 random initial states. Similar definitions are given for $Worst$ and $Mean$. Whitley et al. have done similar experiments in [96]. Their results are rewritten in Table 4.1 as AC and GA-100. Although in the *Best* case, the results of the PCBP-based GARL are similar to the results of Whitley, in the $Worst$ case,

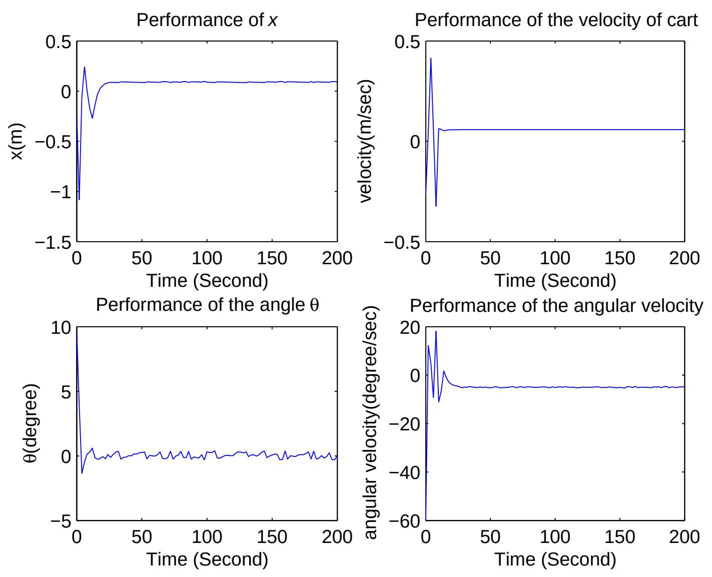


Figure 4.6: Performance of PCBP method for the cart-pole system

the results of PCBP-based GARL are much better than those of Whitley, and the average performances of the PCBP-based GARL are also better [42].

Further analysis shows that most of the failures in the PCBP-based GARL experiments are due to choosing initial states too close to the boundaries of the state variables range. In these cases, there really is no control policy that can keep the system in balance. If the initial states are restricted within a reasonable range, the success rate is very high, as the experimental results have proved. Thus the improved PCBP-based GARL algorithm can deal with the initial state problem

Table 4.1: Performance comparing for 100 random initial states, 5000 time steps

Method	Best	Worst	Mean	Std
AC	60	11	30.7	11.6
GA-100	71	4	47.5	14.2
LCBP	56	44	50.1	3.9
PCBP	71	51	61.4	5.2

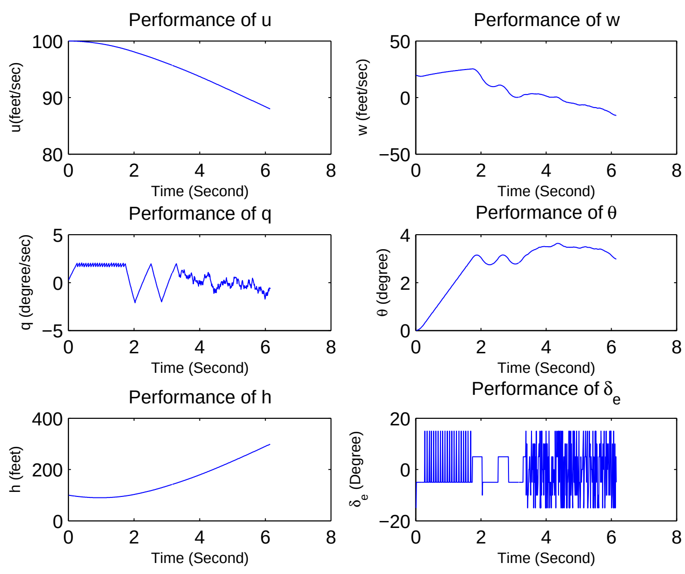


Figure 4.7: Performance of an altitude control system using the conventional TDRL method

very well.

4.2.2 GARL with the Exemplar-based Policy (EBP) for an Altitude Control System

The EBP-based GARL is tested in an altitude control system of an aircraft (explained in Section 5.5.2). The experimental results show that this system is very difficult to control with tabular-based RL algorithms because of the larger state space and poor exploration efforts, as represented in Figure 4.7.

In a typical RL problem with continuous states and actions, a bad action in a long sequence of actions affects little on the total reinforcement. In such a case, $Q(s, a_1)$ and $Q(s, a_2)$ may be very close, although actions a_1, a_2 are different in state s. The smaller the sample time step, the closer the Q-values of different actions in a state. Therefore, it is difficult to distinguish good actions from bad ones. This characteristic makes RL algorithms learn slowly in continued and long sequential tasks. A flight control system (Section 5.5) is a typical large continuous

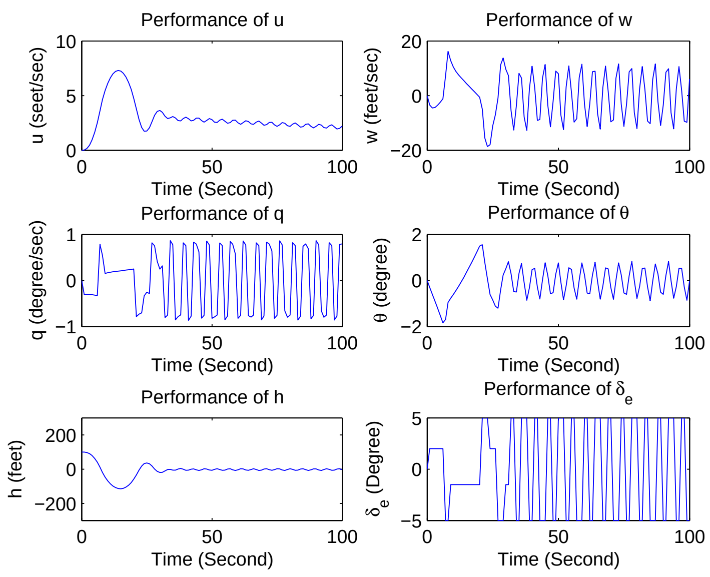


Figure 4.8: Performance of an altitude control system using the EBP-based GARL method

system; therefore, some special techniques should be developed to design an RL-based controller for an aircraft that works in a continuous situation.

In the following experiment, a policy is expressed as exemplars in the form of the state-action pairs, and the initial exemplars are set randomly. The optimal control policy is learned by the improved GARL algorithm described in Section 4.1. Figure 4.8 shows that the EBP-based GARL algorithm can control the altitude system by using the policy learned off-line.

4.3 Proposed Experience Cloning Module

For RL, the trial-and-error strategy is adopted to accumulate knowledge, but it is time consuming and sometimes risky. If some prior knowledge is introduced to guide the interaction in an environment, a learning can be faster and more efficient. Prior knowledge can provide some knowledge or experience about the environment but cannot provide dynamic instructions or evaluation during learning procedure. Such knowledge can provide only the initial guideline for learning, and the guideline might not be correct or enough. Therefore, prior knowledge is useful only for establishing the initial value for on-line RL algorithms. The procedure that transfers the prior knowledge learned off-line to the initial value for an on-line learning algorithm is called experience cloning.

Morales et al. [58] and Ng [63] have employed the cloning technique to establish the behavior of the environments. In real applications, the prior knowledge can be obtained in different ways, for example,

- By Knowledge and Experience

  This way requires the designers' background, knowledge, and experience of specific domains. When there are many variables and the state space is very large, this method is impractical.

- By Learning

  If some suitable initial knowledge can be provided by certain off-line learning methods, this basic, but limited, knowledge can be used as initial guideline and then be updated by on-line RL algorithms.

- By Simulating the Actions of Humans

  Humans can cope with many tasks but sometimes cannot clearly explain their decisions or actions. Their actions can be used as the exemplars for a policy. Therefore, some exemplars acquired from human simulations are adopted as initial knowledge. However, obtaining the whole of a human's experience in a large working space is difficult and expensive.

In this dissertation, the first two methods are relevant for the experiments. Figure 4.9 portrays the experience cloning module, and it is performed as follows.

```
Experience Cloning of EBP-based Reinforcement Learning
Establish a Q-value table $Q(S, A)$ based on the exemplars from GARL
        and the action set;
Initialize $Q(S, A) \leftarrow 0$;
  Repeat for all the exemplars of the policy
    The state of the exemplar is the state of the Q-value table;
    The action of the exemplar gives an initial value to correspond
    action in the Q-value table.
  End
Enlarge the Q-value table by
  Test the initial policy and record the procedure, i.e., the state-action pairs;
  Extend the Q-value table by adding the records.
Return the Q-value table with the initial values.
```

4.4 Proposed Hybrid RL (HGATDRL) Method

Although both TDRL and GARL algorithms have been successfully incorporated to solve some difficult RL problems, only a few studies [59, 60, 87, 96] have directly compared the two learning approaches. At this point, the question of when and why the TDRL or GARL method performs better remains open. Since GAs have multiple offspring, GAs can explore the solution space in multiple directions simultaneously. If one path turns out to be a dead end, GAs can easily eliminate it and continue to work in more promising directions. Therefore, GAs have more opportunities to find the optimal solution in an episode and are particularly well-suited for solving problems in the space of large solutions due to the parallelism. Another advantage of GAs is that they work very well in an environment of Partially Observable Markov Decision Process (POMDP) [81]. However, since GAs are typically off-line algorithms, most real-time tasks cannot be carried out by GAs. In contrast, TDRL methods, such as SARSA-learning and Q-learning algorithms, can update their policies on-line, but only explore the solution space in one direction at a time and are inefficient for solving problems with a large solution space. Here, on-line learning refers to learning and updating a policy while adopting the policy to perform the given task. The differences between the two learning approaches are summarized in Table 4.2.

Table 4.2: Differences between TDRL and GARL

	Policy Expression	Information Used	Modification Scheme	Learning Procedure	On or Off Line
TDRL	value function	TD error	learning	in serial	on-line
GARL	parameters	fitness value	searching	in parallel	off-line

In fact, the two RL approaches are by no means mutually exclusive, rather complementary [87]. Several hybrid approaches, including SAMUEL [29], ALEC-SYS [23], and NEAT+Q [95], have been proposed. However, most of these hybrid systems highlight the strengths of GAs that are provided by off-line learning approaches. In fact, on-line learning is a significant characteristic of RL, and its importance cannot be overestimated. Inspired by the advantages of TDRL and GARL algorithms, a hybrid RL architecture, HGATDRL that combines GARL and TDRL together, is proposed in this dissertation. The idea is that GARL learns an initial policy off-line, and then, the experienced cloning module transfers the policy to the initial values for TDRL, and TDRL updates and finetunes the policy on-line. The objective of this architecture is to enhance the learning ability of on-line TDRL algorithms by using the initial experience learned by off-line GARL algorithms.

4.4.1 HGATDRL Method and Architecture

Following are the main learning steps of the HGATDRL method.

1. Learn a policy with the GARL method by searching an optimal policy in a large policy space;
2. Freeze the policy after it exhibits an acceptable performance;

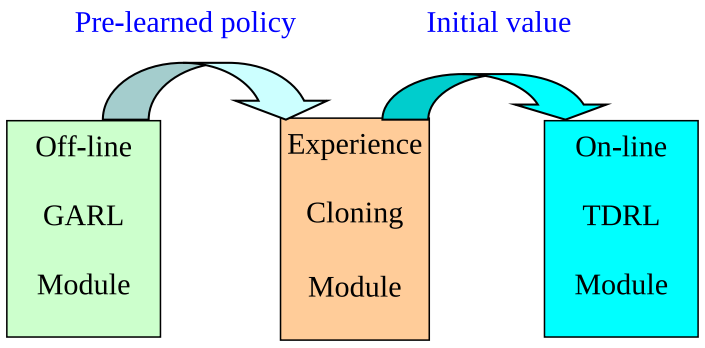


Figure 4.9: The HGATDRL architecture

3. Transfer the policy to a TDRL knowledge form with the experience cloning module;
4. Set the initial values for TDRL with the knowledge;
5. Update the policy on-line by using TDRL algorithms.

Figure 4.9 illustrates that the HGATDRL method consists of three modules, GARL, experience cloning, and TDRL modules. The off-line GARL module is displayed in Figure 4.1; the experience cloning module is explained in Section 4.3, and the on-line TDRL module adopts several TDRL algorithms described in Section 3.3.

Usually, it is difficult to perform the conversion between GARL and TDRL. EBP establishes a bridge to simplify the conversion. In the EBP method, a policy is composed of a set of exemplars, that is, state-action pairs. The exemplar-based policy can be easily converted into the initial values of a state-action table used by TD algorithms. Consequently, the EBP method is adopted in this dissertation to construct the policies for GARL.

Figure 4.8 gives the dynamic process under the control of this GARL controller. It is clear that the altitude can be controlled; although the performance is not very good. In fact, the GARL controller can be learned more generations to improve control quality. However, in order to use TDRL to learn the controller on-line later, only the policy is used as the initial value for TDRL.

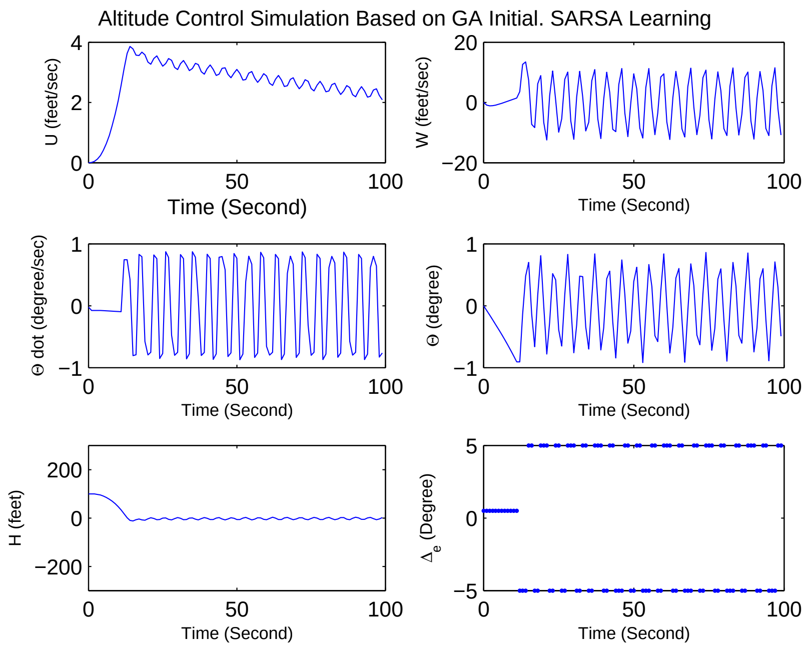


Figure 4.10: Performance of the altitude control system using HGATDRL

4.4.2 The Performance of the Altitude Control System with HGATDRL Method

Figure 4.10 provides the dynamic control process of the altitude system by using the control policy learned with the HGATDRL method developed in Section 4.4.1. Clearly, the dynamic performance of the altitude is much better than that in Figure 4.8 because it converges faster and with less overshoot. The fitness function used in the experiment is defined by Equation (4.1). Since the reward function is designed according to only the information of the altitude, the performances of the other variables such as the pitch angle and pitch rate do not change very much.

Chapter 5

Experiments and Simulation Results

Although RL algorithms can learn their policies without a model of the environment, researchers often use a dynamic model of the environment for simulation or testing the learned policies (controllers). RL is a trial-and-error procedure that might occasionally produce unacceptable results. It is risky to directly learn a policy in the real world without any pre-experience or knowledge. Consequently, a policy is often learned and evaluated in a simulation environment rather than in the real one directly. Different from model-based design techniques, the simulation for RL does not require an exact, complicated mathematical model. The accuracy of the model does not affect the learning results very much. The existence of a model of environment for simulation does not imply the existence of an exact mathematical model used for designing a controller or computing the optimal control policy mathematically.

This chapter presents several popular test-beds to evaluate the proposed AM-RLS. To explain the experimental results, several terms used in this chapter are defined as follow.

- Step

  A Step means a discrete time point at which an agent should make a decision.

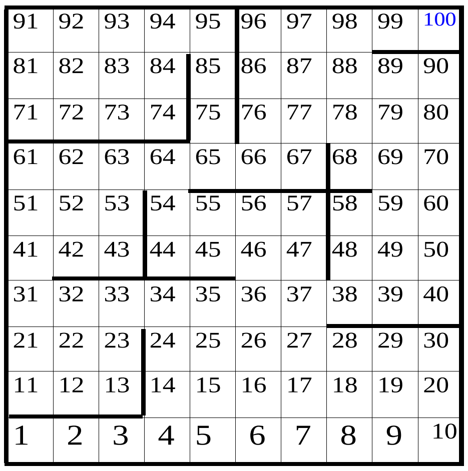


Figure 5.1: A maze world environment

- Episode

  An episode is the period of time from a start state to the state with a predefined terminal condition. An episode contains a number of steps.

- Run

  A run is composed of many episodes of learning until some criteria, for example, the convergency, are satisfied.

5.1 Maze World Problem

The maze world environment in Figure 5.1 is studied as an abstraction of the real navigation problem in mobile autonomous robots environments.

5.1.1 Experimental Setup

In this experiment, a robot equipped with three different RL algorithms (learners) attempts to discover the shortest path from a randomly given initial state to a given goal state. The view depth of all the learners is 1. The thicker black lines are obstacles, which represent walls that the robot cannot see or pass through. At each location, one of the four actions - moving right, left, up, or down - should be selected by each algorithm. If a learner selects an action that attempts to make the robot pass through the obstacles or the borders of the environment, the learner gives up the action.

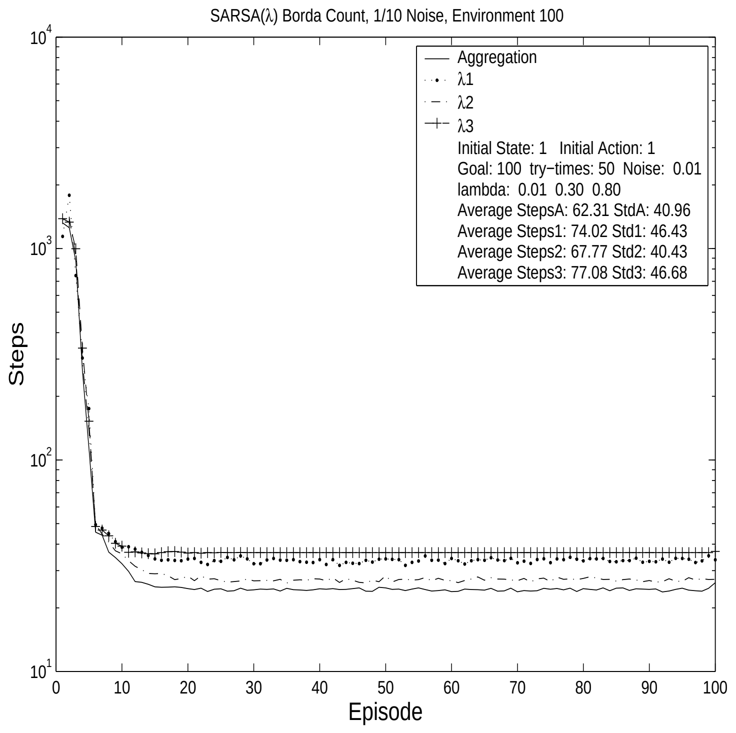
Two aggregation strategies, aggregation based on different values of parameter λ and aggregation based on different learning algorithms, are tested in this example. In the first case, both Q(λ)-learning and SARSA(λ)-learning algorithms are used to train the learners. These two learning algorithms provide similar results. To avoid redundancy, only the results of SARSA(λ)-learning with different λ values are presented. In the second case, AC(λ), Q(λ)-learning, and SARSA(λ)-learning algorithms are used by different learners respectively. To compare the "goodness" of different algorithms, two criteria are suggested. One is the number of average steps needed to find out the shortest path. The other is the rate at which each algorithm achieves the optimal policy, that is, finds the shortest path, within a given number of training episodes.

5.1.2 Results and Discussion

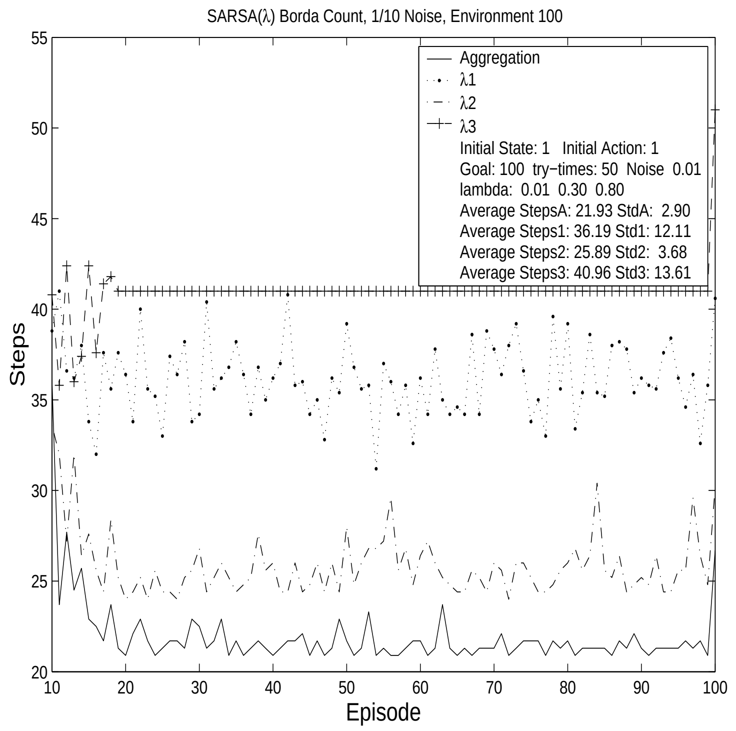
Aggregation Based on Different Values of λ

The results of aggregation based on Weighted Borda Count (WBC) and Weighted Majority Voting (WMV) with respect to $\lambda_1 = 0.01, \lambda_2 = 0.5$, and $\lambda_3 = 0.8$ are presented in Figures 5.2 and 5.3 in terms of the average number of steps needed to achieve the goal during the learning procedure. All the results are expressed as the average of 50 continuous runs. In each run, learning is repeated for 100 episodes until convergence.

Figure 5.2 (a) gives the number of average steps from the initial state *1* to the goal state *100*. In the beginning, learners have no knowledge about the environment and need many steps to achieve the goal. It shows that AMRLS learns faster than any individual algorithm. Figure 5.2 (b) reflects the learning qualities based on the WBC method after the learning algorithms converge. Obviously, the number



(a)



(b)

Figure 5.2: Aggregation using the WBC method

of average steps needed by AMRLS is less than those needed by the others. Thus, AMRLS has the best ability to find the optimal path within the restricted training episodes.

The experimental results are listed in Tables 5.1 and 5.2. Table 5.1 summarizes the average steps and standard deviation of different learning methods after convergence. In Table 5.1, results are presented in the form as: average number of steps to the goal/standard deviation of error.

Table 5.1: Results of the average steps

	AMRLS	$\lambda = 0.01$	$\lambda = 0.5$	$\lambda = 0.8$
WBC	21.9/2.9	36.2/12.1	25.9/3.7	41.0/13.6
WMV	24.6/1.8	35.9/12.6	27.6/5.9	46.4/9.0

Table 5.2: Results of optimization rate

	AMRLS	$\lambda = 0.01$	$\lambda = 0.5$	$\lambda = 0.8$
WBC	96/3.14	78/9.44	92/3.21	64/6.03
WMV	95/2.51	76/8.15	91/3.18	62/5.98

Table 5.2 provides the rate of finding the shortest path. In this example, each learning algorithm is trained only 100 episodes. In some episodes, learners cannot find the shortest path and converge to a sub-optimal path. To calculate the rate of finding the shortest path, each experiment is repeated 50 runs, and the rate is obtained based on the average of 50 runs. The results given in Table 5.2 are presented as: the rate (per cent) of finding the shortest path / standard deviation of error.

Figure 5.3 shows the learning qualities based on the WMC method after the learning algorithms converge. Different from the experiment shown in Figure 5.2, noise is introduced to the experiment. For example, in the episode 40^{th} an obstacle (noise) is added to the environment at a random position, and the obstacle is removed after 20 episodes. Figure 5.3 demonstrates that AMRLS adapts to a dynamic environment quickly.

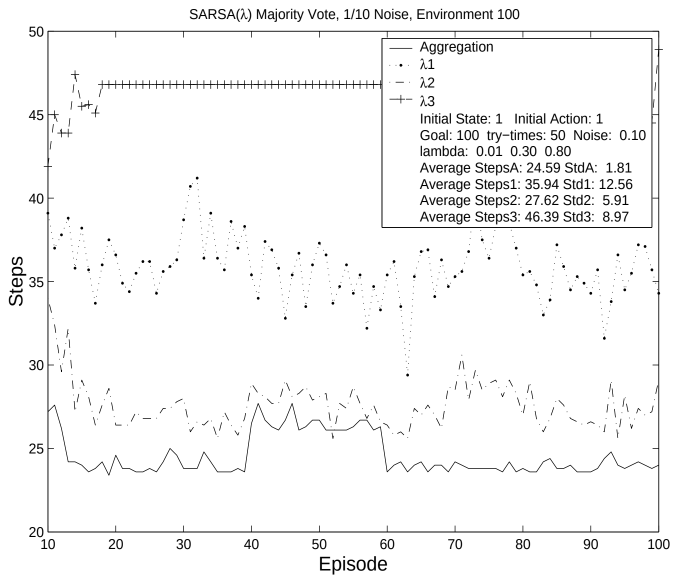


Figure 5.3: Aggregation using the WMV method

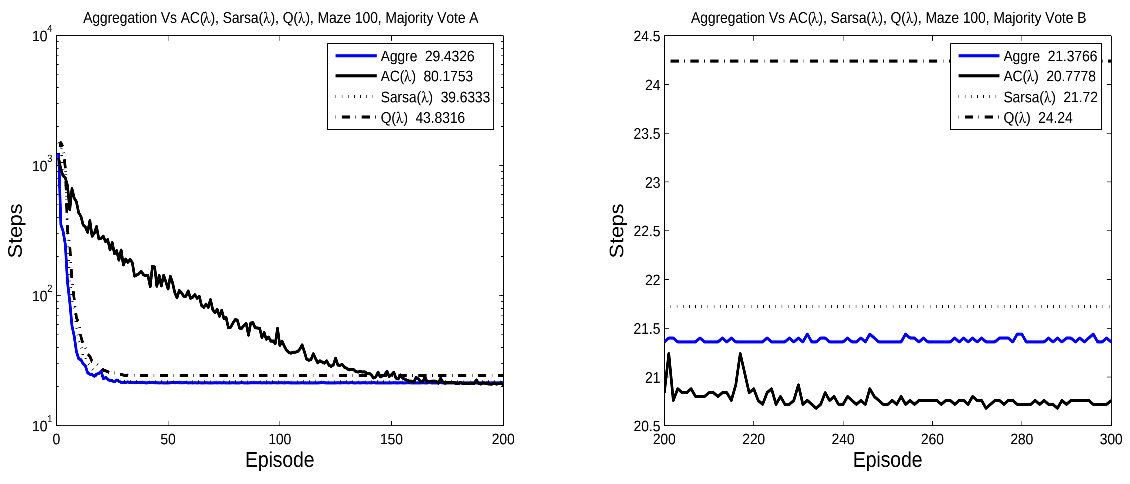
Table 5.3: Results of different learning algorithms with WMV

Method	AMRLS		AC(λ)		Q(λ)		SARSA(λ)	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Steps (Whole Period)	29.4	7.7	80.2	14.2	43.8	14.7	39.6	14.0
Steps (Stable)	21.4	0.024	20.8	0.088	24.2	0.00	21.7	0.00
CPU-time (second)	1.9	0.25	1.7	1.64	1.1	0.14	0.85	0.10

Aggregation Based on Different Learning Algorithms

In this experiment, each learner adopts one of the three learning methods: AC(λ), Q(λ)-learning, or SARSA(λ)-learning, with $\lambda = 0.5$. The aggregation is still based on the decisions of the different learners. Only the results of using the WMV method are given here.

Figure 5.4 signifies the average of the results for 50 runs. Here, one run contains 300 episodes so that each algorithm can converge. Figure 5.4 (a) shows the procedure from the beginning to the 200^{th} episode, and Figure 5.4 (b) provides



(a) (b)

Figure 5.4: Results of different learning algorithms with WMV for the maze world

the performances of the different algorithms after convergence. Obviously, AMRLS works much better in the initial stage, as seen in Figure 5.4 (a). The number of steps needed from the start state to the goal state drops dramatically in the initial stage. This characteristic demonstrates that AMRLS learns faster than other algorithms at this stage. In the steady stage (Figure 5.4 (b)), AMRLS is not the best learner, but is better than Q(λ) and SARSA(λ). AMRLS is not best because in a given state, the optimal action is no long unique in this example. There are some ties in optimal action selection. In this case, the current AMRLS just selects an action randomly. AMRLS needs improvement in this respect. The average performance over 50 runs is summarized in Table 5.3.

The experimental results also prove that the WMV and WBC methods work better than the simple MV and BC methods. The following techniques are used to determine the weights on-line.

- Determine the weights based on the number of times that each algorithm has been selected before.
- Determine the weights based on the characteristics of different algorithms during different learning stages.

> Different learning algorithms have different characteristics. For example, AC algorithm learns much better in the steady stage, but converges slowly. On the other hand, the SARSA-learning algorithm converges faster in the early state. Therefore, larger weights are given to SARSA-learning and Q-learning in the early stages and then weighs AC algorithm more in the steady stage.

In the reset of this dissertation, all the aggregations are based on the weights obtained on-line.

5.2 Pursuit Domain Problem

A pursuit domain problem consists of a discrete, grid-like environment and two types of agents: predators and prey. The goal of the predators is to capture the prey. Each predator receives sensory information about its environment, cooperates with the other predator, and moves around in the environment to capture the prey.

The pursuit domain problem is a popular example used to study different control strategies for Multiagent Systems (MASs) [31, 78, 85]. Here, this example is used to demonstrate that AMRLS can make cooperative learning by sharing and aggregating some information.

Figure 5.5 shows the environment used in this dissertation. It consists of a grid-like environment with $x \cdot y$ cells. Each cell owns a distinct position denoted by its x and y coordinates. The predators and prey have the same limited visual depth d.

5.2.1 Experimental Setup

The experimental environment can be described as follow. Two predators and one prey move in a 10×10 grid-like environment. Both predators and prey have four possible actions to choose from: moving up, down, left, and right. If the action that one agent chooses causes it to bump into the boundary, the agent does not move. The predators move based on the policy they learned with different RL algorithms,

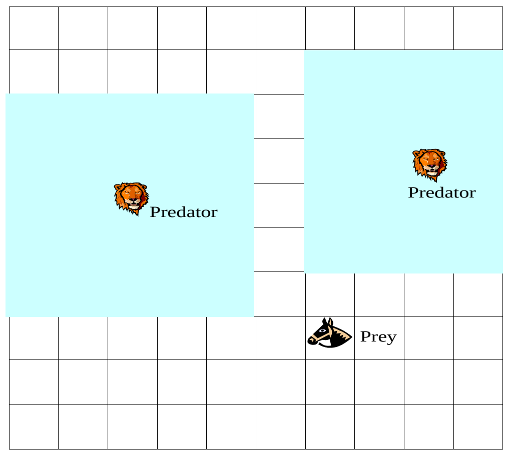


Figure 5.5: The pursuit domain world

but the prey moves randomly. Initially, predators and prey are placed randomly in the environment. A prey is captured when one of the predators moves to the same cell that is occupied by the prey.

When a prey is captured, the predator or predators involved get a +1 reward; otherwise, they receive a −0.1 reward for each movement. Each episode ends when the prey is captured. A run consists of many episodes until the algorithms converge. Results are averaged over 50 runs because of the random nature of RL.

Two methods are used to code the states in experiments.

1. Relative state [85]
   Given that the visual depth of every predator is d, then there are $(2d + 1) \times (2d + 1)$ cells in the visual depth. If the prey is in the lower and left corner of the visual field, the position is $(-d, -d)$; if the prey is in the upper and right corner, the position is (d, d). All the cells beyond the visual depth are denoted by one special state. The relationship between position and state is

$$s = x \times (2d + 1) + y + (2d \times (d + 1) + 1), \tag{5.1}$$

where, s is the state, x, y is the relative position of the prey to a predator. If the prey is on the right of a predator, x is positive; otherwise, x is negative. If the prey is on the up of a predator, y is positive; otherwise, y is negative.

Therefore, if a predator uses the sensory information obtained by itself, there are total of $(2d+1) \times (2d+1) + 1$ states for this code method. When the state of another predator is taken into account, there are $\big((2d+1) \times (2d+1) + 1\big)^2$ states.

2. Absolute state
There are still $(2d+1) \times (2d+1)$ cells in the visual field of a predator. Every cell may be occupied by a prey or not. So, there are $2^{(2d+1)\times(2d+1)}$ different states.

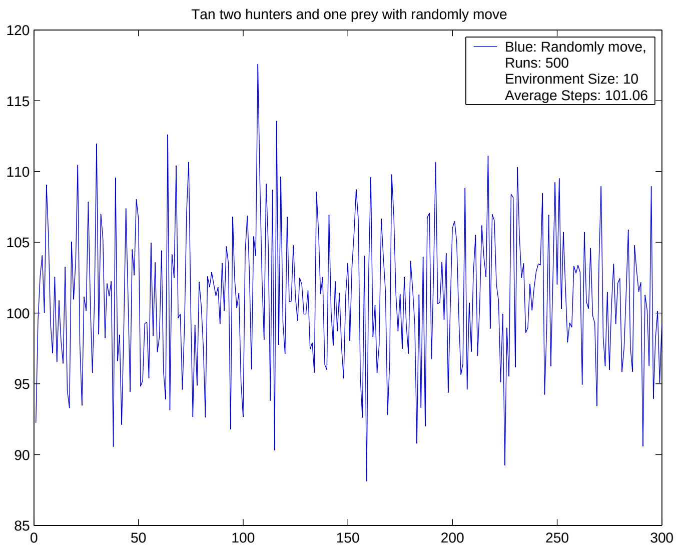
Compared with the second method, the first one has fewer states. It is more suitable for the tabular case. However it is hard to generalize because the adjacent state number may represent a totally different state; on the other, the adjacent state may have very different state numbers. Here, only the results of the relative state coding method are given.

5.2.2 Results and Discussion

A baseline performance, in which both predators and prey move randomly, is compared with different learning methods. Figure 5.6 gives the differences and shows that RL methods take much fewer steps, on average, than the random moving baseline. If there is no learning, the number of average step to capture the prey is 101.06 and it is random. By contrast, the number of average step for the learnable agents is 29.86, even though there is no cooperation. This shows that learning can greatly improve the agents' ability.

Two cooperation methods, sharing sensation and sharing policies [85] are adopted to improve learning qualities.

1. **Sharing Sensation**
Each learner has limited visual depth. If a prey is outside a predator's visual



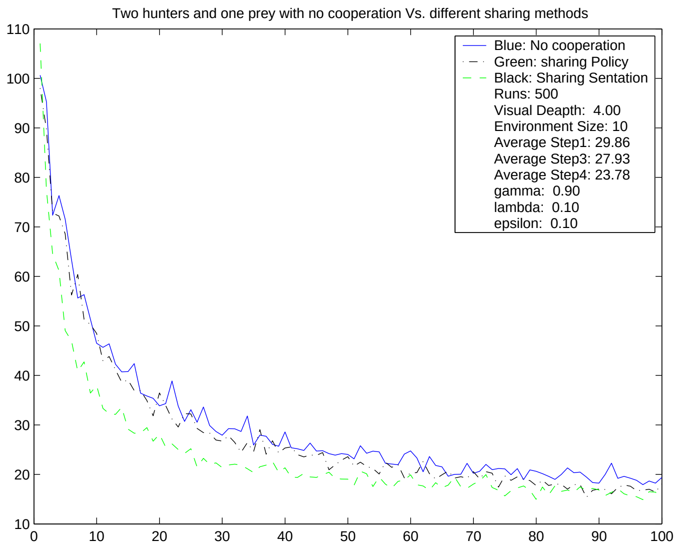


Figure 5.6: Results of two predators and one prey randomly moving vs. learning

field, the predator will move randomly. If there is more than one predator, they can share their sensations. At each step, predators exchange their sensations and calculate the position of the prey based on the sensations and the relative position of the predators. By this method, the visual field of every predator will increase and the number of random moving will reduce. As a result, the number of average steps to capture of the prey will decrease, too.

2. **Sharing Policies**
   In this experiment, predators are homogeneous and the relative state representation method is used to transfer the location to state, so the predators can use the same decision policy and contribute to update the same policy. Because in every trial the policy will be update N times (N is the number of predators in the environment), the policy will converge more quickly.

Figure 5.6 also shows that cooperation can be combined with RL, and the cooperative performance is better than the individual one. The average numbers of steps for cooperative methods are less than those of the independent agents. More importantly, the cooperative methods not only converge faster but also have fewer steps after the policy converges. Thus, the cooperative learning results are better than non-cooperative ones. The numbers of steps needed by different cooperation strategies to capture a prey are shown in Table 5.4.

Table 5.4: The average steps needed by different cooperation strategies

Sharing Sensation	Sharing Policies	No Cooperation
27.93	23.78	29.86

The Figure 5.6 demonstrates that the cooperation by sharing policy converges more quickly than by sharing sensation, because the former updates its Q-value twice in each trial. However, the sharing sensation method needs fewer steps to capture a prey after convergence because the sharing sensation cooperative method uses more information (the position of the other predator and its sensation) than the sharing policy method.

In this experimental environment, the aggregation technique is applied in two situations, aggregation of different RL algorithms and aggregation of different cooperation strategies. The first case is similar to the example of the maze world problem. Several learners using the same Q(λ)-learning algorithm with different λ values learn together, and several aggregation methods are adopted to make combination. Experimental results, the numbers of steps needed by different learners to capture a prey, are provided in Table 5.5, which supports that AMRLS is superior to individual learning algorithms.

Table 5.5: Comparison of AMRLS with Q(λ)-learning using different λ values

Method	AMRLS	$\lambda = 0.1$	$\lambda = 0.5$	$\lambda = 0.8$
MV	23.02	23.52	23.57	70.99
BC	23.87	23.53	23.67	72.44

The second case of this example demonstrates that AMRLS can also be used in the aggregation of different cooperation strategies. Since cooperation can be combined with RL algorithm, and aggregation of different RL algorithms can improve the learning performance, it is expected that aggregation of different cooperation strategies can achieve a better learning performance. Three cooperation strategies, independent (no cooperation), sharing policy, and sharing sensation are combined by the MV and BC aggregation techniques separately. Table 5.6 presents the number of steps needed to capture a prey.

Table 5.6: Comparison of AMRLS with different cooperative strategies

Method	AMRLS	No Cooperation	Sharing Policy	Sharing Sensation
MV	23.69	29.76	23.29	27.59
BC	22.31	29.65	23.43	27.33

5.3 Cart-pole Balancing System

The cart-pole balancing system, shown in Figure 5.7, is a classic testing system used to evaluate different control strategies. The system is described by the following second order equations:

$$\ddot{\theta}_t = \frac{mgsin\theta_t - cos\theta_t[F_t + m_p l\dot{\theta}_t^2 sin\theta_t]}{(4/3)ml - m_p l cos^2\theta_t}, \tag{5.2}$$

$$\ddot{x}_t = \frac{F_t + m_p l[\dot{\theta}_t^2 sin\theta_t - \ddot{\theta}_t cos\theta_t]}{m}. \tag{5.3}$$

The parameters are explained in Table 5.7.

Table 5.7: Parameters of the cart-pole system

x	the position of the cart
$\dot{x}$	the velocity of the cart
θ	the angle of the pole
$\dot{\theta}$	the angular velocity of the pole
l	the length of the pole = 0.5 m
m_p	the mass of the pole = 0.1 kg
m	the mass of the cart and pole = 1.1 kg
F	the magnitude of force = 10 N
g	9.8 m/s^2

5.3.1 Experimental Setup

This system is transformed to a discrete domain by using Euler's method with a sample rate of 0.02 seconds. Then, the discrete-time equations are simulated, and the state variables $x, \dot{x},\ \theta,\ \dot{\theta}$ are updated as

$$\begin{aligned}
\dot{x}(k+1) &= \dot{x}(k) + TAU \cdot \ddot{x}(k+1) \\
x(k+1) &= x(k) + TAU \cdot \dot{x}(k+1) \\
\dot{\theta}(k+1) &= \dot{\theta}(k) + TAU \cdot \ddot{\theta}(k+1) \\
\theta(k+1) &= \theta(k) + TAU \cdot \dot{\theta}(k+1)
\end{aligned}$$

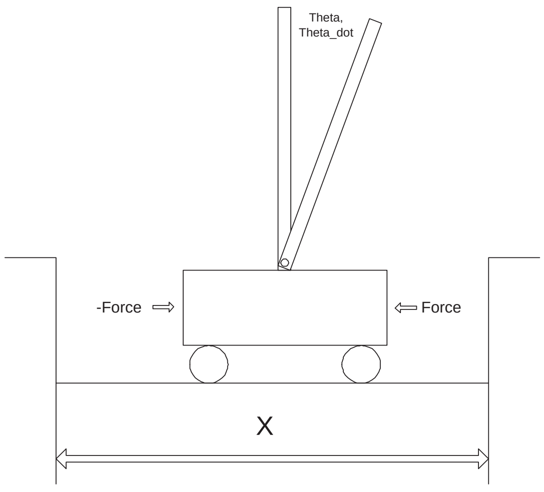


Figure 5.7: Cart-pole balancing system

where, TAU is the sample rate.

Many methods have been studied for the cart-pole balancing system. Barton et al. have proposed an RL method [9]; Hall and Pokorng have used a fuzzy logical control technique [30]; Absil and Sepulchre have suggested a hybrid control scheme [1]; Ramamoorthy and Kuipers have applied a qualitative heterogeneous control method [68]; and Whitley et al. have developed a genetic reinforcement learning algorithm [96] to control the cart-pole system. In this dissertation, Barton et al.'s method is extended to a multiple learning algorithms environment. In this situation, an AMRLS is equipped with three different RL algorithms (for example, AC(λ), Q(λ), SARSA(λ)), and attempts to keep the pole in balance for more than a given number of steps. At each state, AMRLS selects one of the two control forces, a positive or negative unit, to balance the system. The system is dispersed into 162 states [9]. By using some approximation methods, for example, the neural network method, the environment can be extended from a discrete domain to a continuous domain. For each training episode, the initial state is $\theta = 0$, $\dot{\theta} = 0$, $x = 0$, and $\dot{x} = 0$. If the angle of the pole is smaller than twelve degrees and the position of the cart is in the given range (-2.4, 2.4), the system is considered to be in balance. Otherwise, the system is considered to be out of control (fails). Two conditions are applied to end a run. One is that the pole stays in balance through more than 100,000 continuous steps, indicating that the training is successful. The other is that the number of failures (episodes) exceeds 1,000, which means the training has failed. If the system remains in balance, the immediate reward is zero, but, when the system fails, the reward is negative two. To trade-off the exploitation and exploration of RL, the $\epsilon-$greedy strategy is adopted. The number of training episodes, the total training steps, and the CPU-time used for training the controller are measured to evaluate the performance of each learning algorithm.

The aggregation methods explained in Section 3.4.3 are adopted, and the aggregation is done for two different cases:

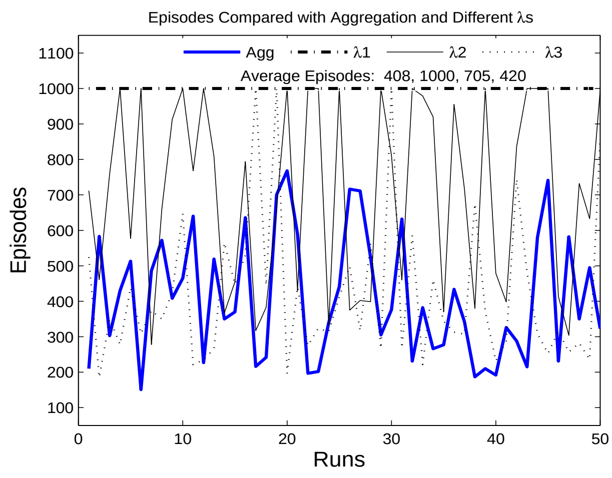
- learners with the same learning algorithm but different values for parameter λ;
- learners with different learning algorithms.

In the first case, both Q(λ)-learning and SARSA(λ)-learning algorithms are studied, and in the second case, AC(λ)-learning, Q(λ)-learning, and SARSA(λ)-learning algorithms are examined.

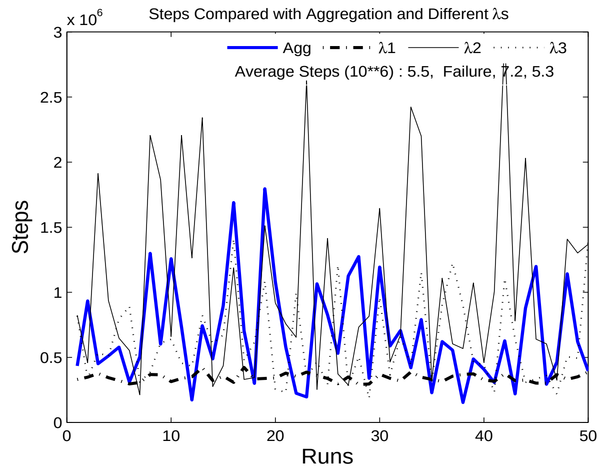
5.3.2 Results and Discussion

Figures 5.8 and 5.9 display the experimental results of aggregation with different values of parameter λ based on SARSA(λ)-learning and Q(λ)-learning respectively. The results are given in terms of episodes, steps, and CPU-time for training the cart-pole system to remain in balance. Clearly, both learning algorithms perform well. Various combinations of λs are tested, and the results are similar. Only the results of WBC with λ = 0.1, 0.5, and 0.8 are presented here. Obviously, smaller training episodes, steps, and CPU-times indicate a better training performance.

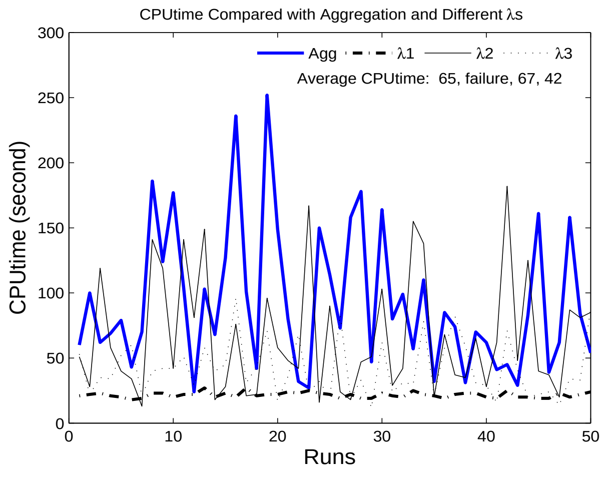
It should be mentioned that in Figures 5.8 and 5.9, the algorithm in which λ = 0.1 cannot successfully train a policy to keep the cart-pole system in balance, because the number of failures reaches 1,000, which means that the training has failed. The fact that the training steps and CPU-time of this algorithm are the least is irrelevant.



(a)

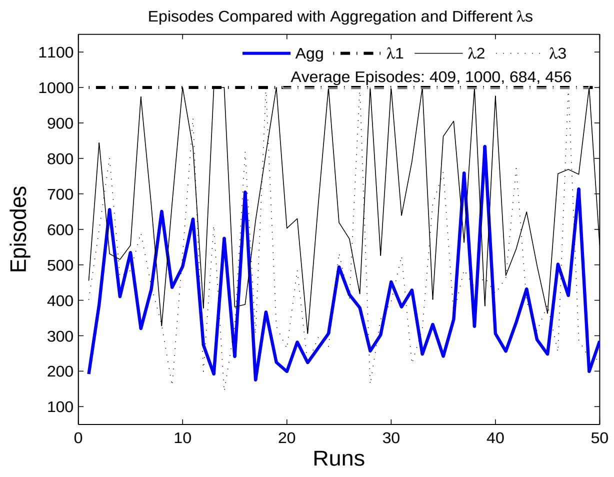


(b)

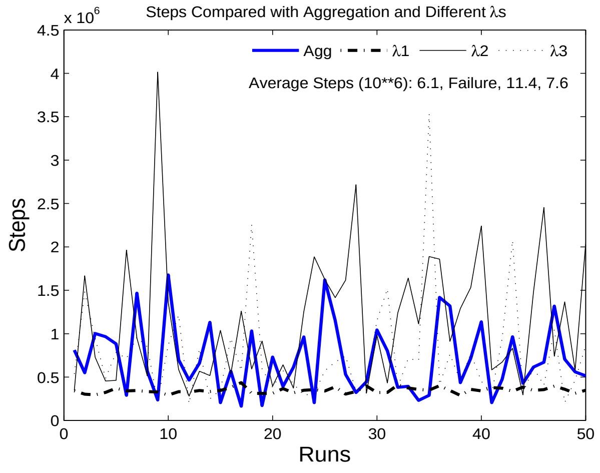


(c)

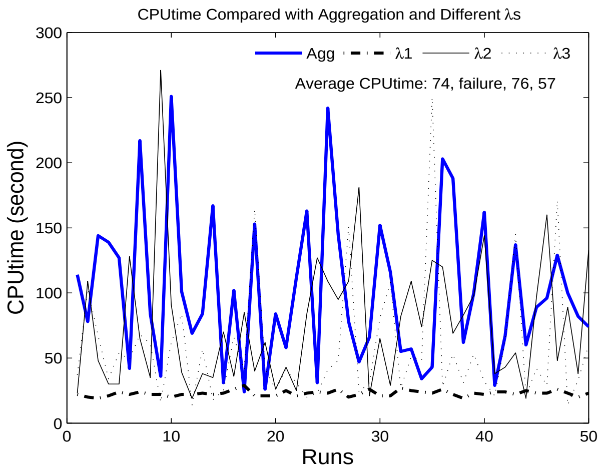
Figure 5.8: Aggregation with different λs using the WBC method and SARSA(λ)-learning algorithm



(a)



(b)



(c)

Figure 5.9: Aggregation with different λs using the WBC method and Q(λ)-learning algorithm

Table 5.8: Summary of results with different learning parameters (using WBC method)

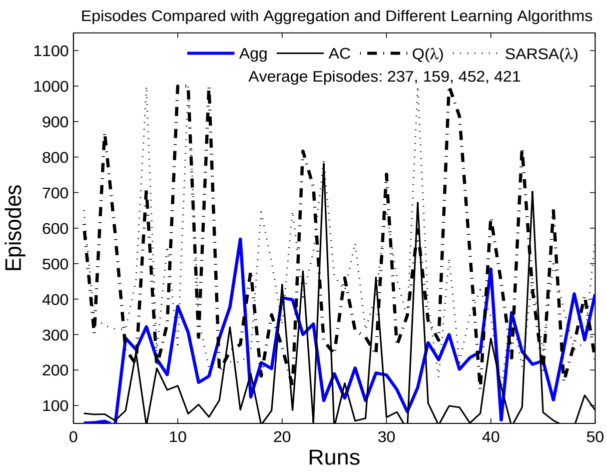
Method	AMRLS		$\lambda1 = 0.1$		$\lambda2 = 0.5$		$\lambda3 = 0.8$	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
SARSA(λ) Episode	408	172.2	1000 (failure)	0	705	265.7	420	208.5
Q(λ) Episode	409	200.2	failure	0	684	227.6	456	227.5
SARSA(λ) Step(10^5)	5.5	3.2	failure	0.3	7.2	4.2	5.3	3.3
Q(λ) Step(10^5)	6.1	4.5	failure	0.3	11.4	7.5	7.6	5.9
SARSA(λ) CPU-time	65	37.1	failure	2.1	67	45.7	42	21.6
Q(λ) CPU-time	74	43.2	failure	2.1	76	49.6	57	46.9

In addition, Figures 5.8 and 5.9 highlight the advantage of aggregation. The training procedure of AMRLS is smoother than those of the individual algorithms. For example, in Figures 5.8 (a) and 5.9 (a), each learning algorithm fails once (the number of failures is more than 1000), but the AMRLS is consistently successful. This advantage is very important in some cases because out of control is dangerous in real applications.

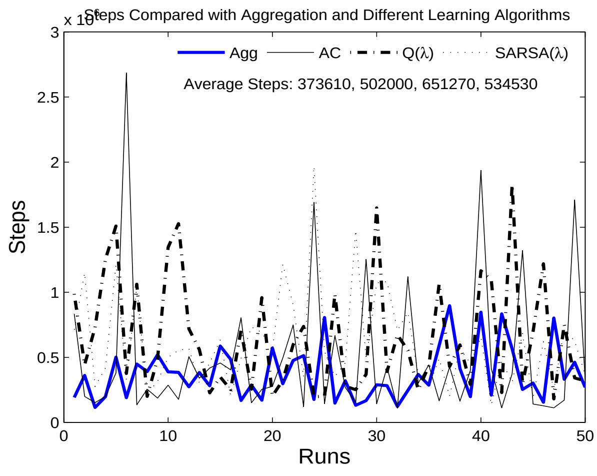
There is a cost, though, for aggregation. Figures 5.8 (c) and 5.9 (c) show more CPU-time is needed for aggregation. However, Figures 5.8 (c) and 5.9 (c) also show that the CPU-time used by the AMRLS is only a little longer than that of each algorithm, but much shorter than the sum of them. Thus, AMRLS is more efficient than the simple ensemble of the individual algorithms. In the experiments, the algorithms are learned in serial. If more processors are used, then AMRLS can perform in parallel. In that case, the training speed of AMRLS might be faster than that of individual ones.

Table 5.8 lists the performance of AMRLS and the algorithms with different values of λ in terms of the average and standard deviation of the episodes, steps and CPU-time. These results are the average of 50 runs.

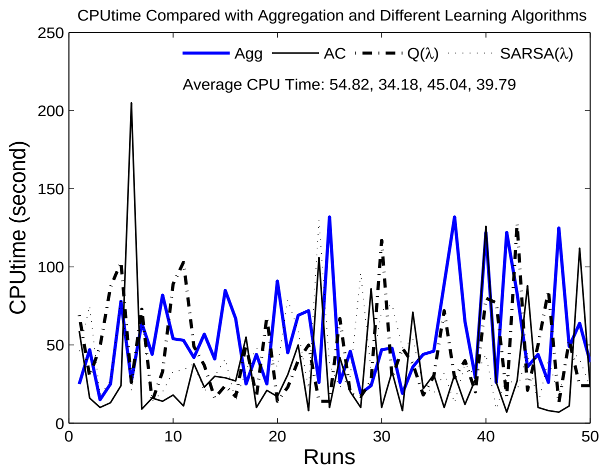
In the cart-pole experiments, the learners in the first case can be treated as homogenous, but in the second case, the learners require different algorithms and are,



(a)



(b)



(c)

Figure 5.10: Aggregation of different learning algorithms using the WMV method

Table 5.9: Summary of results with different learning algorithms (using WMV method)

Method	AMRLS		AC(λ)		Q(λ)		SARSA(λ)	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Episode	237	118.4	159	178.3	452	263.5	421	205.2
Step(10^5)	3.7	2.1	5.0	5.4	6.5	4.4	5.3	3.6
CPU-time(second)	55	31.3	34	27.9	45	30.3	40	24.0

therefore, heterogenous. Figure 5.10 gives the experimental results of aggregation with different learning algorithms. Only the results with $\lambda = 0.7$ and using WMV aggregation method are provided here. Figure 5.10 denotes that AMRLS performs better than the individual algorithms. It requires the fewest learning steps, fewer episodes, and most important, learns smoothly. The average performances of 50 runs of the second case is summarized in Table 5.9.

5.4 Mountain Car Problem

The mountain car problem is another popular test-bed to evaluate various control strategies. The task is to drive an underpowered car up a steep mountain slope as illustrated in Figure 5.11. The RL learner must learn to get momentum from the left slope and accumulate an initial velocity at the bottom of the mountain. Sutton and Barto [84] have pointed out that learning parameters α and λ significantly influence learning performance. The strategy of decreasing the values of parameters according to certain rules during the learning is unsuccessful because not all the values of the parameters can guarantee convergency. Sutton and Barto's experiments also support this conclusion. In this case, aggregation of the parameters becomes more useful and important.

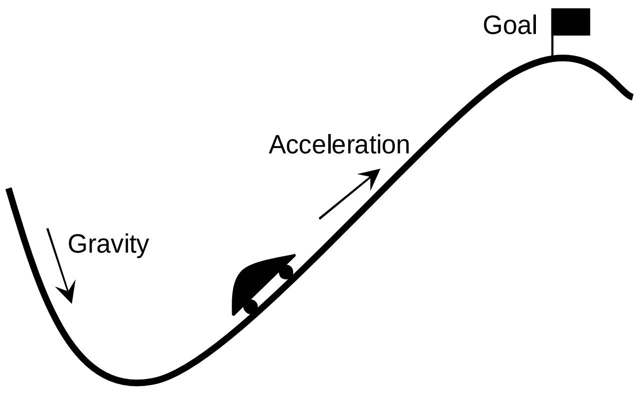


Figure 5.11: Mountain car problem

5.4.1 Experimental Setup

In this experiment, the tile coding method, which is described in Section 3.4.9, is applied to convert the continuous state variables to binary features. In the first and second experiments, the partition number of each variable is 9, and the tiling number is 10.

There are two state variables in this example, the position of the car, x_t, and the velocity of the car, $\dot{x}_t$. The dynamic equation of the problem is

$$\begin{aligned} x_{t+1} &= bound1\ [x_t + \dot{x}_{t+1}] \\ \dot{x}_{t+1} &= bound2\ [\dot{x}_t + 0.001a_t - 0.0025cos(3x_t)], \end{aligned} \tag{5.4}$$

where $bound1 = [-1.2, 0.5]$, $bound2 = [-0.07, 0.07]$, and a_t is the possible action, which can be -1, 0, or 1.

In the experiments, the reward is set to be -1 at each time step except at the state where $x_t = 0.5$, which means that the car arrives at the top of the mountain. In this case, the reward is 0, and one episode ends.

5.4.2 Results and Discussion

The first experiment demonstrates the effect of learning parameters α and λ on the learning performance of the mountain car problem. Sutton and Barto [84]

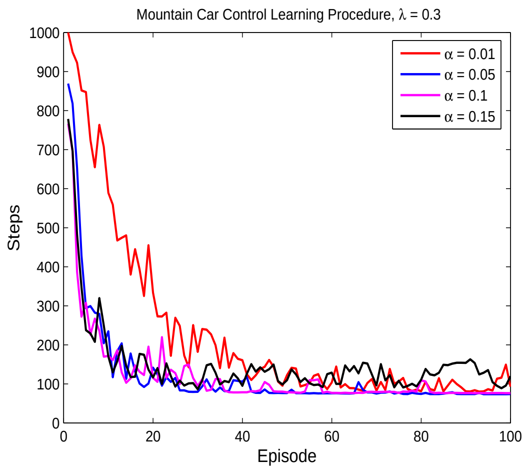
Table 5.10: Influence of α, λ on the early learning performance for the mountain car problem

Steps	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.1$	$\alpha = 0.15$
$\lambda = 0.3$	616.2	286.4	245.3	259.8
$\lambda = 0.5$	581.9	262.3	247.8	338.2
$\lambda = 0.8$	508.5	254.8	458.6	684.0
$\lambda = 0.95$	364.4	295.3	771.6	830.5

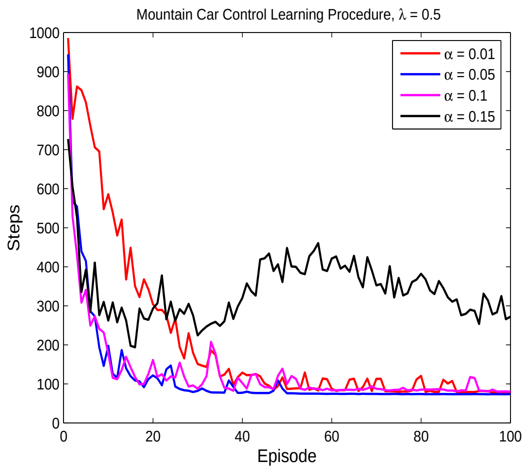
showed that with some parameter values, a learner may fail to learn. The same experiments are conducted in this dissertation with different combinations of the values of parameters α and λ, and a similar learning performance is observed. Strens and Moore [79] has pointed out that the control policy of the mountain car problem can be obtained trivially and the goal achieved finally; therefore, the early learning performance is chosen to evaluate control policies. The average number of steps in the first 20 episodes is used to indicate the learning performance, and the average is based on 50 runs.

Table 5.10 indicates that the number of average steps changes dramatically with different α, λ combinations. Moreover, it is difficult to find a rule to choose the values. For a larger λ, a smaller α seems better, but for a smaller λ, a larger α performs better. Figure 5.12 illustrates the entire learning process of the mountain car problem with different values for the learning parameters. It is also shown that although a larger α value results in a good performance during the early procedure, the steady performance is not good. Therefore, neither larger nor smaller values always works well.

The choice of a suitable value for learning parameters is time and computation consuming work, and the values are affected by many factors. In many applications, α can be changed according to a rule, but in the mountain car problem, both the smaller and larger α values result in poor learning performance. Therefore, neither fixed nor changed learning parameter values can produce a good learning performance. AMRLS can be employed to cope with this problem, as in the second experiment described here.



(a) $\lambda = 0.3$



(b) $\lambda = 0.5$

Figure 5.12: Learning performance with different values of the learning parameters

In the second experiment, aggregations are based on different values of learning parameters. Here, the results of aggregation based on different λ values are given. Figure 5.13 presents the learning performances of AMRLS and the individual learning algorithms with different values of λ. In this experiment, $\alpha = 0.01$ and $\lambda s = 0.3,\ 0.5$, and 0.8 respectively. This is the worst case shown in Table 5.10. It is evident that AMRLS is superior to each algorithm, not only in the early stage but also in the steady stage. The number of average steps taken in the first 20 episodes reduces to 343.8 (the average value is based on 30 runs), which is far fewer than those of the individual algorithms of 616.2, 581.9, and 508.5 respectively. Figure 5.13 reflects two more advantages of AMRLS: fast convergency and smooth learning procedure. AMRLS converges to the steady value at approximately 20 episodes, but for the individual algorithms, more than 35 episodes are needed. This indicates that AMRLS can learn faster than individual ones. The learning procedure of AMRLS is smoother than that of individuals, especially in the steady stage.

The third experiment presented here is the aggregation based on different numbers and sizes of tile. As mentioned in Section 3.4.9, a tiling method is used to generate the features from the input states. In this experiment, AMRLS is used to increase the robustness for the choice of tiling parameters.

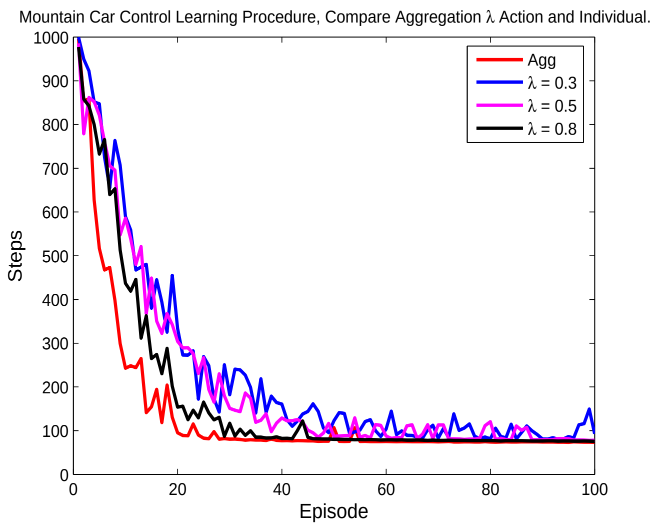


Figure 5.13: Comparing aggregated and individual learning performances

Tiling parameters N_p and N_t affect learning performance significantly. Different tile coding causes diverse learning performances for RL with different learning parameters, such as α. AMRLS is applied again to improve the learning quality by aggregating several learners.

In the experiments, three groups of N_t and N_p, $(N_t, N_p) = (10, 9)$, (10,5), and (5,9), are selected respectively. For each learning algorithm, input states should be converted into features. AMRLS takes action according to the aggregated decision. Tables 5.11 to 5.13 list the experimental results. The pair $(N_t, N_p) = (10, 9)$ refers to an algorithm that uses tiling parameters $N_t = 10$ and $N_p = 9$. The experimental results are the averaged values of 30 runs.

The first column of these tables provides the parameter values used in the experiments. All RL algorithms are SARSA(λ) algorithms. The second column indicates the learning ability of an algorithm. The higher success rate implies a better on-line learning ability. The third column represents the convergence speed. The fewer the steps, the faster the convergence speed. The Column 4 shows the performance in the whole procedure.

From the three tables, the following conclusions are drawn.

Table 5.11: Influence of the partition size on the learning performance for the mountain car problem with $\lambda = 0.8$ and $\alpha = 0.01$

$\lambda = 0.8$ $\alpha = 0.01$	Success Rate in the First 100 Episodes	Average Steps in the First 20 Episodes	Average Steps in the First 100 Episodes
$(N_t, N_p) = (10, 9)$	95 ± 0.9	508 ± 19.0	172 ± 4.6
$(N_t, N_p) = (10, 5)$	98 ± 0.4	211 ± 7.6	102 ± 1.5
$(N_t, N_p) = (5, 9)$	89 ± 1.0	716 ± 17.7	249 ± 7.5
AMRLS	93 ± 0.8	539 ± 33.1	183 ± 9.8

Table 5.12: Influence of the partition size on the learning performance for the mountain car problem with $\lambda = 0.8$ and $\alpha = 0.05$

$\lambda = 0.8$ $\alpha = 0.05$	Success Rate in the First 100 Episodes	Average Steps in the First 20 Episodes	Average Steps in the First 100 Episodes
$(N_t, N_p) = (10, 9)$	98 ± 0.9	273 ± 53.6	125 ± 14.2
$(N_t, N_p) = (10, 5)$	98 ± 0.3	244 ± 19.3	102 ± 11.5
$(N_t, N_p) = (5, 9)$	97 ± 0.7	302 ± 25.1	129 ± 10.3
AMRLS	98 ± 0.8	299 ± 23.0	128 ± 6.5

Table 5.13: Influence of the partition size on the learning performance for the mountain car problem with $\lambda = 0.8$ and $\alpha = 0.15$

$\lambda = 0.8$ $\alpha = 0.15$	Success Rate in the First 100 Episodes	Average Steps in the First 20 Episodes	Average Steps in the First 100 Episodes
$(N_t, N_p) = (10, 9)$	42 ± 15	718 ± 143	705 ± 127
$(N_t, N_p) = (10, 5)$	35 ± 8.9	629 ± 106	754 ± 168
$(N_t, N_p) = (5, 9)$	89 ± 2.1	302 ± 8.1	207 ± 199
AMRLS	75 ± 16	413 ± 149	351 ± 104

- Larger partition number N_p does not mean a good learning quality. $N_p = 5$ works better than $N_p = 10$, which was used by Sutton and Barto [84], in the experiments.
- The tiling parameters have significant influences on learning performance, especially when α is larger (Table 5.13).
- The choice of tiling parameters N_t, N_p depends on the setting of learning parameter α. For example, $(N_t, N_p) = (10, 5)$ works best if $\alpha = 0.01$ (Table 5.11), but performs worst if $\alpha = 0.15$ (Table 5.13). It is difficult to coordinate these parameters, especially when the value of α is decreased during the learning process.
- AMRLS exhibits a good robustness with respect to these parameters. Although AMRLS does not work best in all cases, it is better than the worst. More importantly, unlike the individual learning algorithm that may work very well in some situations but may work very poorly in another situation, AMRLS always works well; therefore, it avoids the risk of performing very poorly and does not cost time to search for the "optimal" values of tiling parameters.

5.5 Flight Control Systems

Aircraft in flight exhibit random and unpredictable behavior due to unexpected disturbances of the atmosphere [3]. Designing automatic aircraft controllers is a challenging research problem because of the inexact model, high-dimensional motion equations, noisy and unexpected environment, and scarce knowledge. The first automatic flight control device in the world was designed by Eimer Sperry and his son Lawrence Sperry in 1912 to maintain the altitude of flight. Then, in 1914, Lawrence Sperry demonstrated the automatic flight controller (autopilot) at the Paris air-show. Since then, autopilots have been developed into Automatic Flight Control Systems (AFCS), which is used to improve the flight quality and enhance the stability of aircraft. AFCS can help pilots guide a plane to a destination, reduce the work load of pilots, and provide a comfortable flight for passengers.

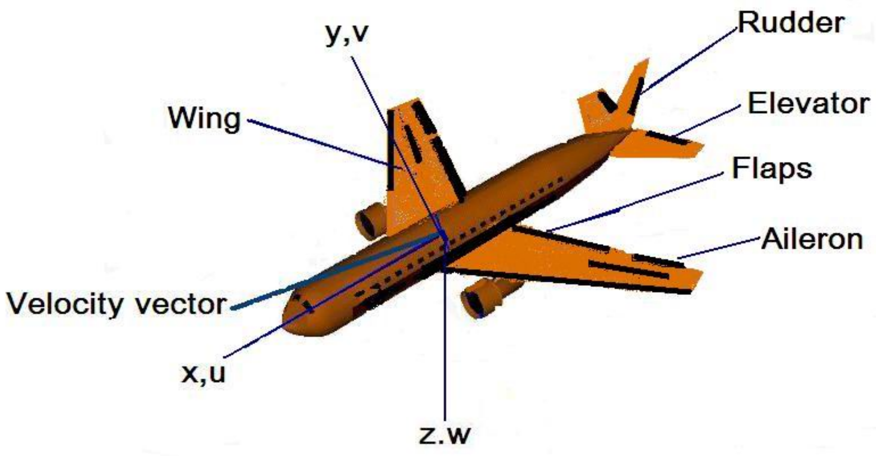


Figure 5.14: Coordinates of an aircraft

Figure 5.14 shows the coordinates of an aircraft. Normally, an aircraft is controlled by three control faces, the elevator, rudder, and ailerons. For fighters, more control faces are introduced, such as flaps, leading edge slats, and horizontal and vertical canards.

A Boeing 747 aircraft has been selected as the research object for this dissertation. The dynamic model of the aircraft is provided in [13]. To simplify, the linearized small perturbation equations are discussed, and the performances of the controller (policy) are analyzed within a given range. The linearized equations of motion for the Boeing 747 are expressed by eight variables but can be separated into two fourth-order sets. These two sets represent the perturbations in the longitudinal and lateral motions, respectively. The speed in the direction of axial x, u, speed in the direction vertical z, w, and pitching (θ, q) motion form the longitudinal motion, whereas the speed in the direction of axial y, v, rolling (ϕ, p) and yawing (r, β) movements form the lateral motion. Although there is a small amount of coupling between the lateral motion and the longitudinal motion, it is usually ignored. Thus, the motion equations can be treated as two decoupled fourth-order sets for designing controllers for an aircraft.

Although a controller can be designed and analyzed based on the linearized equations, to do so requires a comprehensive understanding of aircraft and control theory, significant experience in controller design, and the exact dynamic model. Unfortunately, the real motion equations of aircraft are nonlinear, and the environment is dynamic and unexpected; therefore, conventional control methods such as linear feedback control [26], quantitative feedback theory [34], and optimal quadratic control [56] cannot be adopted to design controllers with a good performance.

Generally, in designing a controller for an aircraft, researchers often face the following difficulties:

1. A complicated or non-mathematical model

   Normally, the model of aircraft is non-linear, high dimensional, strongly coupled, and stochastic. It is difficult to establish a precise mathematical model for an aircraft. In this case, traditional techniques, such as linear feedback method, root trace method, adaptive control, and quantitative feedback theory, which need the theoretical model to design a controller, do not work well.

2. An unpredictable environment

   An aircraft often flies in unpredictable environments. Changes of pressure, gusts of wind, and other disturbances all affect the movement of an aircraft in unexpected ways. Therefore, an elaborate, predefined, fixed control system may be worthless in this case. Instead, an online learning algorithm is needed to learn a control policy in real time.

3. Insufficient training data

   For a complex flight control problem, it is impossible to know all the desired behaviors in the whole range of a flight envelop. Therefore, supervised learning methods cannot be applied in this case.

4. Incomplete state information

   In many cases, state information is incomplete because of a lack of sensors. This is the case of the Partially Observable Markov Decision Process (POMDP).

The first and second difficulties make the traditional model-based predesign control methods inefficient for designing an effective AFCS. Researchers have developed various model-free, on-line parameter adjustment methods to learn a control policy for controlling aircraft. For example, a neural fuzzy network can use supervised learning algorithms to learn control policy without the aircraft model [91]. However, this method needs a great deal of training data that should be complete and correct. This requirement causes the third difficulty. In aircraft control applications, detailed and precise training data may be very expensive or even impossible to obtain. Instead, in most cases, the basic information obtained is only the evaluations of actions. RL is the most promising technique for designing a flight control system because it does not need a mathematical model or very much training data and it can learn on-line.

However, conventional tabular-based RL cannot be used to design a high quality flight control system because of the continuous states and sparse examples. In this section, first, a pitch controller is designed by using the technique of RL and CMAC. Then, the proposed hybrid learning method, HGATDRL, is used to design an altitude control system to keep the altitude of an aircraft constant under an initial perturbation.

5.5.1 Pitch Control System

Pitch control is a basic control of AFCS. Pitch is defined as a rotation around the lateral or transverse axis, which is parallel to the wings, and is measured as the angle between the direction of speed in a vertical plan and the horizontal line. Changes of pitch are caused by the deflection of the elevator, which rises or lowers the nose and tail of the aircraft. When the elevator is raised (defined as negative value), the force of the airflow will push the tail of aircraft down; hence, the nose of the aircraft will rise and the altitude of the aircraft will increase. One of the goals of a pitch control system is to control or help a pilot to control an aircraft to keep the pitch attitude constant, that is, make the aircraft return to the desired attitude in a reasonable length of time after a disturbance of the pitch angle, or make the pitch follow a given command as quickly as possible.

Experimental Setup

As mentioned above, for most aircraft, pitch attitude is controlled mainly by elevator δ_e. In experiments, a linear longitudinal perturbation equation given in [53] was used to express a pitch system.

$$\dot{X} = A \cdot X + B \cdot \delta_e;$$

in details

$$X = \begin{bmatrix} \alpha \\ q \\ \theta \end{bmatrix},$$

$$A = \begin{bmatrix} -0.313 & 56.7 & 0 \\ -0.0139 & -0.426 & 0 \\ 0 & 56.7 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.232 \\ 0.0203 \\ 0 \end{bmatrix}.$$

This is a linear perturbation equation, which means that the aircraft flies around the equilibrium points $[\alpha, q, \theta]$. The variables are explained in Table 5.14. The units of the state variables $[\alpha, q, \theta]$ are *rad*, *rad/second*, and *rad* respectively. In this example, the pitch system is naturally unstable because it has a free integrator.

For simulation, the corresponding discrete equations can be obtained by using an Euler method. Therefore,

Table 5.14: Variables of the pitch control system

Variable	Name	Definition
α	the attack angle	the angle between speed and body line (chord line)
θ	the pitch angle	the angle between speed and horizontal line
q	change rate of pitch	the change rate of pitch angle
δ_e	the elevator angle	the deflection of elevator

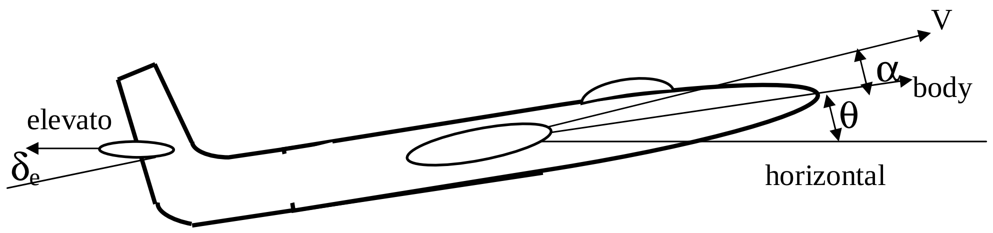


Figure 5.15: Variables of a pitch control system

$$X(k+1) = \dot{X} \cdot TAU + X(k)$$

where, TAU is the sample time step and is set as 0.02 second.

In this experiment, action, i.e., the elevator angle, is assigned as discrete values

$$\delta_e = [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10] \cdot grd,$$

where $grd = \pi/180$.

The limitations of state variables $[\alpha,\ q,\ \theta]$ are $[\pm 15grd,\ \pm 0.5grd/sec,\ \pm 15grd]$, and they are divided into 7 sections, separately. If the absolute value of the pitch angle is less 1 degree and keeps in the error range for 100 seconds, then the experiment is defined as success. If one of the state variables is out of its limitation, or in a given time the pitch angle does not come within the given error range, the experiment is defined as a failure. Both success and failure will end an episode.

In this experiment, the CMAC method, which is explained in Section 2.2.3, is used to map the input space in order to deal with the problems of large state space and generalization. The results using Albus's CMAC architecture are provided here. The number of neighbors is selected as 5. To save memory size, a hash function (or hash-coding) is used to map the conceptual memory C on a physical memory P.

Hash-coding is a technique for reducing the size of memory needed to store data when the data is stored in sparse form, i.e., a small amount of data located over a large memory size. If the data is stored in a sparse memory with size M, and the data is to be mapped on a small memory with size m, the hash function $H(k)$ should satisfy

$$0 \leq H(K) < m, \quad \forall\ 1 \leq K \leq M.$$

There are many methods for forming a hash function. The hash function used in this experiment is the multiplicative mapping [50],

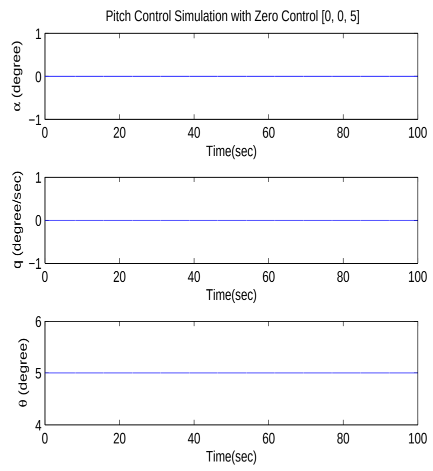
$$H(K) = fix(m((\frac{F}{w}K) mod\ 1))$$

where fix is the function that rounds a value towards to zero. Normally, w is in the form of 2^t, where t is the word size of a computer. F is a free integer, which can be chosen related to w. When F is chosen to be the nearest integer to $\frac{sqrt(5)-1}{2}w$, the hash function is called the "Fibonacci hash function" [50].

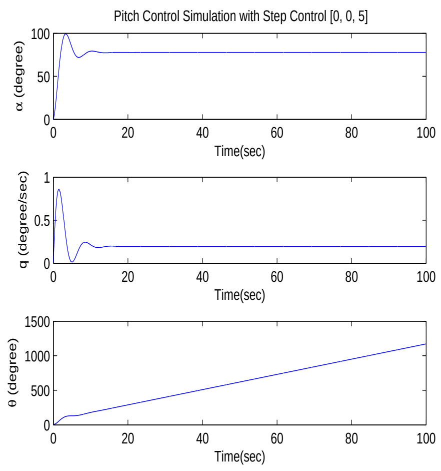
Experimental Results

Figure 5.16 displays the dynamic process of the pitch system under different control strategies. Figure 5.16 (a) indicates that the pitch system is static stable, since it can keep the initial values without any control. However, Figure 5.16 (b) indicates that the system is dynamic unstable because there exists an integrator in the system. The dynamic process with random control is given in Figure 5.16 (c), which is not acceptable. Figure 5.16 (d) presents an example of the dynamic process when a RL-based controller is applied. SARSA(λ) algorithm is used to learn the control policy. Figure 5.16 (d) indicates that the dynamic process is much better than those in Figure 5.16 (a), (b), and (c).

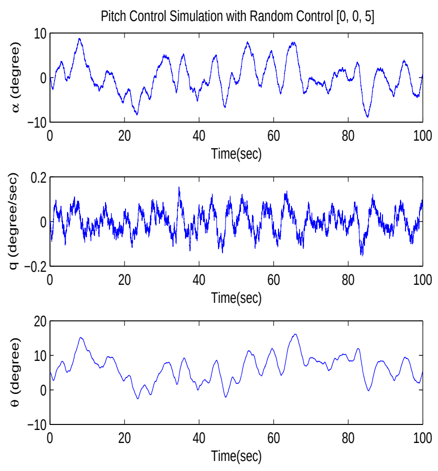
Although an individual RL controller can provide an acceptable dynamic process, the performance of the process is greatly affected by the learning parameters



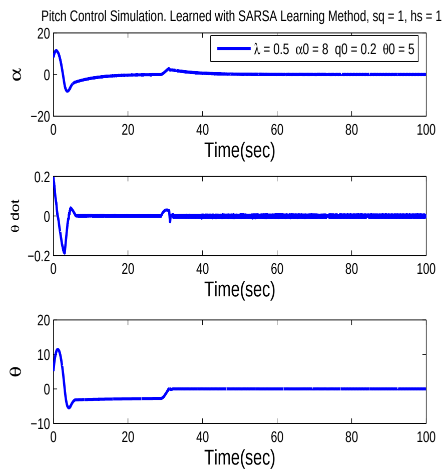
(a)



(b)

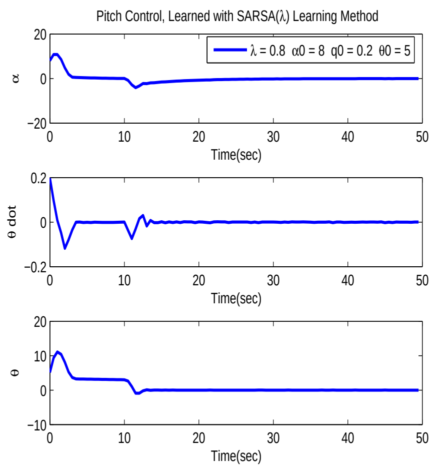


(c)

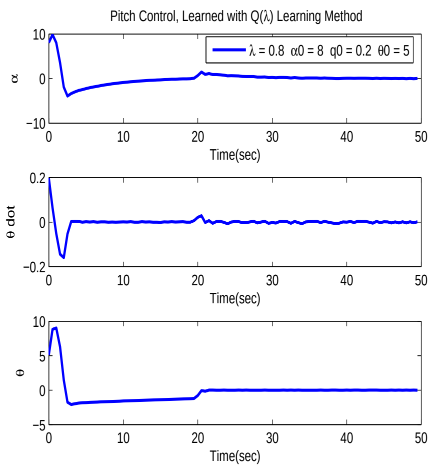


(d)

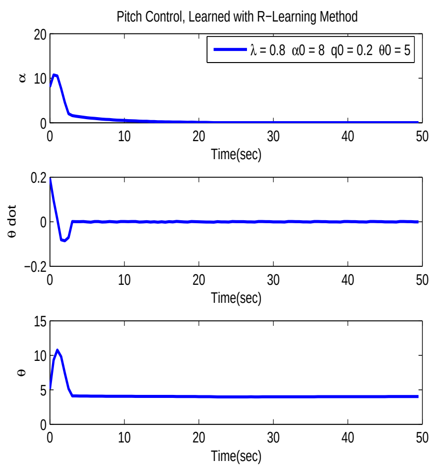
Figure 5.16: Results of the different control strategies (a) no input, no control (b) step input, no control (c) random control (d) SARSA(λ)-learning



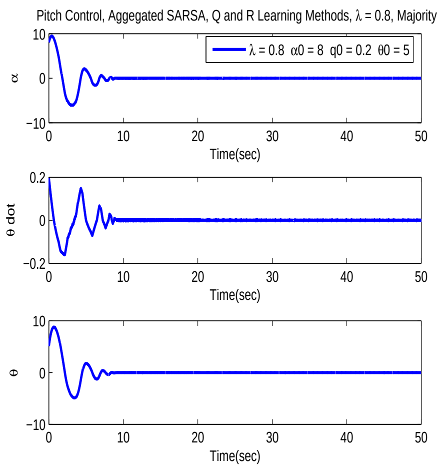
(a)



(b)



(c)



(d)

Figure 5.17: Results of individuals and aggregation with different RL algorithms (a) SARSA(λ)-learning (b) Q(λ)-learning (c) R-learning (d) AMRLS

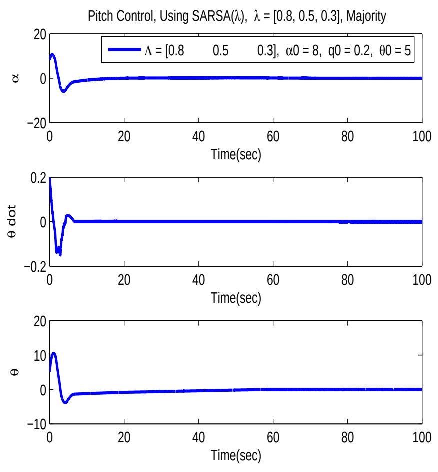
and learning algorithms. As mentioned above, instead of searching and testing the optimal parameters and algorithms, aggregation technique has been incorporated to RL methods. Part of the experimental results are given in Figures 5.17 and 5.18. Figure 5.17 presents the results when AMRLS performs aggregation based on different learning algorithms, SARSA(λ), Q(λ), and R-learning, which is introduced in Section 3.3. Figure 5.18 gives the results when AMRLS uses different learning and aggregation methods and aggregates based on different values of λ.

Figure 5.17 compares the dynamic processes of individual RL methods, SARSA(λ), Q(λ), and R-learning with that of AMRLS. Figure 5.17 (a) and (b) show that the pitch angle can return to zero, the equilibrium point, under the control of the policies that are trained using SARSA(λ) or Q(λ) with CMAC method, but it converges slowly. R-leaning can converge quickly, but unfortunately, it cannot converge to the equilibrium point (Figure 5.17 (c)). AMRLS gives the best result, which is quickly enough and the steady state error approaches zero (Figure 5.17 (d)) [43].

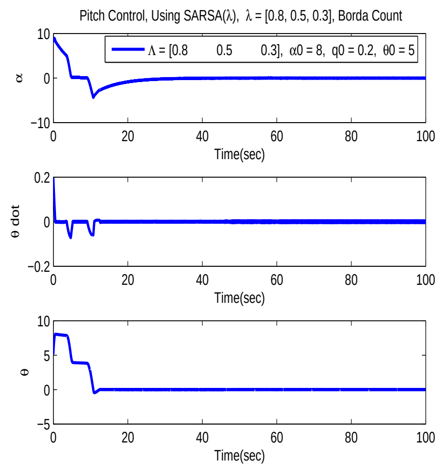
Figure 5.18 presents the control processes of AMRLS using different learning algorithms (SARSA(λ) and Q(λ)) and aggregation methods (WMV and WBC), and aggregation based on different values of λ. Except in the case of Q(λ)-learning with the WBC aggregation method, AMRLS gives satisfactory performances.

5.5.2 Altitude Control System

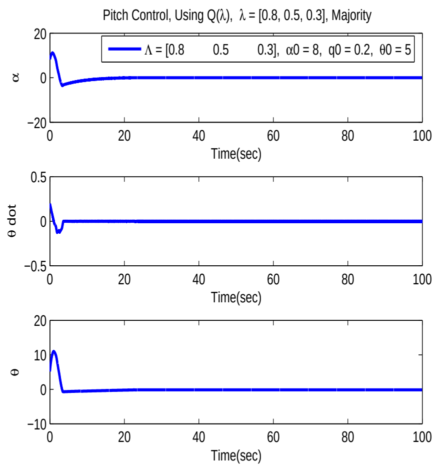
In flying an aircraft, one of the most important tasks of pilots is to hold the altitude of an aircraft on a specific value. Altitude control is very important to safe flight. To keep aircraft from colliding, those that fly on an easterly path are required to be on an odd multiple of 1000 feet and those that are on a westerly path must keep on an even multiple of 1000 feet. The error of altitude of an aircraft should be less than one hundred feet. However, holding altitude is boring, especially in long distance flying. To lessen the work load of pilots and improve the flight performance, aircraft require an altitude control system or an autopilot. Different from the pitch control system, which only assists pilots to control attitudes, an altitude control system can replace pilots for a certain time to control an aircraft. To make passengers comfortable, the overshoot of the dynamic process should be less than 10%.



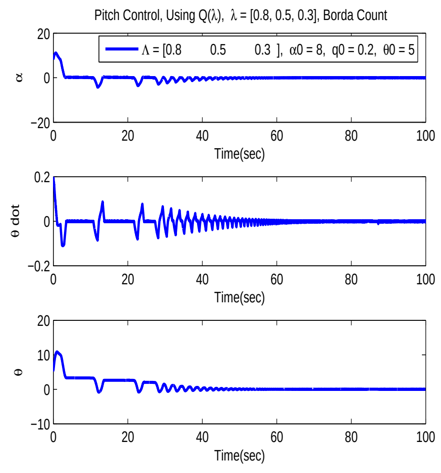
(a)



(b)



(c)



(d)

Figure 5.18: Results of aggregation with different learning parameters λ (a) SARSA(λ) and WMV (b) SARSA(λ) and WBC (c) Q(λ) and WMV (d) Q(λ) and WBC

Similar to pitch control, for most aircraft, altitude control is mainly by elevator δ_e. A linear longitudinal perturbation equation is give in [26]. It is for a model of a Boeing 747 aircraft in horizontal flight at a nominal speed $U_0 = 830 ft/sec$ at 20,000 ft (Mach = 0.8), with a weight of 637,000 lb.

However, altitude control is much more difficult than pitch control. Pitch control is regarded as a short-term moving mode, which responds quickly to a control command. The dynamic process is within 10 seconds. Normally, pitch control is often used as an inner loop for an altitude control system, which is a long-term moving mode. The dynamic process of an altitude system is about 20 seconds or more. Therefore, an altitude control problem is a long-delayed reward problem or long-range dependencies system [81], which is difficult to control. Although RL has been successfully applied to many real applications, and the aircraft's altitude can be controlled by several methods, there is no report about the successful control of an aircraft's altitude with on-line RL technique.

To determine altitude changes, the following equation should be added to the longitudinal equations.

$$\dot{h} = V_{ref} sin\theta - wcos\theta \tag{5.5}$$

The correspond linear equation is

$$\dot{h} = V_{ref}\theta - w. \tag{5.6}$$

Therefore, a linear small perturbation dynamic model for the aircraft in the above given state is presented as follow.

$$\begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} -0.00643 & 0.0263 & 0 & -32.2 & 0 \\ -0.0941 & -0.624 & 820 & 0 & 0 \\ -0.000222 & -0.00153 & -0.668 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 830 & 0 \end{bmatrix} \cdot \begin{bmatrix} u \\ w \\ q \\ \theta \\ h \end{bmatrix} + \begin{bmatrix} 0 \\ -32.7 \\ -2.08 \\ 0 \\ 0 \end{bmatrix} \cdot \delta_e \tag{5.7}$$

The definitions of the variables in Equation (5.7) are provided in Table 5.15. The unit of u, w is *feet/second*; q has the unit *rad/second*; θ's unit is *rad*, and the

Table 5.15: Variables of the altitude control system

variable	name	definition
u	forward velocity	velocity perturbation in body-axis x direction
w	vertical velocity	velocity perturbation in body-axis z direction
q	change rate of pitch	change rate of pitch, about positive body-axis y direction
θ	pitch angle	pitch angle perturbation from a reference value
h	altitude	altitude perturbation from a reference value
δ_e	elevator angle	the deflection of elevator from a reference value

unit of h is *feet*. δ_e, deflection of elevator, is the same as the one in pitch control and its unit is *rad*. These units are used in calculation. For display, *rad* is converted to *degree*.

Let $X = [u, w, q, \theta, h]^T$, equation (5.7) is simplified as

$$\dot{X} = A \cdot X + B \cdot \delta_e;$$

Normally, there are two purposes for an altitude control system. One is to hold the altitude constant, i.e., when there is any perturbation of the altitude, the control system can diminish the perturbation and make the steady state error as small as possible. The other is to follow an altitude change command as quickly and accurately as possible. In this experiment, focus is on the first purpose. In order to make passengers comfortable, the damping ratio for altitude should be around 0.5.

Experimental Setup

In the experiments, the action, elevator angle, is assigned as discrete values

$$\delta_e \quad = \quad [-5, -4, -3, -2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, 3, 4, 5] \cdot grd \qquad (5.8)$$

where $grd = \pi/180$. The limitations of state variables $[u,\ w,\ q,\ \theta,\ h]$ are [100, 100, 10, 20, 300].

Table 5.16: Division of variable sections

Variable	Division	Number of section
u	[-100,-50), [-50,-10), [-10,0), [0,10), [10,50), [50,100]	6
w	[-100,-50), [-50,-10), [-10,0), [0,10), [10,50), [50,100]	6
q	[-10,-5), [-5,-2), [-2,0), [0,2), [2,5), [5,10]	6
θ	[-20,-10), [-10,-5), [-5,0), [0,5), [5,10), [10,20]	6
h	[-300,-200), [-200,-100), [-100,-50), [-50,-20), [-20,-10), [-10,0) [0,10), [10,20), [20,50), [50,100), [100,200), [200,300]	12

The definition of success in this experiment is that the absolute value of altitude is fewer than 10 feet and is kept in the range for a given number of time steps. If one of the state variables is out of its limitation, or in a given time the altitude cannot be within the given error range, the experiment is defined as a failure.

In the first experiment, the state space is transformed into discrete domain. The variables u, w, q, θ, h are divided into 6, 6, 6, 6, and 12 sections respectively within their limiting ranges (shown in Table 5.16).

The dimension of the Q-value table is $15552 \times 15 = 233280$, which is very large for tabular-based RL algorithms. Even when the CMAC technique is used to generalize, the results are not good. In designing the RL scheme for the altitude control system, the following factors are taken into account.

- State Space Partition

  For tabular-based RL algorithms, a state space partition is an important step. There are five variables in the altitude system. Even if a continuous feedback technique is used to control the system, it is not easy to design a controller that performs well in controlling the altitude system. When discrete control values δ_e were used for RL, it was very difficult to control the system. Based on trial-and-error, the above state space partition seems more effective.

- Reward Function

As mentioned in Section 2.1.5, the design of a reward function has great influence on learning performance. In the altitude control problem, people not only want to keep the altitude in a given error range but also want to drive the altitude to the error range as quickly as possible with an acceptable performance. One of the rewards is defined as follows:

$$r = \begin{cases} 1, & \text{if } |h_{new}| < |h_{old}|; \\ -1, & \text{else.} \end{cases} \tag{5.9}$$

where, $|h_{new}|$ means the absolute value of the disturbance of altitude in the new time step, and $|h_{old}|$ is the value in the previous time step.

It seems that the definition of the reward function is reasonable because the altitude, h, should approach zero. In experiments, an interesting result, shown in Figure 5.19, is observed.

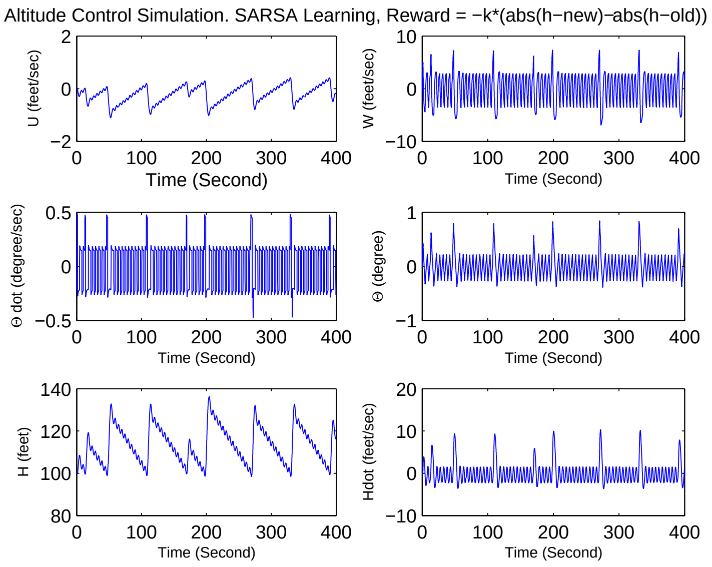


Figure 5.19: Unexpected responds with an incorrect reward definition

This is not the dynamic performance expected, but it shows the way that a learning algorithm only wants to enlarge its accumulated rewards rather than meet the goal. In most of the time steps, the learner decreases the h

value slowly and obtains the reward in each time step. Then, in one time step, the learner increases the h value rapidly and is punished by a negative reward. However, as the learner obtains more rewards than punishment in a certain time period, it wants to repeat the procedure again and obtain more rewards. This produces the dynamic process shown in Figure 5.19, which is poor. Therefore, reward functions should be designed carefully. In the experiments, the reward function is defined as

```
Definition of reward function
if failure
    reward = PUNISHMENT;
else
    reward = REWARD;
        if (h × $\dot{h}$ < 0)
            r = r + REWARD1;
        else
            r = r - REWARD1;
        end
        if abs(h) < $h_{ideal}$
            r = r + REWARD2;
        end
end
```

- CMAC Method

  Because of the larger state space, many state-action pairs were not visited, even though the exploration rate is increased. Establishing the Q-value table is very difficult. In order to generalize, a CMAC method was adopted. However, the weight vector of the CMAC controller is very sparse because of the sparse table of Q-values. Consequently, the control effect is not good.

  The Q-value table is sparse because the altitude system is static stable, which tends to keep variables constant. Therefore, it is difficult to encourage exploration only by increasing the ϵ value of an $\epsilon-$greedy approach, or increasing the value of T in Equation (2.5). More efficient exploration strategies should be used to obtain experience in more state-action pairs. This stimulates the adoption of GARL methods.

- Initial Q-values

  Initial Q-values can be regarded as an off-line teacher who provides the initial experience for learning algorithms. Initial values can be obtained by

  - Knowledge
    If $h \times \dot{h} < 0$, $|h|$ will decrease. Therefore, these states are desired.
  - Experience
    A positive value of δ_e will decrease the attack angle α, and lose the lift force. Therefore, the altitude will decrease. The experience can guide the setting of initial Q-values.
  - Learning
    When a system is complicated, such as in the altitude system, the above two ways to set up the initial Q-values are inefficient. Learning the values is a more feasible way.

  Using initial values is performed by the experience cloning module explained in Section 4.3. In the experiments, all three ways were used to establish the initial Q-values, and the last one gives the best results.

Experimental Results

Figure 5.20 provides the dynamic control process of the altitude system by aggregating three HGATDRL algorithms using different λ values: $\lambda_1 = 0.8$, $\lambda_2 = 0.5$, $\lambda_3 = 0.3$. The performances are improved because the steady-state error becomes smaller. Compared with the dynamic performances shown in Figure 5.21 (provided by Franklin et al. [26]) that is controlled by the controller designed based the the classical feedback control theory, the dynamic performances of the altitude system controlled with HGATDRL algorithms are very good. It should be

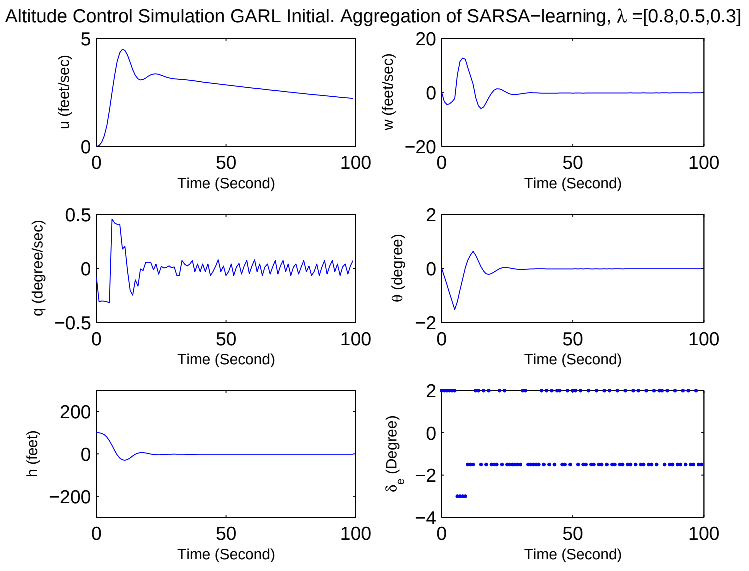


Figure 5.20: Performance of an altitude control system using aggregation of SARSA(λ) based on GARL

emphasized that the HGATDRL-based controller is learned by the learning system itself, but the feedback-based controller should be designed by designers with great efforts [26].

The robustness of the control policy has been tested in a new system with slight changes to matrix A.

$$A = \begin{bmatrix} -0.00643 & 0.0263 & 0 & -32.2 & 0 \\ -0.0941 & -0.624 & 761 & -196.2 & 0 \\ -0.000222 & -0.00153 & -4.41 & -12.48 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 830 & 0 \end{bmatrix},$$

Figure 5.22 provides the dynamic control process of the altitude system. Obviously, the performance is good, too, indicating that the control policy performs robustly.

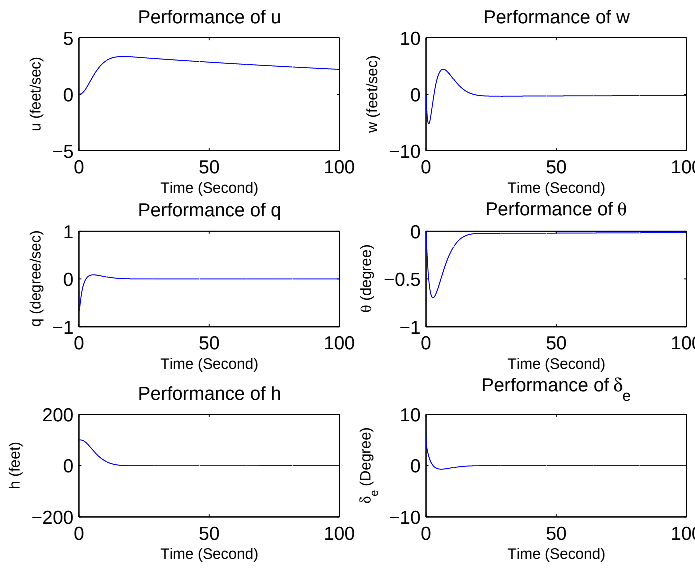


Figure 5.21: Performance of altitude control system using Classical Feedback method

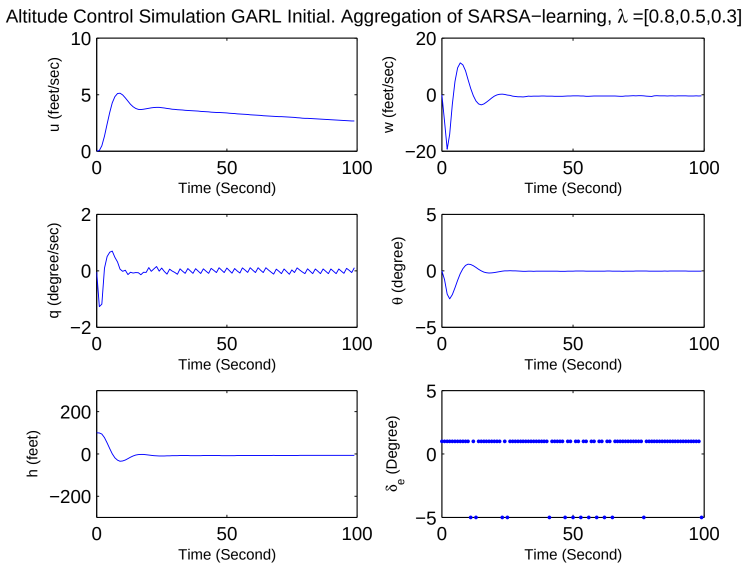


Figure 5.22: Performance of an altitude control system using aggregation of SARSA(λ) based on GARL, with the changed matrix A

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Improving the learning abilities of Reinforcement Learning (RL) algorithms is a challenging research topic in the fields of machine learning and automatic control. One of the main contributions of this dissertation is the proposal of a general and novel multiple learning framework - the Aggregated Multiple Reinforcement Learning System (AMRLS), which on-line aggregates several different learning algorithms or algorithms with the same learning method but different values of a certain learning parameter. AMRLS consists of two main modules: the learning module and the aggregation module. These two modules interact with each other at each time step to form a dynamically aggregated RL procedure. Chapter 3 describes the proposed framework in detail.

RL algorithms have been successfully applied to various real problems. However, there is no universal rule to guide the practice of choosing a suitable learning algorithm and the setting of the values of some learning parameters for a given task. Usually, this practice can be accomplished based on experiments or experience, but it is time-consuming and inefficient. With aggregation, AMRLS successfully avoids the job of presetting parameters and algorithms. Also, AMRLS improves the overall learning performances and increases the adaptability to a dynamic environment by dynamic aggregation.

AMRLS can employ various information for aggregation. For example, it can aggregate the knowledge (Q-value function), statistical information (the probabilities of different actions), preferences (the ranks of different actions), or decisions (the selected actions) provided by the learning module. To improve the performance of AMRLS, several intelligent on-line aggregation strategies have been developed. For instance, this dissertation proposes a weighting technique to enhance the effectiveness of aggregation. The weights can be updated automatically during the learning process, and each learner is emphasized by the weights in aggregation.

Many original experiments are designed and presented in Chapter 5. The experimental results verify that AMRLS reduces the number of training episodes and steps, increases the success rate, learns more smoothly, exhibits an improved dynamic and static learning performance, improves the confidence of learning, and ensures the ability of robustness and fault tolerance. It has been confirmed that AMRLS is a valuable research platform for developing more efficient RL algorithms [39, 40, 41].

Value Function Learning (VFL) and Direct Policy Search (DPS) are two different learning strategies for RL. Many papers have described these two RL strategies, but they mainly emphasize the advantages of only one strategy. In fact, both strategies exhibit their strengths and weaknesses. In many complicated applications with large state space, partially observed states, or sparse information, DPS-based RL algorithms, such as GARL, perform much better than conventional VFL-based RL algorithms, for example, TDRL. However, GARL is an off-line learning scheme; therefore, it is not applicable in dynamic environments.

This dissertation improves the GARL used by Ikeda [37] through modifying the crossover operation, adding a mutation operation, and introducing a bonus function to the fitness function. The improved GARL algorithm exhibits very strong learning ability. It can train a high quality controller (policy) to control some systems that are difficult to handle with conventional RL algorithms. GARL also exhibits robustness in the choice of initial states. This improved GARL is tested in the cart-pole system and flight control system. The experiments produce wonderful results.

This dissertation suggests a hybrid RL architecture of GARL and TDRL, HGAT-

DRL, which integrates an off-line GARL algorithm with certain on-line TDRL algorithms to combine their strengths. The HGATDRL contains three modules: off-line GARL module, experience cloning module, and on-line TDRL module. The three modules work in serial. Due to the efficient parallel search ability, GARL can provide an acceptable initial policy for HGATDRL. Then, the experience cloning module converts the policy to the initial values for TDRL algorithms. Finally, TDRL algorithms on-line update the policy in a dynamic environment.

HGATDRL is tested in the altitude control system of an aircraft to prove its feasibility. The experimental results show that this method can control the altitude system that is very difficult to control with conventional RL algorithms. At the same time, HGATDRL also presents a strong robust property to the initial policy and an adaptability to the changes of environment.

6.2 Future Work

Although AMRLS has been successfully applied to a number of tasks, its performance can be improved further in several directions.

- Develop More Efficient Aggregation Techniques

  Only a few aggregation techniques, weighted average, weighted majority voting, weighted plurality, weighted Borda count, and weighted instant runoff voting, are employed in this dissertation. The effects of aggregation are good, but can be improved by adopting new aggregation methods. More aggregation techniques, such as fuzzy integral algorithms or the Bayes approach, could be studied and modified for AMRLS.

- Learn Adaptive Strategies to Adaptively Aggregate

  Different learning algorithms present different performances for different applications or in different learning stages. In this dissertation, weights are introduced to evaluate the significance of different algorithms. More efficient adaptive aggregation strategies should be developed. Although certain papers [46, 75, 90] provide some useful approaches to adaptive aggregation for classification, they cannot be applied directly to RL because it is a dynamic sequential decisions process. Developing adaptive aggregation strategies by using RL is a viable research topic for future.

- Enhance the Ability of AMRLS by Cooperation

  In this dissertation, AMRLS is only a basic type of Multiagent Systems (MASs), and the interactions of different learning algorithms are coordinated by aggregation. It should be easy to extend AMRLS to large MASs. In such a case, each agent in a MAS will be an AMRLS, that is, an enhanced learner, and the interactions among different agents can be coordinated by cooperation strategies. Thus, the new system will be more powerful than the original MAS.

- Enhance the Learning Ability of HGATDRL by Using CBP Method

  The experimental results demonstrate that CBP-based GARL performs much better than EBP-based GARL in many cases. To combine CBP-based GARL with TDRL requires a new technique for experience cloning. Some experiments have been conducted in the research, but more work needs to be done to enhance the learning ability of HGATDRL.

- Apply AMRLS to Real-time Applications

  All of the experiments in this dissertation have been conducted in simulated environments. AMRLS should be tested in real-time, noisy, realistic tasks.

- Provide Theoretical Proofs

  The conclusions obtained in the dissertation are principally based on experiments. To make more general conclusions, theoretical proofs are required.

Bibliography

[1] P. A. Absil and R. Sepulchre, “A Hybrid Control Scheme for Swing-up Acrobatics”, *Proceedings of the 5th European Control Conference*, ECC 2001, Porto, Portugal, September 4-7, pp. 2860-2864, 2001.

[2] J. S. Albus, “A New Approach to Manipulator Control: The Cerebellar Model Articulation Control (CMAC), *Trans. of the ASME Journal of Dynamic Systems, Measurements, and Control*, pp. 220-227, 1975.

[3] John D. Anderson, *Introduction to Flight (Fourth Edition),* McGraw-Hill Companies, USA, 2000.

[4] Perez-Uribe Andres, *Structure-Adaptable Digital Neural Networks*, Ph.D. Thesis, 2000, http://www.geocities.com/fastiland/thesis/.

[5] Sachiyo Arai, Katia Sycara, and Terry R. Payne, “Multi-agent Reinforcement Learning for Planning and Scheduling Multiple Goals”, in *Proceedings of the Fourth International Conference on Autonomous Agents*, pp. 104-105, 2000.

[6] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally Weighted Learning”, *Artificial Intelligence Review*, 11 (1-5), pp. 11-73, 1997.

[7] Martin Appl and Wilfried Brauer, “Fuzzy Model-based Reinforcement Learning”, 2000, http://www.erudit.de/erudit/events/esit2000/proceedings/AE-02-3-P.pdf.

[8] J. D. Bagley, *The Behaviour of Adaptive Systems Which Employ Genetic and Correlation Algorithms*, Ph.D. Thesis, University of Michigan, Ann Harbor, United States, 1967.

[9] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems", *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-13, no.5, pp. 834-846, 1983.

[10] Andrew G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to Act Using Real-time Dynamic Programming", *Artificial Intelligence*, vol. 72 (1-2), pp. 81-138, 1995.

[11] R. D. Beer and J. C. Gallagher, "Evolving Dynamical Neural Networks for Adaptive Behavior", *Adaptive Behavior*, vol. 1 (1), pp. 91-122, 1992.

[12] R. E. Bellman, "A Markov Decision Process", *Journal of Mathematical Mechanics*, 6, pp. 679-684, 1957.

[13] A. E. Bryson Jr., *Control of Spacecraft and Aircraft*, Princeton, NJ: Princeton University Press, 1994.

[14] I. Bucak and M. Zohdy, "Reinforcement Learning of a Biped Robot System", *Internal Journal of Intelligent Control and Systems*, vol. 3, no. 4, pp. 601-617, 1999.

[15] I. Bucak and M. Zohdy, "Reinforcement Learning of Nonlinear Multi-link System", *Enginering Applications of Artificial Intelligence*, vol. 14, pp. 563-573, 2001.

[16] Georgios Chalkiadakis and Craig Boutilier, "Coordination in Multiagent Reinforcement Learning: A Bayesian Approach", *2nd Intl. Conf. on Autonomous Agents and Multiagent Systems* (AAMAS-03), pp. 709-716, 2003.

[17] Ching-Tsan Chiang and Chun-Shin Lin, "CMAC with General Basis Functions", *Neural Networks*. 9(7), pp. 1199-1211, 1996.

[18] R. H. Crites and A. G. Barto, "Improving Elevator Performance Using Reinforcement Learning", *Advances in Neural Information Processing Systems*, 8, pp. 1017-1023, 1996.

[19] Belur V. Dasarathy, *Decision Fusion.* IEEE Computer Society Press, 1994.

[20] L. Davis, *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.

[21] L. Davis, *Handbook of Genetic Algorithms.* van Nostrand Reinhold editor, New York, USA, 1991.

[22] T. G. Dietterich and X. Wang, "Batch Value Function Approximation via Support Vectors", *http://citeseer.ist.psu.edu/497023.html*, 2002.

[23] Marco Dorigo and Marco Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*, The MIT Press, 1998.

[24] Kenji Doya, "Reinforcement Learning in Continuous Time and Space", *Neural Computation*, vol. 12, no. 1, pp. 219-245, 2000.

[25] D. Floreano and F. Mondada, "Evolution of Homing Navigation in a Real Mobile Robot". M. Dorigo Editor, Special Issue on learning Autonomous Robots. *IEEE Transactions on Systems, Man, and Cybernetics,* Part B 26 (3), pp. 396-407, 1996.

[26] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini, *Feedback Control of Dynamic Systems*, Pearson Prentice Hall, 2006.

[27] P. Gader, M Mohamed, and J. Keller, "Fusion of Handwritten Word Classifiers", *Pattern Recognition Letters*, vol. 17, pp. 577-584, 1996.

[28] Abhijit Gosavi, *Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement Learning.* Kluwer Academic Publishers, 2003.

[29] John J. Grefenstette, Connie L. Ramsey, and Alan C. Schultz, "Learning Sequential Decision Rules Using Simulation Models and Competition", *Machine Learning*, 5, pp. 355-381, 1990.

[30] Lawrence O. Hall and Michael A. Pokorng, "Averaged Reward Fuzzy Reinforcement Learning Applied to Fuzzy Rule Tuning", *http://citeseer.ist.psu.edu/33773.html*, 1997.

[31] F. Ho and M. Kamel, “Learning Coordination Strategies for Cooperative Multiagent Systems”, *Machine Learning*, vol.33, pp. 155-177, 1998.

[32] T. K. Ho, J. J. Hull, and S. N. Srihari, “Decision Combination in Multiple Classifier Systems”, *IEEE Transaction on Pattern Analysis and Machine Intelligence* 16(1), pp. 66-75, 1994.

[33] J. H. Holand, *Adaptation in Natural and Artifcial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, AnnArbor, MI, 1975.

[34] I. Horowitz, “Survey of Quantitative Feedback Theory (QFT)”, *International Journal of Control*. vol. 53, no. 2, pp. 255-291. 1991.

[35] Xiaobing Hu, Shufan Wu, and Ju Jiang, “On-line Free-flight Path Optimization Based on Improved Genetic Algorithms”, *Engineering Applications of Artificial Intelligence*, vol. 17, pp. 897-907, 2004.

[36] Y. S. Huang and C. Y. Suen, “A Method of Combining Multiple Experts for the Recognition of Unconstrained Handwritten Numerals”, *IEEE Trans. On Pattern Analysis and Machine Intelligence* 17(1), pp. 90-94, 1995.

[37] Kokolo Ikeda, “Genetic Policy Search Using Exemplar Based Representations”, *The 8th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, 6th - 7th December 2004, Cairns, Australia, pp. 83-92, 2004.

[38] C. Z. Janikow and Z. Michalewicz, “An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms”, in *Proc. of the 4th Intl. Conference on Genetic Algorithms*, pp. 31-36, 1991.

[39] Ju Jiang, Mohamed Kamel, and Lei Chen, “Reinforcement Learning and Aggregation”, in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics 04*, pp. 1303-1308, 2004.

[40] Ju Jiang, Mohamed Kamel, and Lei Chen, “Aggregation of Multiple Reinforcement Learning Algorithms”, *International Journal on Artificial Intelligence Tools,* vol. 15, no. 5, pp. 855-861, October 2006.

[41] Ju Jiang and Mohamed S. Kamel, "Aggregation of Reinforcement Learning Algorithms", in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN2006)*, pp. 68-72, 2006.

[42] Ju Jiang and Mohamed S. Kamel, "An Algorithm for Improving the Robustness of Direct Policy Search-based Reinforcement Learning", *Dynamics of Continuous, Discrete and Impulsive Systems, An International Journal for Theory and Applications (Series B), Special Volume: Advances in Neural NetworksC-Theory and Applications*, May, 2007 (in press).

[43] Ju Jiang and Mohamed S. Kamel, "Pitch Control of an Aircraft with Aggregated Reinforcement Learning Algorithms", *International Joint Conference on Neural Networks, (IJCNN2007)*, 2007 (in press).

[44] M. Jordon and R. Jacobs, "Hierarchical Mixtures of Expert and the EM Algorithm", *Neural Computing*, pp. 181-214, 1994.

[45] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research* no. 4, pp. 237-255, 1996.

[46] M. Kamel and N. Wanas, "Data Dependence in Combining Classifiers", *Multiple Classifiers Systems, Fourth International Workshop, Surrey, UK*, June 11-13, pp. 1-14, 2003.

[47] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.

[48] Ludmila I. Kuncheva, James C. Bezdek, and Robert P. W. Duin, "Decision Templates for Multiple Classifier Fusion: An Experimental Comparison", *Pattern Recognition*, 34, (2), pp. 299-314, 2001.

[49] Ludmila I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*, A Wiley Interscience Publication, 2004.

[50] D. Knuth, "Sorting and Searching", *The Art of Computer Programming*, vol. 3, pp. 506-512, 1973.

[51] C. S. Lin and H. Kim, "CMAC-based Adaptive Critic Self-learning Control", In *Proccedings on the IEEE Int. Symposium on Intelligent Control,* pp. 200-205, 1989.

[52] Sridhar Mahadevan, "Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results", *Machine Learning,* 22(1/2/3), pp. 159-195, 1996.

[53] "Modeling a Pitch Controller", *www.engin.umich.edu/group/ctm/examples/pitch/Mpitch.html.*

[54] Chris Messom, "Genetic Algorithms for Auto-tuning Mobile Robot", *Res. Lett. Inf. Math. Sci.*, vol. 3, pp. 129-134, 2002.

[55] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, Massachusetts, USA, The MIT Press, 1996.

[56] S. O. R. Moheimani and I. R. Petersen, "Optimal Quadratic Guaranteed Cost Control of a Class of Uncertain Time-delay Systems", *Control Theory and Applications, IEEE Proceedings*, vol. 144, Issue 2, pp. 183-188, 1997.

[57] M. L. Moore, J. Musachio, and K. M. Passino, "Genetic Adaptive Control for an Inverted Wedge", *Engineering Applications of Artificial Intelligence*, vol. 14, no. 1, pp. 1-14, 2001.

[58] Eduardo F. Morales and Claude Sammut, "Learning to Fly by Combining Reinforcement Learning with Behavioural Cloning", *url = citeseer.ist.psu.edu/morales04learning.html*, 2004.

[59] David E. Moriarty and R. Mikkulainen, "Efficient Reinforcement Learning through Symbiotic Evolution", *Machine Learning*, 22(1-3), pp. 11-32, 1996.

[60] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette, "Evolutionary Algorithms for Reinforcement Learning", *Journal of Artificial Intelligence Research* 11, pp. 241-276, 1999.

[61] H. Muhlenbein and D. Schlierkamp-Voosen, "Predictive Models for the Breeder Genetic Algorithm, I. Continuous Parameter Optimization", *Evolutionary Computation*, 1(1), pp. 25-49, 1993.

[62] Y. Nagata and S. Kobayashi, "Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem", in *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 450-457, 1997.

[63] Andrew Y. Ng, *Shaping and Policy Search in Reinforcement Learning*, Ph.D. Dissertation. University of California, Berkeley, 2003.

[64] P. Nordin and W. Banzhaf, "A Genetic Programming System Learning Obstacle Avoiding Behavior and Control a Miniature Robot in Real Time", *Technical report* 4/95, Department of Computer Science, University of Dortmund. 1996.

[65] Akira Oyamal, Shigeru Obayashi, and Takashi Nakamura, "Real-coded Adaptive Range Genetic Algorithm Applied to Transonic Wing Optimization", *www.ifs.tohoku.ac.jp/edge/publications/Ppsnvi.pdf*.

[66] J. M. Porta and E. Celaya, "Reinforcement Learning for Agents with Many Sensors and Actuators Acting in Categorizable Environments", *Journal of Artificial Intelligence Research*, vol. 23, pp. 79-122, 2005.

[67] L. Pitt and C. Smith. "Probability and Plurality for Aggregations of Learning Machines". *Information and Computation* , 77, pp. 77-92, 1988.

[68] S. Ramamoorthy and B. Kuipers, "Qualitative Heterogeneous Control of Higher Order Systems". In O. Maler and A. Pnueli (Eds.), *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, Springer Verlag, pp. 417-434, 2003.

[69] M. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, and R. Ehrmann, "Karlsruhe Brainstormers - A Reinforcement Learning Approach to Robotic Soccer", *Lecture Notes in Computer Science*, vol. 2019, pp. 367-372, 2001.

[70] William H. Riker, *Liberalism Against Populism.* Prospect Heights, Waveland Press, Inc, 1982.

[71] G. A. Rummery and M. Niranjan, "On-line Q-learning Using Connectionist Systems", *Technical Report CUED/F-INFENG/TR 166*, Engineering Department, Cambridge University, 1994

[72] Jurgen Schmidhuber, "Sequential Decision Making Based on Direct Search", In *Sequence Learning: Paradigms, Algorithms, and Applications.* Ron Sun, C. Lee Giles (Eds), pp. 213-240, 2001.

[73] R. Schoknecht, "Optimality of Reinforcement Learning Algorithms with Linear Function Approximation", *Advances in Neural Information Processing Systems,* vol. 15, pp. 1555-1562, 2003.

[74] Anton Schwartz, "A Reinforcement Learning Method for Maximizing Undiscounted Rewards", In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts, pp. 298-305, 1993.

[75] K. Shaban, O. A. Basir, M. Kamel, and K. Hassanein, "Intelligent Information Fusion Approach in Cooperative Multiagent Systems", *World Automation Congress, 2002. in Proceedings of the 5th Biannual*, vol. 13 , 9-13 June 2002, pp. 429-434, 2002.

[76] Jennie Si, Andy Barto, Warren Powell, and Donald Wunsch, *Handbook of Learning and Approximate Dynamic Programming*, John Wiley and Sons, INC. Publication, IEEE Press, 2004.

[77] Satinder Singh and Dimitri Bertsekas, "Reinforcement Learning for Dynamic Channel Allocation in Cellular Telephone Systems", in *Advances in Neural Information Processing Systems*, M. C. Mozer, M. I. Jordan, and T. Petsche, editors, NIPS-9, The MIT Press, pp. 974-980, 1997.

[78] Peter Stone and M. Veloso, "Multi-agent Systems: A Survey from a Machine Learning Perspective", *Autonomous Robotics*, 8(3), pp. 345-383, July, 2000.

[79] Malcolm J. A. Strens and Andrew W. Moore, "Direct Policy Search Using Paired Statistical Test", in *Proceedings of the Eighteenth International Conference on Machine Learning* (ICML-2001), pp. 545-552, 2001.

[80] Ron Sun and Todd Peterson, "Multi-agent Reinforcement Learning: Weighting and Partitioning", *Neural Networks*, pp. 727-753, 1999.

[81] Ron Sun and C. Lee Giles (Eds), *Sequence Learning: Paradigms, Algorithms, and Applications.* Springer, 2001.

[82] Richard S. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding", in *Advances in Neural Information Processing Systems*, 8, pp. 1038-1044, 1996.

[83] Richard S. Sutton, "Learning to Predict by the Methods of Temporal Differences", *Machine Learning*, no.3, pp. 9-44, 1988.

[84] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning, An Introduction*, The MIT Press, Cambridge, Massachusetts, London, England, ISBN 0-262-19398-1, 1998.

[85] M. Tan, "Multi-agent Reinforcement Learning: Independent vs. Cooperative Agent", *Readings in Agents (eds. M.N. Huhns and M.P. Singh)*, Morgan Kaufman, pp. 487-494, 1993.

[86] J. Tani, "Model-based Learning for Mobile Robot Navigation from the Dynamical Systems Prespective", In M. Dorigo, editor, Special issue on learning autonomous robots. *IEEE Transactions on Systems, Man, and Cybernetic,* Part B 26(3), pp. 421-436, 1996.

[87] Matthew E. Taylor, Shimon Whiteson, and Peter Stone, "Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*, pp. 1321-1328, 2006.

[88] Karl Tuyls, Katja Verbeeck, and Tom Lenaerts, "A Selection-mutation Model for Q-learning in Multi-agent Systems", in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 693-700, 2003.

[89] Antanas Verikas, Arunas Lipnickas, Kerstin Malmqvist, Marija Bacauskiene, and Adas Gelzinis, "Soft Combination of Neural Classifiers: A Comparative Study", *Pattern Recognition Letters*, no.20, pp. 429-444, 1999.

[90] N. Wanas, M. Kamel, G. Auda, and F. Karray, "Feature-based Decision Aggregation in Modular Neural Network Classifiers ", *Pattern Recognition Letters,* vol.20, no.11-13, Nov. 1999, pp. 1353-1359, 1999.

[91] Hui Wang, Jin-fa, Xu, and Zheng Gao, "Adaptive Neural Network Attitude Control for Unmanned Helicopter", *Transactions of Nanjing University of Aeronautics and Astronautics*, vol. 21, no. 3, pp. 168-173, 2004.

[92] C. J. C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, 8 (3): pp. 279-292, 1992.

[93] Gerhard Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, 1999.

[94] S. D. Whitehead, "A Complexity Analysis of Cooperative Mechanisms in Reinforcement Learning", in *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pp. 607-613, 1991.

[95] Shimon Whiteson and Peter Stone, "Evolutionary Function Approximation for Reinforcement Learning", *Journal of Machine Learning Research*, pp. 877-917, 2006.

[96] D. Whitley, S. Dominic, and R. Das, "Genetic Reinforcement Learning with Multilayer Neural Networks", in *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 562-569, 1991.

[97] D. Whitley, S. Dominic, R. Das, and C. W. Anderson, "Genetic Reinforcement Learning for Neurocontrol Problem", *Machine Learning*, vol. 13, pp. 259-284, 1993.

[98] Z. Y. Wu, "A New Adapted GA and Its Application in Multimodal Function Optimization", *Control Theory and Applications*, vol. 16, no. 1, pp. 127-129, 1999.

[99] Lei Xu, Adam Krzyzak, and Ching Y. Suen, "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 3, pp. 418-435, 1992.