LEARNING STRATEGIES IN MULTI-AGENT SYSTEMS - APPLICATIONS TO THE HERDING PROBLEM

by

Aditya Gadre

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements of the degree of

Master of Science
In
Electrical Engineering

APPROVED

Dr. Pushkin Kachroo, Chairman

Dr. Hugh VanLandingham

Dr. Will Saunders

November 30, 2001
Blacksburg, VA

Keywords: Dynamic Programming, Reinforcement Learning, Q-learning, reward functions, Artificial Immune System, Idiotopic Network

LEARNING STRATEGIES IN MULTI-AGENT SYSTEMS – APPLICATIONS TO THE HERDING PROBLEM

Aditya Gadre

Abstract

"Multi-Agent systems" is a topic for a lot of research, especially research involving strategy, evolution and cooperation among various agents. Various learning algorithm schemes have been proposed such as reinforcement learning and evolutionary computing.

In this thesis two solutions to a multi-agent herding problem are presented. One solution is based on Q-learning algorithm, while the other is based on modeling of artificial immune system.

Q-learning solution for the herding problem is developed, using region-based local learning for each individual agent. Individual and batch processing reinforcement algorithms are implemented for non-cooperative agents. Agents in this formulation do not share any information or knowledge. Issues such as computational requirements, and convergence are discussed.

An idiotopic artificial immune network is proposed that includes individual B-cell model for agents and T-cell model for controlling the interaction among these agents. Two network models are proposed – one for evolving group behavior/strategy arbitration and the other for individual action selection.

A comparative study of the Q-learning solution and the immune network solution is done on important aspects such as computation requirements, predictability, and convergence.

ACKNOWLEDGMENTS

This is a major milestone in my life, for which I would like to thank my parents. Their love and constant support over the years have been the most important factors in my achievements and my life.

My deepest appreciation goes to my advisor, Dr. Pushkin Kachroo, for his valuable assistance, guidance, and encouragement in bringing this research work to a successful completion. He showed extraordinary patience and was always available for friendly advice. I am also grateful to my thesis committee members, Dr. Hugh VanLandingham and Dr. Will Saunders, for their advise and help.

I would like to thank all my colleagues and friends that I have acquired throughout my years at Virginia Tech. It is their company and all great moments that I shared with them - that enabled me to successfully complete my research work.

There are many others who were indirectly supportive to my work and I would like to thank all of them from my heart.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Motivation

Learning in a dynamic, uncertain multi-agent system is a challenging task. A lot of research is done in this area using various models of procedural learning. Many models assume agents in isolation; others include interactions between agents to varying extents. All agents learn strategies from their own experiences and then interact to evolve strategies that collectively achieve certain goals.

A large class of such systems is studied under Pursuit-Evasion games, with a great emphasis on evolutionary game theory and reinforcement learning. These Pursuit-Evasion games include a number of agents with conflicting goals. Various algorithms are studied to solve this class of problems. These include Dynamic Programming, Neural Networks, and Stochastic Iterative Algorithms.

In this thesis we study the Pursuit-Evasion game in the form of a herding problem in which a number of agents try to herd other agents to a certain location. The main focus of this research is on the study of learning algorithms that are on-line. The herding problem is basically an action selection problem in which a number of agents select various actions so as to find the optimal solution.

Action selection problem in multi-agent games is a topic of active research. Application areas of this research include autonomous robots, Internet software agent technology and many more.

In recent years substantial research is done in the field of distributed multi-agent systems and learning algorithms to solve these systems using biological paradigms. These paradigms include genetic algorithms and neural networks. Recently emphasis is given on the use of artificial immune systems (AIS), modeled on human immune system. AIS are particularly impressive in distributed systems because of properties such as reinforcement learning capability, distributed detection and execution and tolerance to perturbations that otherwise tend to affect performance of the system. This paradigm is called as immune engineering and makes use of immunological concepts to solve engineering problems.

1.2 Scope of the Thesis

This thesis can be divided into two major parts. In one, reinforcement-learning algorithms are used to solve the multi-agent herding problem. In the other part, artificial immune network modeling is used to solve the same problem.

We study reinforcement learning algorithm in the form of Q-learning in detail for multi-agent herding problem. First, Q-learning is studied for a simple herding problem that contains only two players – one pursuer and one evader. This algorithm is then extended to the case of herding problem with multiple agents.

Artificial immune network modeling is designed to solve the same problem, using two paradigms. In one paradigm, group behavior arbitrations are studied using fixed strategies, which have predefined actions. Agents are modeled as components of the artificial immune network. This AIS model is used to study decision making on a macro level, where agents interact with each other to choose broad strategies. In the other paradigm, immune network is designed so that agents learn to choose individual actions, which taken together form various strategies.

Important design considerations studied in both these methodologies are the computational requirements of each, distributed nature and speed of learning process as well as the desirability of solutions obtained by these algorithms.

1.3 Outline of following chapters

Chapter 2 contains a brief discussion about previous work done in the areas of Q-learning and the artificial immune systems.

In chapter 3, dynamic programming solution to basic herding problem is presented. This chapter also discusses details of the herding problem and properties of that system in general.

Q-learning algorithm is used for solving the herding problem is chapter 3. This chapter also introduces dynamical aspects of the system. Q-learning algorithm is introduced in this chapter and various implementation details are discussed. Herding problem is then solved using Q-learning method. Cost-based and reward based Q-learning formulations are presented. A comparison of Q-learning and dynamic programming is also presented.

Multi-agent herding using reinforcement Q-learning is presented in chapter 5. It introduces a formulation of the problem that is independent of the dimension of the problem, which is important for keeping the computational requirements as low as possible. Individual and batch processing are also discussed.

Chapter 6 introduces artificial immune system. Human immune system is briefly discussed and then mathematical aspects and idiotopic network modeling of immune system are discussed.

Group behavior arbitration using artificial immune system is presented in chapter 7. This chapter gives all implementation details of immune network design. Chapter 8 discusses the use of AIS for individual action selection. It also discusses important differences between Q-learning and AIS implementation.

Finally chapter 9 lists important conclusions of this research and future work.

2 LITERATURE REVIEW

Extensive research is done in the field of dynamics programming, and stochastic iterative learning algorithms in the context of multi-agent systems. Various methods for solving reinforcement learning problems, including dynamic programming, TD(λ) are used in [2]. Specific application of dynamic programming to evader herding problem is presented in [3] and it is the starting point for the research work presented in this thesis. This solution is not applicable for multi-agent systems, although it can be extended to this class of systems.

Various methods and strategies that are useful in multi-agent systems are discussed in detail in [4]. These strategies include effective interaction with system environment, deontic logic. It also discusses ways to use evolutionary game theory to solve multi-agent problems. Similar discussion is found in [7] which gives various practically useful strategies that can be used in multi-agent setting. Learning cooperative procedures in multi-agent systems is very crucial. References [5] and [6] discuss various ways to accomplish this goal. These include use of collective memory and strategy knowledge sharing among various agents.

In Q-learning it is important that the system is a Markov decision Process. Reference [9] discusses Markov games for a building a framework for multi-agent reinforcement learning. Detailed implementation and discussion of Q-learning algorithm is done in [13]. It also discusses various practical implementation issues concerning Q-learning algorithm. It also introduces discrete Q-learning formulation for a predator-prey problem, which is similar to the herding problem.

In a multi-agent system, agents may have similar or conflicting goals and the issue of cooperation among agents and that of conflict resolution is a major aspect of reinforcement learning algorithms. Reference [8] introduces a solution for cooperation among various agents. Various strategies are introduced in this paper, including information sharing and knowledge sharing. [11] discusses implementation of reinforcement learning in multi-agent systems for agents with multiple goals and discusses schemes to schedule various goals.

References [12], [14], [18], and [24] present various schemes in cooperative and non-cooperative agent settings.

Issue of dimensionality of Q-learning is discussed in [15] which takes an approach similar to that presented in this work. Region-based Q-learning is discussed in detail in [15], [19], [20]. An exploration technique – Boltzmann exploration – for Q-learning that is used in this thesis is presented in [17]. Initialization of Q-values in the algorithm is another important aspect in Q-learning. Many methods proposed include fuzzy rules implementation [22], [23].

Reference [26] introduces artificial immune system and gives various implementations for a number of applications, ranging from mobile autonomous robots, optimization techniques, control applications and parallel search algorithms. The approach taken in this thesis is based on the implementations given in [29], [30], [32], [33] and [34]. Similar herding problem is solved in [34], using different formulations and modeling of the immune network. The immune network dynamics as used here are taken from [32] and [33].

3 HERDING PROBLEM – DYNAMIC PROGRAMMING FORMULATION

3.1 Introduction

We want to use reinforcement learning algorithms and biologically inspired learning algorithms to solve a class of multi-agent herding problems. We will start with a very basic formulation of the herding problem, with the minimum grid size (3 x 3) and with one pursuer and one evader. Various learning algorithm methods are studied using this basic problem formulation and then these methods are generalized to variable sized grid, in multi-agent setting, which contains more than one active agent (pursuer) and more than one passive agent (evader).

3.2 3 x 3, 1 Pursuer – 1 Evader Herding problem

The system consists of a grid of size 3 x 3 and has two agents - a pursuer and an evader. The evader tries to run away from the pursuer whenever the pursuer is in any of the adjacent cells as that of the evader. The goal of the pursuer is to use this tendency of the evader to herd it to a specified cell, called pen, which is the position (0, 0) on the grid. The game ends when the evader reaches the pen. Pursuer can be anywhere on the grid, except in the cells adjacent to the pen (0, 0), when evader is in the pen, as is allowed by the evader dynamics. It is intuitive to see that only states where evader is in the position (0, 0) and pursuer in either (0, 1) or (1, 0) are not terminal states, because in that case, evader will move out of the pen, as guided by the evader dynamics, discussed later.

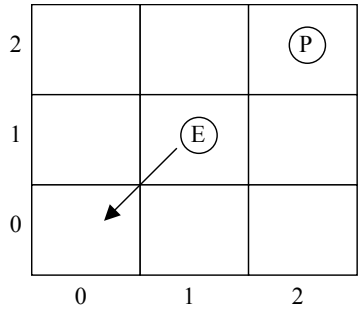


Figure 3.1 Herding Problem Grid

The herding problem can be thought of as a finite state machine, with finite terminal states. All terminal states are zero-cost terminal states, which means that whenever the system reaches any of these states, it remains there at no extra cost (or effort from any of the agents) to remain in the terminal state. The goal of the pursuer in this problem is to herd the evader to the pen, in minimum number of movements of the pursuer and the evader taken together.

3.2.1 Role of Learning algorithms

The aim of the learning algorithms in the herding problem is to allow the active agent (pursuer) to explore the system (i.e. the states of the system) so as to be able to come up with a policy (which is a sequence of choices the pursuer makes) that is optimal in solving the problem. The optimal policy consists of various actions that the pursuer takes from any starting state of the system (i.e. any starting positions of the pursuer and the evader) so as to herd the evader to the pen, with minimum number of moves of the pursuer and the evader, taken together. The evader is a passive agent – that is, it does not learn the policy that would maximize the cost for the pursuer to herd the evader to the pen. Evader just follows the dynamics as described later in the chapter. All actions, either by the pursuer or the evader, are completely deterministic, and hence the problem is a deterministic pursuit-evasion game. Learning algorithms are later extended to the case of multi-agent cases where there are more than one pursuer and evader, in a variable size grid.

3.3 Problem specifications

1. The problem is discrete time, deterministic finite state problem.
2. The system may have a very large number of states, but always has a finite number of states.
3. The system, at any instance, is in any one of the possible finite states.
4. The pursuer and the evader both can occupy any position on the grid.
5. The pursuer and the evader cannot be in the same position, at any given time.
6. Aim of the pursuer is to herd the evader to the pen, i.e. to position (0, 0).

7. Pursuer can move one step in one time step. It can move in any of the directions, i.e. this step can be horizontal, vertical or in diagonal direction. It is true for the evader as well.

8. Whenever either the pursuer or the evader moves, a cost of 1 unit is incurred.

3.4 Problem Formulation

3.4.1 State of the system

At any instant k, the system is completely described by the state of the system. The state of the system consists of all the important information about the system that completely describes the system. The state should consist of as much information as possible, still keeping number of states as small as possible. If we include information that is really not necessary to define the system, we unnecessarily increase the number of states of the system, and for larger and more complicated systems, this approach could lead to prohibitively large number of states. On the other hand, if we try to keep the number of states to the minimum, then the state may not contain all the information that is absolutely necessary. Thus, there has to be a trade-off between the information stored in the state and the number of states of the system. It is very important to know that for efficient learning process, we would like to include as much information about the system as possible, but a very large number of states may lead to very slow learning process, which will not be computationally efficient.

Designing the state information for the above system directs us to include the following data in the state information – size of the grid, location of the pen, position of the pursuer and the position of the evader. Out of the above information, the size of the grid, and the position of the pen are constants, and are hence not important to describe the system dynamics at any instant k. So the state information could only contain the position of the pursuer and the position of the evader. A state of the above system thus consists of the position of the pursuer and the evader. Thus, at any time, state of the system is given by the 4-tuple

$\mathbf{X(k) = \{x_d(k), y_d(k), x_s(k), y_s(k)\}}$, where,

$X(k)$ = state of the system at any instant k,

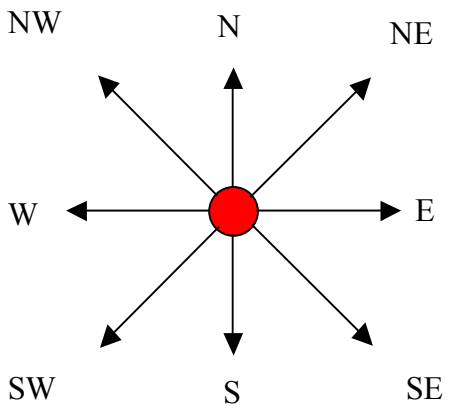
$x_d(k)$, $y_d(k)$ = x and y-coordinates of the pursuer position at instant k,

$x_s(k)$, $y_s(k)$ = x and y-coordinates of the evader position at instant k,

At any given instant, all these coordinates can take a value of either 0, 1 or 2, and thus, pursuer can take 9 positions, and so also the evader. Thus, we have a total of 81 states in the system. Some of these states are not possible. For example, the pursuer and the evader cannot be in the same position at the same instant.

3.4.2 Control set for the Pursuer

The pursuer is the active agent and has a number of control actions available at each state. A control action set is the set of all possible actions from which the pursuer chooses an action that it perceives as the optimal for a given state. All the states have the same control action set, but it is also be possible to have different control actions for different states. The control set is the set of possible directions that the pursuer can move to. From any position, pursuer can move in any of the eight directions shown below, or decide to stay at the same position. Thus, at any state, we have a control set of nine control actions.



CONTROL SET FOR THE PURSUER

E = EAST
NE = NORTH EAST
N = NORTH
NW = NORTH WEST
W = WEST
SW = SOUTH WEST
S = SOUTH
SE = SOUTH EAST
HERE = Stay at the same position

Figure 3.2 Control Action Set for the pursuer

3.4.3 Q-values for State-Action pairs

The reinforcement-learning algorithm that we use is called the Q-learning algorithm. There is one Q-value (Quality value) associated with each state-action *(X, a)* pair. Q-learning algorithm is discussed in detail in the next chapter. For the 3x3 grid, with one pursuer and one evader, we have 81 states and 9 actions for each state, making a total of 81 x 9 = 729 Q-values.

This should be a good indication of the fact that as the systems becomes more and more complex, either by increasing the grid size, and/or by adding more agents, the learning involves more and more Q-values. As an example, following are the figures for various grid sizes, keeping only one pursuer and one evader in the system.

Grid Size	Number of States	Control Actions	Q-values
3 x 3	$3^4 = 81$	9	81 x 9 = 729
5 x 5	$5^4 = 625$	9	625 x 9 = 5625
10 x 10	$10^4 = 10000$	9	10000 x 9 = 90000
15 x 15	$15^4 = 50625$	9	50625 x 9 = 455625

Table 3.1 Number of Q-values for n x n grid, for 1 pursuer - 1 evader

Now if we add multiple agents to the system, the number of Q-values will increase exponentially. For example, if we add one pursuer and include position of the evader and positions of both the pursuers in the state information, following are the numbers of Q-values that we have to deal with.

Grid Size	Number of States	Control Actions	Q-values
3 x 3	$3^6 = 81$	9	81 x 9 = 729
5 x 5	$5^6 = 15625$	9	15625 x 9 = 140625
10 x 10	$10^6 = 1000000$	9	1000000 x 9 = 9000000
15 x 15	$15^6 = 11390625$	9	11390625 x 9 = 102515625

Table 3.2 Number of Q-values for n x n grid with 2 pursuers - 1 evader

3.4.4 Evader dynamics

The evader follows a set of rules to decide its next action. These rules are defined *a priori* and are not learnt by the evader through a process of reinforcement learning. When pursuer is not in one of the adjacent cells, evader does not move. If pursuer is in one of the adjacent cells, evader moves in a direction governed by the dynamics given below. These dynamics are independent of the size of the grid. The grid is partitioned into 4 CORNERS, 4 EDGES and the remaining area, called ELSEWHERE. The condition CORNER signifies that the evader is in one of the corners of the grid (left-top, right-top, left-bottom, right-bottom), condition EDGE signifies that the evader is on one of the edges of the grid (left, right, top, bottom), and condition ELSEWHERE signifies that the condition is neither the CORNER

nor the EDGE. This partitioning of the grid is also independent of the grid size and is used for all grid sizes (greater than 3x3) used in the thesis. At corners and edges, boundary conditions apply that result in different directions of movement than the directions that would be selected under same conditions if the evader were not at any of the borders (i.e. is "ELSEWHERE", not in any of the corners nor on any of the edges).

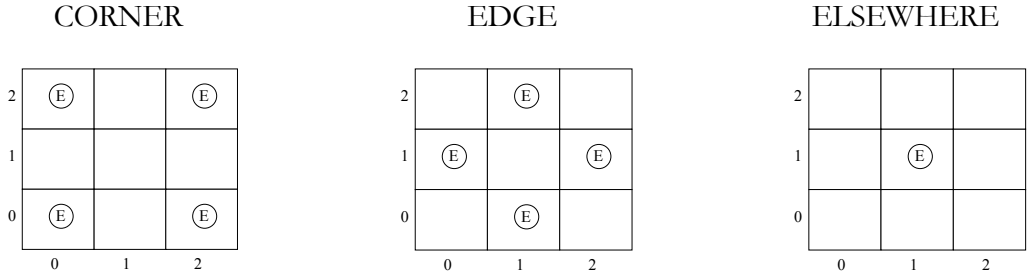


Figure 3.3 Grid Partitioning - CORNER, EDGE and ELSEWHERE Conditions

Following are the evader dynamics rules –

In the following discussions, symbols used are –

Xe_old, Xe_new = old and new x-coordinates of the evader,

Ye_old, Ye_new = old and new y-coordinates of the evader,

Xp_old, Xp_new = old and new x-coordinates of the pursuer,

Yp_old, Yp_new = old and new y-coordinates of the pursuer.

1. When the pursuer is not in the adjacent position, evader does not move.
2. When the pursuer is in any of the adjacent positions, evader moves in a direction governed by the rules, deterministically. There is no probability involved in the direction that evader takes when it encounters the pursuer.
3. ELSEWHERE Condition:

   Xe_new = Xe_old + (Xe_old – Xp)

   Ye_new = Ye_old + (Ye_old – Yp)
4. CORNER Condition:

Common condition at all corners:

IF abs(Xe_old – Xp_old) = 1 AND abs(Ye_old – Yp_old) = 1

THEN Xe_new = Xe_old AND Ye_new = Ye_old

Left-Top Corner AND Right-Bottom Corner:

Xe_new = Xe_old + (Ye_old – Yp)

Ye_new = Ye_old + (Xe_old – Xp)

Right-Top Corner AND Left-Bottom Corner:

Xe_new = Xe_old - (Ye_old – Yp)

Ye_new = Ye_old - (Xe_old – Xp)

5. EDGE Condition:

Common condition at all edges:

IF abs(Xe_old – Xp_old) = 1 AND abs(Ye_old – Yp_old) = 0

THEN Xe_new = Xe_old AND Ye_new = Ye_old

Left Edge AND Right Edge:

Xe_new = Xe_old

Ye_new = Ye_old + (Ye_old – Yp)

Top Edge AND Bottom Edge:

Xe_new = Xe_old + (Xe_old – Xp)

Ye_new = Ye_old

3.5 Final Equilibrium and General Equilibrium States

According to the above evader dynamics, we can categorize the states into following [3] –

3.5.1 Equilibrium State of the evader

At any instant k, evader is said to be in equilibrium position if,

$\forall k \geq T,\ x_e(k) = x_e(T), y_e(k) = y_e(T)$

Put in simple terms, evader is in equilibrium state, when it does not move, i.e. whenever

pursuer is not in any of the adjacent positions as that of the evader.

3.5.2 Final Equilibrium State of the evader

At any instant k, evader is said to be in final equilibrium position if

$\forall k \geq T,\ x_e(k) = 0, y_e(k) = 0$

Put in simple terms, evader is in the final equilibrium state, when it is in the pen and does not move out of it. Following figure shows the final equilibrium states –

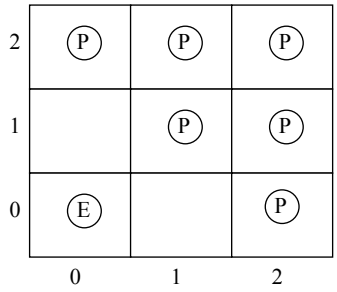


Figure 3.4 Final Equilibrium States

When evader is in any of the general equilibrium states (not the final equilibrium states, reaching which the game ends), it does not move to a new location, and hence pursuer has to take some action so as to make the evader move. Pursuer can choose any of the action from the set of finite number of actions defined above. This set of actions is dependent on the state of the system, e.g. when evader in not in equilibrium position, it takes some action to go away from the pursuer and in effect, pursuer has to take no action but to stay at its current position (in other words, execute HERE control action). In the Q-learning algorithm, pursuer learns to execute the HERE action in non-equilibrium states, but for the dynamic programming solution [3], this action for non-equilibrium states is enforced on the pursuer.

3.6 The Dynamic Programming Solution

We are not going to solve this problem using the DP technique in this thesis, but it is a good exercise to understand how the DP algorithms work and how effective they are in finding an optimal solution to the problem. This problem is solved using DP technique in [3], and all the following details are taken from the same paper.

Let U be the discrete set of actions available to the Pursuer when the system is in the state X. The Pursuer defines a *policy* μ: x$\rightarrow$U that is a mapping from states to actions. We also

define a *value function* $V_\mu(X)$, which is the sum of all future instantaneous costs incurred by the pursuer, given that the initial state of the system is X and the system follows the policy μ. As we know, whenever the pursuer or the evader makes a move a cost of 1 unit is incurred.

3.6.1 Introduction to Dynamic Programming

The dynamic programming solution is based on the Bellman's equation, which is given by –

$$V^*(X(k)) = \min_{u \in \mu(x)} \{c_i(u) + V^*(X(k+1))\}$$

This equation is a mathematical version of what is known as the Bellman's law of optimality, which states –

"An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision"

Important elements of a DP problem are –

1. A discrete-time dynamic system, whose state transitions depend upon control actions taken. At state i, a control action must be chosen from a set U(i). At state i, choice of control u specifies the transition probability Pij(u) to the next state j.
2. A cost J that accumulates additively over time as the system goes from any starting state to any of the final states and depends on the states visited and controls chosen. At the k^{th} state, a cost $\alpha^k g(i, u, j)$ is incurred, where g is a given function and α is the discount factor, which is a scalar such that $0 < \alpha \leq 1$. $\alpha < 1$ means that future costs matter less than the same costs incurred at the present time. This allows negotiating small perturbations in the cost structure of the system.

The aim is to choose a policy μ: x→U that minimizes the cost-to-go from any of the staring state to any of the final states. A policy is a set of control actions that are taken in succession.

A policy is thus a mapping from states to control actions as shown below –

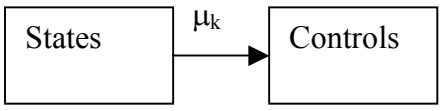


Figure 3.5 Mapping from States to Control Actions

Once a policy is fixed, the sequence of states becomes a Markov chain with transition probabilities given by –

$$P(i_{k+1} = j \mid i_k = i) = p_{ij}(\mu_k(i))$$

In a deterministic system, as the one that we are interested in, no probabilities are involved.

For the system that we are dealing with –

1. Decisions are made in stages (a decision is taken only in one of the possible finite states of the system) [2],
2. Output of each decision is not fully predictable, unless complete digraph of the systems is traced several times (theoretically infinite number of times) [2],
3. Each decision results in some immediate cost,
4. Each decision also affects the context in which future decisions are made, and hence affects the costs incurred in subsequent stages. [2]

According to [2], such DP system can be graphically demonstrated as follows –

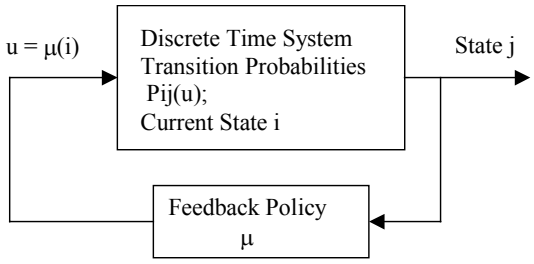


Figure 3.6 DP Algorithm System

To go from current state i to next state j, the cost incurred is g(i, u, j), under the control action u, which is determined from the policy μ. To decide u from a number of possibilities (policies μ), we must not only look for the instantaneous cost g(i, u, k), but also rate the desirability of the next state j. This is done by using the optimal cost (over remaining states until the final or terminal states), denoted by J*(j), starting from the state j.

3.6.2 Value and Policy iteration in DP

The herding problem a deterministic shortest path problem, where there are a few cost-free termination states and once the system reaches one of those states it remains there at no extra cost. The structure of the problem is such that termination is inevitable, at least under the optimal policy [3]. The problem is in effect a finite horizon problem, but the length of the horizon may be random and affected by the policy being used. The essence of the problem is how to reach the final or the terminal state or one of the terminal states with minimum cost incurred.

The DP problem can be solved by using either the method of value iteration or the method of policy iteration.

Value Iteration is the DP iteration that generates a sequence of values, starting from some J. It requires infinite iterations theoretically to completely map the whole cost structure of the digraph, but may terminate finitely under certain circumstances [3]. The estimate of the cost-to-go vector is updated simultaneously for all states, using the Gauss-Seidel method [2].

$$FJ(1) = \min_{u \in U(i)} \sum_{j=0}^{n} P1j(u)\left[g(1,u,j) + J(j)\right]$$

$$FJ(i) = \min_{u \in U(i)} \left[\sum_{j=0}^{n} Pij(u) \cdot g(i,u,j) + \sum_{j=1}^{i-1} Pij(u) \cdot FJ(i) + \sum_{j=i}^{n} Pij(u) \cdot J(i) \right], \quad i = 2,3,...,n$$

Value iteration requires an infinite number of iterations to obtain J*.

In policy iteration, we start with a proper policy μ_0 and go on to generate a sequence of new policies μ_1, μ_2, …. For each policy μk, we perform following steps -

1. Policy Evaluation Step, where we calculate $J^{\mu k}$ for that policy by the following equation -

$$J^{\mu k}(i) = \sum_{j=0}^{n} Pij(\mu_k(i))\left[g(i,\mu_k(i),j) + J(i)\right], \quad i = 1,2,...,n$$

$$J(0) = 0$$

2. Policy Improvement Step, where new policy is calculated using the following equation –

$$\mu_{k+1}(i) = \arg\min_{u \in U(i)} \sum_{j=0}^{n} Pij(u)\left[g(i,u,j), J^{\mu k}(j)\right], \quad i = 1,2,...,n$$

This iteration is repeated until $J^{\mu_{k+1}}(i) = J^{\mu_k}(i), \ \forall i$.

3.7 DP Formulation of the herding problem[1]

3.7.1 Problem Statement

The herding problem can be thought of as a digraph G = (V, E) that consists of a finite number of vertices or nodes V (in this case 81) that represent the states in the system, and a finite set E of edges or connections between the nodes. An edge connects two nodes (states) A→B, if starting in state A and following the dynamics discussed above we can reach state B in one step. The digraph is a directed network, or a weighted digraph, since a cost of 1 is associated with each edge. Since at any state, either the pursuer or the evader can move, and a cost of one is accrued whenever either of them moves, hence each edge is associated with a cost of 1. The adjacency matrix of the digraph is 81 x 81 matrix, whose diagonal elements are all zero. 81 is the number of states in the system and is also known as the cardinality of the digraph N(V).

3.7.2 Properties of the herding problem digraph

1. The digraph is not a strongly connected digraph, but is a weakly connected digraph.
2. Starting from any valid starting state (valid state is any state that is allowed by the rules and the dynamics), one of the final equilibrium states can be reached.
3. All the final equilibrium states have paths connecting them together, as shown in the following figure – arrows show connection directions (which correspond to the actions that the pursuer can take) from one state to another.

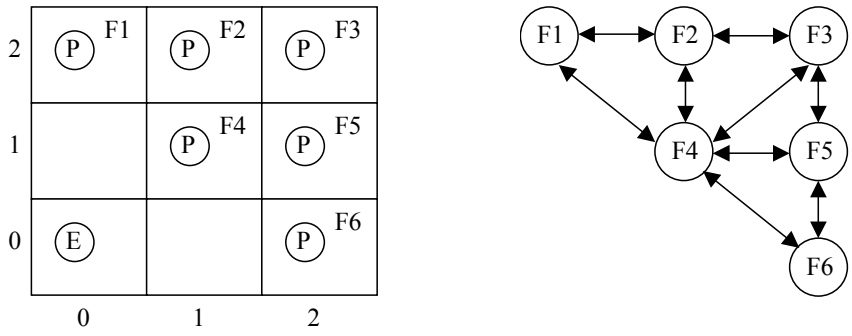


Figure 3.7 Fully connected final equilibrium states

[1] This formulation is based on and taken directly from the solution obtained in [3].

From (2) and (3) we can deduct that, starting from any allowable valid state, there is a path to all final equilibrium states. This shows that the digraph is connected

4. Number of nodes that are adjacent to a node representing an equilibrium state depends on the position of the pursuer. In the ELSEWHERE condition for the Pursuer, there are eight such adjacent nodes, out of which one is not allowed, depending on the position of the evader. In the EDGE condition for the Pursuer, number of such nodes is 5 and in the CORNER condition for the pursuer, the number of such nodes is 3.
5. The number of nodes that are adjacent to a node representing a non-equilibrium node is always 1.
6. Necessary condition for the existence of a value in this pursuit-evasion game is that there should be no cyclic paths for the maximizing player (here the evader) in the digraph. If a cyclic path is possible for the minimizing agent (the pursuer), it will not follow this path, as this cycle will increase the cost to infinity. But evader, if allowed such a path, will follow it as it increases the cost for the pursuer to infinity.

3.7.3 DP Solution to the herding problem

The dynamic programming solution involves following steps –

1. Compute at each new state the maximum (or the minimum) of the values for all possible transitions from previous states.
2. Once the final state is reached, backtrack to find the path(s), which lead to the optimal final state.

Dynamic programming solution thus finds a value function for each possible state in the system, and an optimal policy from each of the possible states to the final state(s). In the herding problem, this value function is the number of movements the pursuer and the evader make. Thus, we get a total of 81 values for 81 states. V-values for all states are represented in a tabular form. The table is interpreted as follows – the values P(0, 0), P(0, 1), … represent various positions of the pursuer and E(0, 0), E(0, 1), … represent positions of the evader. Thus, a cell at the intersection of the row corresponding to P(0, 1) and the column E(1, 1) represents the state X = {0, 1, 1, 1}, and the value in that cell represents the value function for

that state. This value is the cost that will be incurred in going from this state X = {0, 1, 1, 1} to one of the final equilibrium states as defined in the earlier section.

	E(0, 0)	E(0, 1)	E(0, 2)	E(1, 0)	E(1, 1)	E(1, 2)	E(2, 0)	E(2, 1)	E(2, 2)
P(0, 0)	∞	3	4	3	4	6	4	6	7
P(0, 1)	0	∞	3	3	3	6	4	5	6
P(0, 2)	0	1	∞	3	3	6	4	5	6
P(1, 0)	0	3	4	∞	3	5	3	6	6
P(1, 1)	0	2	3	2	∞	5	3	5	6
P(1, 2)	0	1	2	2	2	∞	3	4	5
P(2, 0)	0	3	4	1	3	5	∞	6	6
P(2, 1)	0	2	3	1	2	4	2	∞	5
P(2, 2)	0	2	3	2	1	4	3	4	∞

Table 3.3 V-values for the 3 x 3 1 pursuer - 1 evader herding problem

Each state thus, has a sigle V-value associated with it, which is an indication of the distance (not in the exact literal meaning) of that state from the final equilibrium state(s). Thus, a state with higher V-value if farther from the final equilibrium state(s) than some other state that has a lower V-value than the former.

3.8 Decision Making in DP method

Once we run the DP algorithm on the digraph of the problem, we get a complete solution to the problem. Thus, at every state the system is in, the active agent (here pursuer) makes a decision that is based on the knowledge obtained by running the DP algorithm. At each state, it chooses that action that will take the system to the most favorable state among the states that can be reached from the current state. The most favorable state will be the state with the lowest of the V-values among all reachable states.

3.9 Simulation Software

Simulator software is developed in Visual Basic to study the DP formulation of the herding problem. This software lets user make changes in the evader dynamics and study the pursuer behavior.

3.10 Conclusion

In this chapter a detailed outline of the very basic form of the herding problem (3 x 3 grid size with 1 pursuer and 1 evader) is presented. Also, a DP formulation of the problem is discussed, along with brief introduction to the DP algorithm. Software was developed for the

DP formulation of the problem, details of which are given in Appendix I. In the next chapter we will solve the herding problem using reinforcement learning algorithms and will then discuss the similarities and the differences of those algorithms with the DP algorithm.

4 RL FORMULATION (Q-LEARNING)

4.1 Introduction

In this chapter, we will solve the herding problem using reinforcement learning (RL) algorithms, which are on-line (agent learns as it experiences the world) unlike off-line DP algorithms. DP method for obtaining an optimal policy for the herding problem assumed detailed model was available. The model consisted of knowledge of all the states of the system, state transition probabilities and the underlying cost structure. Reinforcement learning is primarily concerned with how to obtain the optimal policy when such a model is not known in advance. The agent must interact with its environment directly to obtain information which, by means of an appropriate algorithm, can be processed to produce an optimal policy.

The agent identifies the current state, takes an action so that the system goes to another state. Depending upon the desirability of the new state, agent receives a reward or penalty (reinforcement) which updates the Q-value associated with the lastest state-action pair.

Q-learning algorithm illustrates that learning an explicit world model is not necessary for the central purpose of learning optimal actions and that the agent can perform adaptively in a world without understanding it [13].

4.2 Q-Learning

Q-learning algorithm works by using Q-values, which are approximations of the V-values in the DP algorithm. Q-learning algorithm assumes that the system can be modeled as a Markov Decision Process (MDP), in which an adaptive agent interacts with the environment. In a multi-agent system, other agents form a part of the environment which the adaptive agent experiences.

An MDP is defined by a set of states X and actions U. A transition function defined as $T : X \times U \rightarrow PD(X)$, models the effects of various actions on the state of the environment [17]. PD(.) is the probability distribution of state transitions, which are the probabilities of choosing actions in the given state. The reward function $R : X \times U \rightarrow \Re$ specifies the agent's task [17].

The objective of the adaptive agent is to find a policy mapping from the set of observed states X to the set of actions U, from the interactions with the world, so as to maximize the reward. Normally, this reward is a discounted reward, by a discounting factor γ ($0 \leq \gamma < 1$). This discounting factor determines the effects of future rewards on the optimal policy. The Q-value (Quality-value) of a state-action pair is the estimated cumulative discounted reward received by taking that action and following the optimal policy thereafter.

Following assumptions are made about the system for applying the Q-learning algorithm -

1. Number of states and actions in the system is finite,

2. All states are fully observable,

3. All decisions in any state result in one of the possible states (i.e. state map is closed),

4. No feasible state-action pair is ignored in the learning algorithm.

4.3 The Q-Learning Algorithm

The quality value of a state-action pair, given by *Q(x, a)*, is the total expected discounted reward accrued by the non-stationary policy[1] that takes an action $a \in U$ at state $x \in X$ and then follows the optimal policy after that.

Mathematically Q-learning algorithm is as follows –

$$Q(x,a) := (1-\alpha)\cdot Q(x,a) + \alpha \cdot \left[r + \gamma \max_{b \in U} Q(y,b) \right]$$

$$Q^*(x,a) = V(x) = \sum_a \Pr(a \mid x) \cdot \max_{a \in U} Q(x,a)$$

where $\Pr(a \mid x)$ is the probability that the agent will choose action *a* at the state *x*.

In the above equations α is the learning rate, γ is the discounting factor, and r is the reward the agent gets after executing action *a* at the state *x*. At each state *x*, agent takes an

[1] The policy is evolving during the learning of the agent and hence is not stationary.

action *a* so that the system goes to new state y. Agent then receives a reward r and accordingly updates the Q-value for the initial state-action pair (x, a). If each action is executed in each state an infinite number of times on an infinite run and learning rate is decayed appropriately, the Q values will converge with probability 1 to Q^*. The actual values that the Q-values will take will differ according to the reward function, and with the discount factor.

4.3.1 Exploration and Exploitation

When the Q values are nearly converged to their optimal values, it is appropriate for the agent to act greedily, taking in each state, the action with the highest Q-value. But in the early stages of learning, exploration of the system state-space and the action-set is more important so that the agent can build up a comprehensive map of the state-action pairs of the system. As this knowledge increases, exploration should decrease and exploitation should increase. Two different exploration strategies are used in solving the herding problem. They are - the random exploration policy and the Boltzmann exploration.

In random exploration, the agent at every time instant chooses an action *a* in the given state *x*, using a probability distribution that is directly proportional to the Q-values of the state-action pairs involved. The probability of choosing an action *a* in the state *x* is given by

$$\Pr^{rnd}(a \mid x) = \frac{Q(x,a)}{\sum_a Q(x,a)}.$$

In Boltzmann exploration, the probability of choosing an action *a* is given by –

$$\Pr^{bltz}(a \mid x) = \frac{e^{\frac{Q(x,a)}{T}}}{\sum_a e^{\frac{Q(x,a)}{T}}}$$

T is called the virtual temperature of the system, and we will set this to 1 in our simulations. Normally, an annealing technique is used in which T is assigned a large value initially and then it is lowered according to some annealing schedule.

4.3.2 The Reward function

As the agent takes an action in the current state, system goes to a new state. Reward function is used to reinforce the desirability of the new state, and the action taken in the previous state is reinforced using the reward function. If the new state is desirable, a positive reward is given, otherwise a penalty (i.e. a negative reward) is given to the last system-action pair and accordingly corrosponding Q-value is updated. It is important to design the reward function in such a way that it represents the goal for the agents, and need not be very complicated so as to map the whole world into the agent knowledge. Rewards can be associated with states, so that if an action takes the system to a better state, reward is higher than other action if it ends up in a not-so-good state.

It is difficult to predict what behavior will lead to maximizing the reward. Reward functions are the "black art" of Reinforcement Learning, the place where design comes in [13]. Many unintuitive and arbitrary reward schemes have been proposed after a long trial-and-error, but still it remains a trial-and-error method to a great extent.

4.3.3 Discounting factor and learning rate

Discounting factor regulates the effect the optimal Q-value of the next state has on the emerging policy that is built up in a number of time steps. Thus, if dicsounting factor is smaller, it gives more weight to the reward than the optimal Q-value of the next state. Learning rate decides how much weight is to be given to the old Q-value. Typically, simulation starts with $\alpha = 1$, which wipes out any old Q-value, and it is then reduced in a systematic way. As α decreases, the Q-value is building up an average of all experiences, and the odd new unusual experience won't disturb the established Q-values much [13]. As the time $t \rightarrow \infty$, $\alpha \rightarrow 0$, which means that there is no more learning and the Q-values have converged to a final value.

The scheme used in the simulation is to monotonously decrease the value of α, uniformly for all state-action pairs. For any epoch, learning rate will be same for all state-action pairs. This scheme can be formulated as –

$\alpha_n(x,a) = \delta \cdot \alpha_{n-1}(x,a),$
where δ *is the decay factor and* $0 < \delta < 1$

4.4 Herding problem formulation

Two different paradigms are used to solve the herding problem. In one paradigm, we use cost (which is negative reward), and thus the agent seeks that action that has minimum Q-value. This paradigm does not violate the general Q-learning law as maximizing positive reward is same as minimizing negative reward (cost). In the other paradigm, instead of cost, we use a delayed reward function that rewards the pursuer when it herds the evader to the pen.

In both methods, state information that an agent gets is given by –

$X(k) = \{x_p(k), y_p(k), x_e(k), y_e(k)\}$, where,

$X(k)$ = state of the system at any instant k,

$x_p(k)$, $y_p(k)$ = x and y-coordinates of the pursuer position at instant k,

$x_e(k)$, $y_e(k)$ = x and y-coordinates of the evader position at instant k.

Also, at each state, pursuer has 9 possible control actions, which are moving in 8 different directions, or to stay at the same position, as described in the chapter on DP solution.

4.5 Cost-based Q-learning solution

The important aim of using this paradigm is to verify that the Q-learning algorithm converges and that it converges to the exact same optimal solution as obtained in the DP formulation. We will also show that this algorithm gives the optimal Q-values that are exactly equal to the V-values obtained in the DP formulation.

4.5.1 Cost (negative reward) structure

Reward = 0;
If pursuer moves one step Reward := Reward – 1;
Else Reward := Reward;
If evader moves one step Reward := Reward – 1;
Else Reward := Reward;

Learning rate and discounting factor are set to values $\alpha = 1$, and $\gamma = 1$.

The Q-learning algorithm can be expressed as –

$Q(x,a) := r + \min_{b \in U} Q(y,b)$, where r is the reward, which is always either 0 or –1 per step.

We use greedy policy to determine the next action for the given state. Given a state, pursuer chooses the action with the minimum cost. If there are a number of actions with the same minimum Q-value, one of these actions is chosen randomly.

4.5.2 Q-value initialization

Q-values for all feasible state-action pairs are initialized to zero. Any random initialization would also work since at the start of the simulation learning rate α is 1 and hence any initial Q-value is wiped out.. There are a number of states that are not allowed – such as pursuer and evader being in the same position. Also for many allowed states, there are a number of actions that are not allowed due to boundary conditions and other dynamics. Q-values for all infeasible state-action pairs are initialized to a value ∞. In actual simulation, a very large value (100,000) is used to represent ∞. Only exception to this initialization scheme is the initialization of Q-values corrosponding to the boundary conditions, which are initialized to 0, similar to all feasible / allowed state-action pairs. This allows the pursuer to learn that some actions are not allowed due to the boundary conditions. Whenever a boundary condition is encountered, corrosponding Q-value is set to ∞.

4.5.3 Pseudo-code for cost-based Q-learning implementation

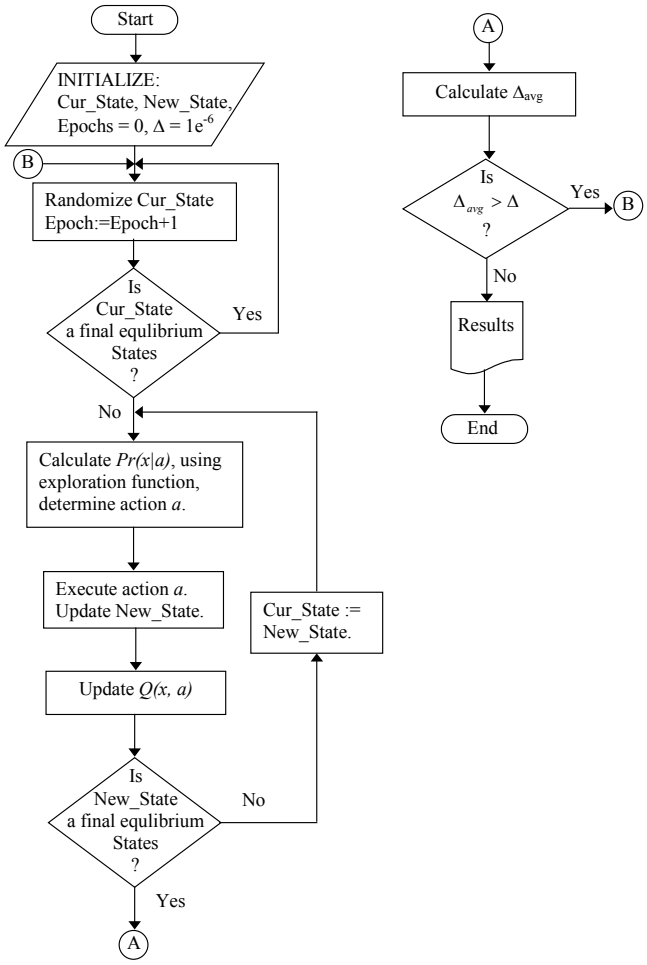


Figure 4.1 Flowchart for the Q-learning algorithm

4.5.4 Simulation, results and discussion

Following are the optimal Q-values obainted by the simulation, after convergence.

	E(0, 0)	E(0, 1)	E(0, 2)	E(1, 0)	E(1, 1)	E(1, 2)	E(2, 0)	E(2, 1)	E(2, 2)
P(0, 0)	∞	3	4	3	4	6	4	6	7
P(0, 1)	0	∞	3	3	3	6	4	5	6
P(0, 2)	0	1	∞	3	3	6	4	5	6
P(1, 0)	0	3	4	∞	3	5	3	6	6
P(1, 1)	0	2	3	2	∞	5	3	5	6
P(1, 2)	0	1	2	2	2	∞	3	4	5
P(2, 0)	0	3	4	1	3	5	∞	6	6
P(2, 1)	0	2	3	1	2	4	2	∞	5
P(2, 2)	0	2	3	2	1	4	3	4	∞

Table 4.1 Optimal Q-values obtained in the cost-based Q-learning algorithm

These values are equal to the V-values obtained by the DP algorithm solution. Important thing to recall is that for this Q-learning algorithm, we assumed that we knew the underlying cost structure completely and that all the experiences were weighed equally. Following table gives corrosponding optimal actions in various states (0 = HERE, 1 = NW, 2 = N, 3 = NE, 4 = E, 5 = SE, 6 = S, 7 = SW, 8 = W).

	E(0, 0)	**E(0, 1)**	**E(0, 2)**	**E(1, 0)**	**E(1, 1)**	**E(1, 2)**	**E(2, 0)**	**E(2, 1)**	**E(2, 2)**
P(0, 0)	-	3	3	3	2, 4	3, 4	3	2, 3	2, 3, 4
P(0, 1)	3	-	3	3, 4	3	4, 5	3, 4, 5	3	3
P(0, 2)	0	0	-	4, 5	4	5	4, 5	4	4
P(1, 0)	2	2, 3	1, 2, 3	-	3	3	3	1, 2	3
P(1, 1)	0	1, 2	2	4, 5	-	3, 4	4	2, 3	2, 4
P(1, 2)	0	0	0	5	4	-	5	0	0
P(2, 0)	0	1, 2	1, 2	0	2	2	-	1	2
P(2, 1)	0	1	1	0	2	0	0	-	0
P(2, 2)	0	8	8	6	0	0	6	0	-

Table 4.2 Optimal Action choices in the state-space, obtained in cost-based Q-learning

4.5.5 Q-learning Convergence in simulation

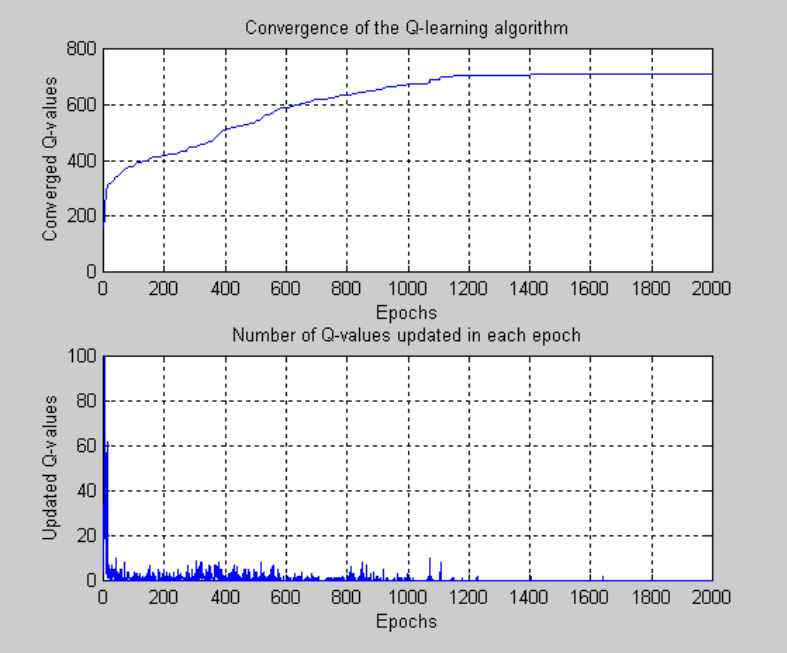


Figure 4.2 Q-learning convergence

As the Q-learning algorithm progresses, more and more Q-values corresponding to state-action pairs start converging to their final values and thus the number of Q-values changed in each epoch decreases.

The first figure shows the number of Q-values that reach their final values. At the start of the simulation, the number of converged Q-values is not zero, but a finite number. This is because, Q-values for infeasible and disallowed state-action pairs are initialized at the start.

The second figure shows the number of Q-values updated during each epoch. During initial runs of the simulation, number of Q-values updated is very large as compared to the number of Q-values updated as the simulation progresses. As the number of Q-values that have converged increases, the number of Q-values that are updated decreases. There is no one-to-one corrospondance between the number of converged Q-values and the number of Q-values that are updated during each epoch.

For various simulation runs, the curves we obtain for the Q-values convergence are different, even though they are similar to each other.

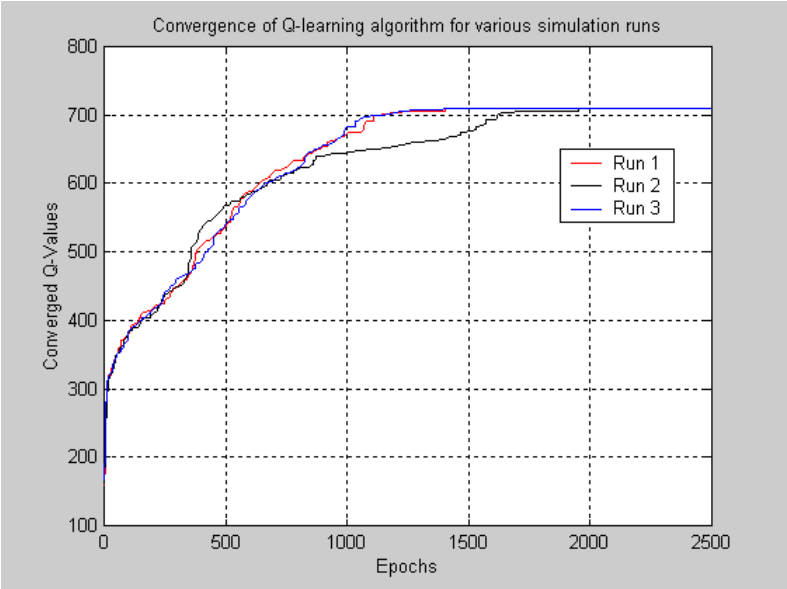


Figure 4.3 Convergence during various Q-learning runs

These variations are due to the inherent randomness in initializing the starting states in epochs, and randomness in choosing actions that have the same minimum Q-values.

4.5.6 State-Action trajectories of the system

State-Action trajectories are the curves for all possible actions for a certain state. These curves show the track of how the corrosponding Q-values are updated. Following figure shows the state-action trajectories for the state {0, 0, 2, 2}. In this case, pursuer is at the position (0, 0) and evader is at (2, 2). Because of the boundary condition restrictions, pursuer can move in only three directions – NORTH (N), NORTH-EAST (NE) and EAST (E).

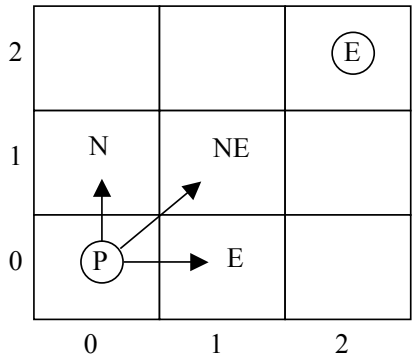


Figure 4.4 Allowed actions for the state X = {0, 0, 2, 2}

Thus, for the given state {0, 0, 2, 2}, there are three state-action trajectories, which are shown in the following figure.

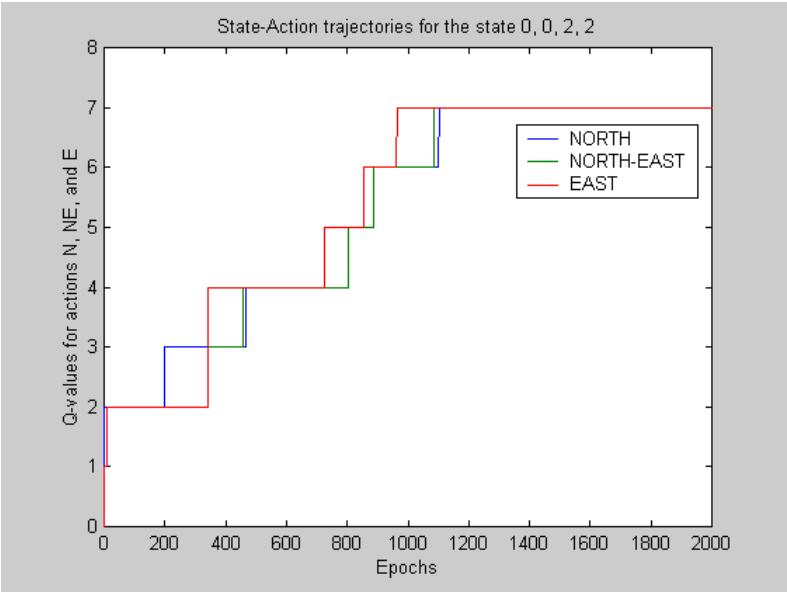


Figure 4.5 State-Action trajectories for the state X = {0, 0, 2, 2}

This state {0, 0, 2, 2} is unique in the sense that all the allowed actions converge to the same Q-value. Thus for this state, any action chosen is an optimal action. Also, all these actions result into symmetric paths for pursuer and evader, if optimal actions are chosen for all succeeding states. They are symmetric about the diagonal (0, 0)-(2, 2).

Following figures show the optimal paths for the state X = {0, 0, 2, 2} for all the three optimal actions. Whenever pursuer is in any adjacent cell to that occupied by evader, evader moves while pursuer stays at the same position. When they are not in adjacent cells, pursuer moves. Red arrows show pursuer movements while green arrows show evader movements.

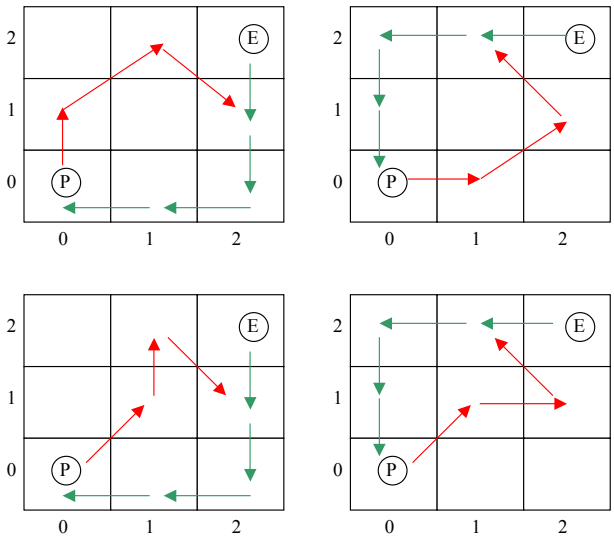


Figure 4.6 State-Action trajectories during various simulation runs

4.5.7 Performance improvement

As the Q-learning progresses, the performance of the agent improves qualitatively. The behavior of the agent tends more and more to the optimal behavior. To varify that this behavior is indeed observed in our simulations, we check the behavior of the pursuer, with the starting state being X = {0, 0, 2, 2}. Various Q-matrices are used as the knowledge base for the pursuer. Pursuer uses the greedy search to find the best action for the current state. Q-matrices used were those obtained after 0, 100, 200, 300, … upto 2500 epochs. Following figure shows the number of steps required for the pursuer to herd the evader to the pen, starting in the state X = {0, 0, 2, 2}, for different runs.

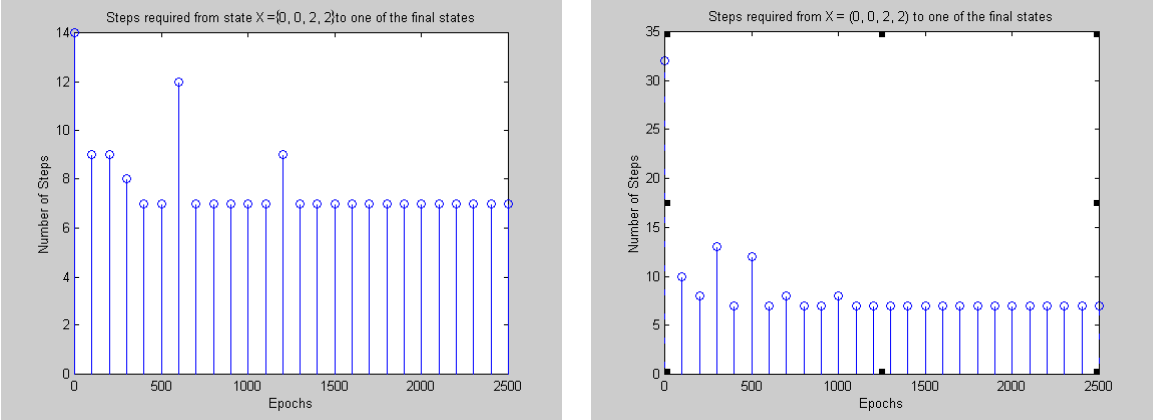


Figure 4.7 Number of steps required ending an epoch for two different simulation runs

There is a randomness in the number of steps taken to herd the evader, in the initial stages of the Q-learning algorithm. This is because of the random initial states for the epochs in every simulation run. When the number of epochs completed is small, the knowledge gained by the pursuer is less as compared to when the number of epochs is large. Hence, the number of steps required to complete the epoch is larger in the earlier stages of the algorithm.

4.5.8 State-Action trajectories and action selection

Following table gives the relation between the state-action trajectories and the action selection for the state {0, 0, 2, 2}. Table shows Q-values for all actions possible for the starting state X = {0, 0, 2, 2} and shows the action that was chosen. In all cases, the action with the smallet Q-value is chosen and when two or more actions have the same minimum Q-value, one of these actions is chosen randomly.

Epochs	Steps	Action Choice	Q(X, NE)	Q(X, N)	Q(X, E)
0	32	N	0	0	0
100	10	NE	2	2	2
200	8	N	3	2	2
300	13	N	3	3	3
400	7	E	4	4	4
500	12	N	4	4	4
600	7	N	5	4	4
700	8	E	5	5	6
800	7	E	5	6	6
900	7	NE	6	6	6
1000	8	N	7	7	7
1100	7	E	7	7	7
1200	7	NE	7	7	7
1300	7	E	7	7	7
1400	7	N	7	7	7
1500	7	NE	7	7	7
1600	7	NE	7	7	7
1700	7	NE	7	7	7
1800	7	N	7	7	7
1900	7	N	7	7	7
2000	7	N	7	7	7
2100	7	E	7	7	7
2200	7	NE	7	7	7
2300	7	E	7	7	7
2400	7	N	7	7	7
2500	7	E	7	7	7

Table 4.3 Action selection and the Q-values for state-action pairs

4.6 Herding problem using Reward functions

In this section, we will solve the herding problem, and study in detail the effects of different exploration strategies and different values of parameters α and γ. Also, effects of random exploration and Boltzmann exploration are studied. Small state-space of the 3 x 3 herding problem allows us to study such properties quantitatively as well as qualititively, unlike in a multi-agent problem where state-space is very large. The results that we obtain from studying this simple system will be used in more complex formulations of multi-agent herding problem.

4.6.1 RL Problem formulation

The only difference in this formulation and the cost-based formulation is that we do not know the underlying cost structure and hence, the model of the system. We will device a reward-penalty function for the Q-learning algorithm.

4.6.2 RL Reward-Penalty function

Generally reward-penalty functions can be categorized into two categories – those with 1 or 2 rewards and those with more than 2 rewards.

First category of reward functions can be represented by following rule [13] –

Reward: if (good event) r else s, r > s

What actual values r and s take is not relavent in such reward functions. Whatever values r and s may take, agent will learn the same policy, as long as r > s. The cost structure in the cost-based herding problem formulation uses a cost of 1 (i.e. a reward of –1) whenever either the pursuer or the evader moves. It is a reward function with only 2 reward values (0 and –1). Any negative number, other than -1, will result in the same optimal policy.

Reward functions with more than 2 rewards can take a form as follows –

Reward: if (A event) reward := r
Else if (B event) reward := s
Else if (C event) reward := t

In such a case, relative differences between reward values will lead to different policies, which will be optimal in their own sense for the specified reward function.

A delayed reward function is used for solving the herding problem. In this formulation, reward is given only when pursuer herds the evader to the pen. Thus, the reward comes only at the end of the epoch. Whenever evader is herded to the pen, a reward of r = 1 is given to the state-action pair that resulted in the herding of the evader to the pen. There is no penalty given for a bad action.

This simple delayed reward function is designed to avoid complex functions that require more that 2 reward values (later in the section, same problem is solved with a 3-value reward function). The aim is to have a two-valued reward function. Also, no penalty is designed because in a two-reward function, no penalty (0 value for penalty) is equivalent to having some other value for the penalty as long as *reward* > *penalty*. Thus, the reward function pseudo-code is as follows –

Reward: if (Evader herded to the pen) r = 1
Else r = 0

Following are the parameter values used for the learning rate and the discounting factor

$$\gamma = 0.9$$

$$\alpha = 1, \textit{with decay rate for } \alpha = 10^{\log 0.001 / 10000} = 0.999309463$$

The decay factor is chosen so that at the start of the simulation α is 1, and after the end of 10,000 epoch, it reaches a value of 0.001. Value of α is modified by the value (α decay).

$$\alpha_n = \delta \cdot \alpha_{n-1}, \ \textit{where } \delta \textit{ is the decay rate}$$
$$\therefore \alpha_n = \delta \cdot \alpha_{n-1} = \delta \cdot (\delta \cdot \alpha_{n-2})$$
$$\therefore \alpha_n = \delta^{n-1} \cdot \alpha, \ \textit{where } \alpha \textit{ is the initial learning rate at the start of the simulation.}$$

4.6.3 Simulation, results and discussion

The algorithm is exactly same as in the flowchart of fig 4.1. Initialization of Q-values is also similar to cost-based Q-learing, with the only difference that Q-values for infeasible state-action pairs are initialized to a value -∞ (i.e.-100,000).

4.6.4 Results for Q-learning with Random Exploration

The simulation was run for 10000 epochs. Following table gives the optimal actions selected by the pursuer (0 = HERE, 1 = NW, 2 = N, 3 = NE, 4 = E, 5 = SE, 6 = S, 7 = SW, 8 = W).

	E(0, 0)	**E(0, 1)**	**E(0, 2)**	**E(1, 0)**	**E(1, 1)**	**E(1, 2)**	**E(2, 0)**	**E(2, 1)**	**E(2, 2)**
P(0, 0)	-	3	3	3	2	3	3	3	3
P(0, 1)	3	-	3	4	3	4	3	3	3
P(0, 2)	0	0	-	5	4	5	4	4	4
P(1, 0)	2	2	3	-	3	3	3	2	3
P(1, 1)	0	1	2	5	-	3	4	3	2
P(1, 2)	0	0	0	5	4	-	5	0	0
P(2, 0)	0	1	2	0	2	2	-	1	2
P(2, 1)	0	1	1	0	2	0	0	-	0
P(2, 2)	0	8	8	6	0	0	6	0	-

Table 4.3 Optimal Actions obtained in Q-learning with random exploration

Following figure gives the State-Action trajectory for the state X = {0, 0, 2, 2}.

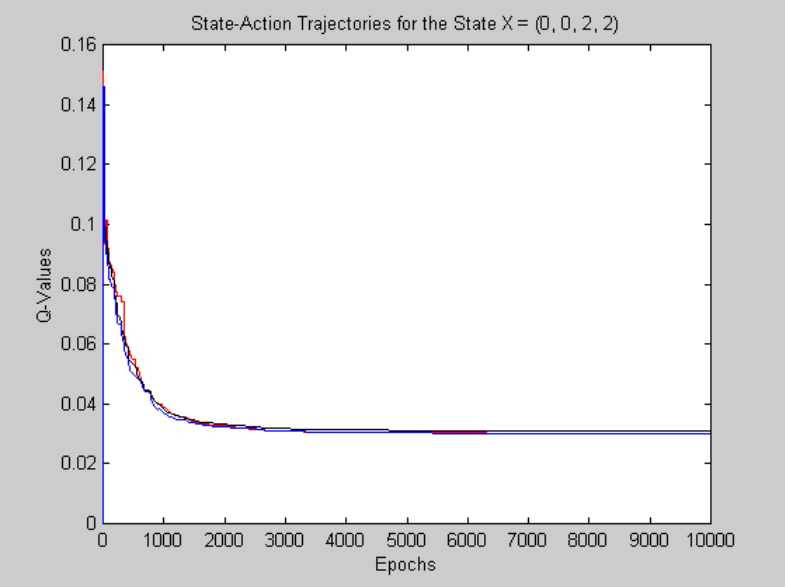


Figure 4.8 State-Action trajectories for state X = {0, 0, 2, 2} in Q-learning with random exploration

4.6.5 Results for Q-learning with Boltzmann Exploration

The simulation was run for 10000 epochs. Following table gives the optimal actions selected by the pursuer (0 = HERE, 1 = NW, 2 = N, 3 = NE, 4 = E, 5 = SE, 6 = S, 7 = SW, 8 = W).

	E(0, 0)	**E(0, 1)**	**E(0, 2)**	**E(1, 0)**	**E(1, 1)**	**E(1, 2)**	**E(2, 0)**	**E(2, 1)**	**E(2, 2)**
P(0, 0)	-	3	3	3	2	3	3	3	3
P(0, 1)	3	-	3	4	3	4	3	3	3
P(0, 2)	0	0	-	5	4	5	4	4	4
P(1, 0)	2	2	3	-	3	3	3	2	3
P(1, 1)	0	1	2	5	-	3	4	3	2
P(1, 2)	0	0	0	5	4	-	5	0	0
P(2, 0)	0	1	2	0	2	2	-	1	2
P(2, 1)	0	1	1	0	2	0	0	-	0
P(2, 2)	0	8	8	6	0	0	6	0	-

Table 4.4 Optimal actions learnt by the pursuer after the simulation, using Boltzmann exploration

The State-Action trajectories for the state X = {0, 0, 2, 2} are shown in the following figure –

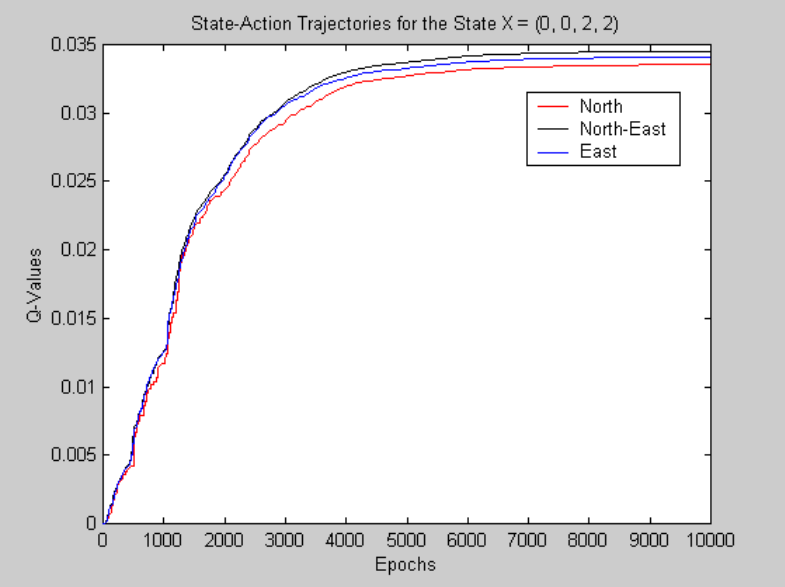


Figure 4.9 State-Action trajectories for state X = {0, 0, 2, 2} in Q-learning with Boltzmann exploration

4.6.6 Optimal Path for RL Q-learning simulation

Following figure shows the optimal path taken by the pursuer to herd the evader to the pen. Unlike the four possible paths that the pursuer could take in cost-based Q-learning simulation, here there is one and only one optimal path.

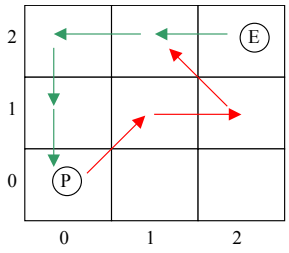


Figure 4.10 Optimal path obtained by RL Q-learning

4.7 Q-learning with Boltzmann and Random exploration

1. Both methods converged to some Q-values for all state-action pairs in the whole state-space of the Q-matrix.

2. Both methods found the same optimal actions (not the Q-values) for all possible states in the state-space.

3. Both methods have one and only one optimal action for every possible state, i.e. the optimal solution set obtained by these two methods is a subset of the actual optimal solution set that was obtained by the DP algorithm and by the cost-based Q-learning algorithm.

4. As seen in the State-Action trajectories of the representative state X = {0, 0, 2, 2}, eventhough there is only one optimal solution (action North-East), final Q-values after convergence for the other state-action pairs (North and East actions- which are in the set of all optimal solutions obtained by the DP formulation) are very close to the Q-value of the optimal action (NORTH-EAST). Thus, both Q-learning algorithms explore the state-action space in its totality, unlike as in the case of the greedy search.

5. Boltzmann exploration takes more epochs for convergence as compared to the Random exploration. Following figures show the steps required by the pursuer to herd the evader to the pen. The pursuer was given Q-matrices which were obtained after 100, 200, …, 10000 epochs.

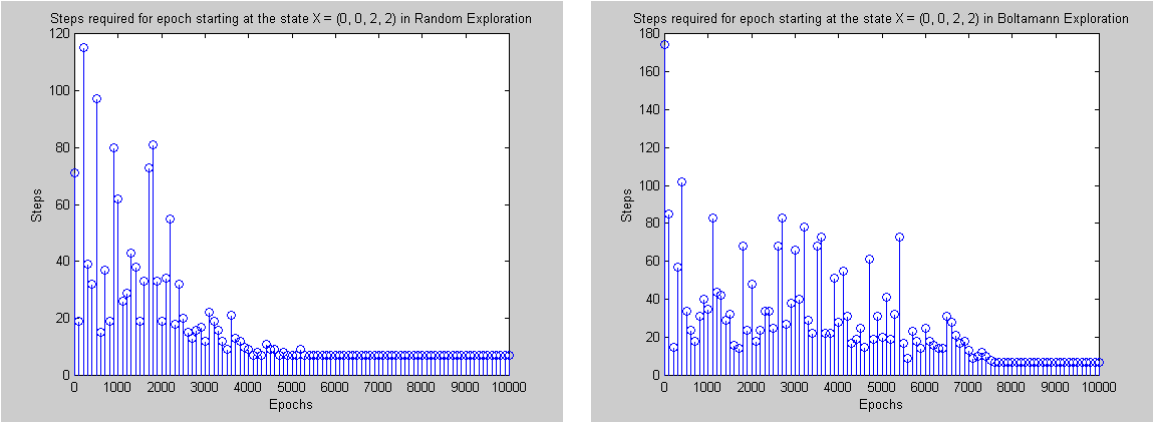


Figure 4.11 Number of steps for completing one epoch, for Random and Boltzmann exploration Q-learning

6. The optimal solution set obtained in both exploration methods is the same and hence, Q-learning is insensitive to exploration strategy used and the solution depends solely on the learning parameters and the reward function.

4.8 Boltzmann and random exploration

Boltzmann exploration gives unequal weightage to Q-values in the matrix while calculating the probability distribution for the actions, whereas in Random exploration, probability distribution is directly proportional to the Q-values in the matrix. Following table gives the probability distribution for Q-values of the state X = {0, 0, 2, 2}. These Q-values are the Q-values obtained after 1000 epochs in the cost-based algorithm.

Q-Value (for action N, NE and E resp.)	4	6	5
Boltzmann Probability	0.09	0.6652	0.2447
Random Probability	0.2667	0.4	0.3333

Table 4.5 Probability distribution

Using the flywheel method to probabilistically find the next action, Boltzmann exploration favors the action with the largest Q-value more than so in the case of random exploration. Boltzmann exploration is an exploitation paradigm (which is similar to the greedy search) whereas Random exploration has more exploration in initial stages of Q-learning. Near convergence stage, random exploration tends more towards exploitation. Boltzmann exploration in effect ensures sufficient exploration while still favoring actions with higher Q-values.

4.9 Q-learning using 3-Value reward function

In this section, 3 x 3 herding problem is solved using a 3-value reward function.

4.9.1 Simulation parameters

$$\gamma = 0.9$$

$$\alpha = 1, \textit{with decay rate for } \alpha = 10^{\log 0.001 / 10000} = 0.999309463$$

4.9.2 Reward function

Xp_New, Xp_old = new and old x-coordinates of pursuer

Yp_New, Yp_old = new and old y-coordinates of pursuer

Xe, Ye = x and y-coordinates of evader

Cur_Dist, New_Dist = current and new distances between pursuer and evader

Cur_Dist is calculated at the same time when current state information is gathered.

Start Reward Funtion

Reward = 0;

New_Dist = max[abs(Xs – Xd_New), abs(Ys – Yd_New)];

IF New_Dist < Cur_Dist reward := reward + 1

ELSE IF New_Dist = Cur_Dist reward := reward + 0

ELSE IF New_Dist > Cur_Dist reward := reward - 1

IF Xs = 0 AND Ys = 0

IF abs(Xs – Xd_New)>1 OR abs(Ys – Yd_New)>1 reward := reward + 5

END IF

End Reward Function

4.9.3 Simulation Results and Discussion

The optimal solution set (optimal actions for every state) is exactly the same as obtained in 2-value reward function Q-learning simulations. Major differences between the 3-value simulation and the 2-value simulation are –

1. Convergence speed,
2. Confidence level for the optimal action set.

Following figure shows the State-Action trajectories for the state X = {0, 0, 2, 2}.

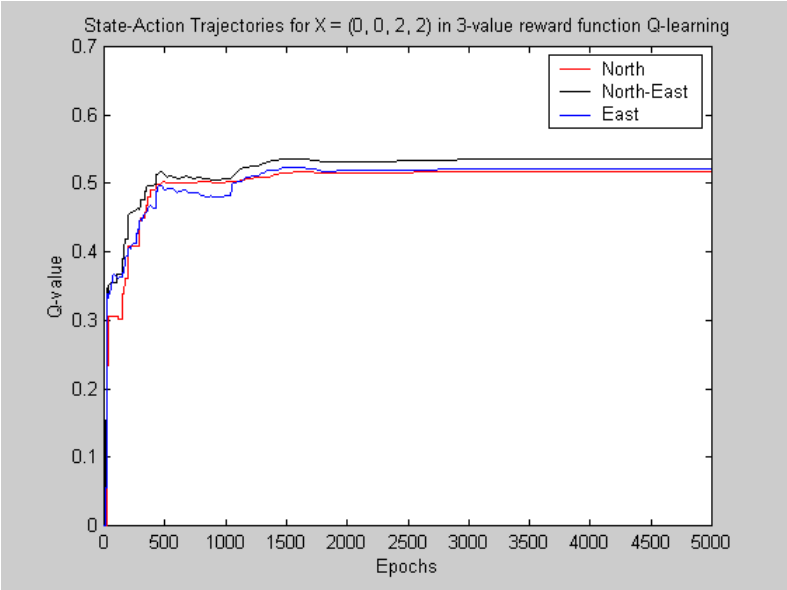


Figure 4.12 State-Action trajectories in 3-value Q-learning

4.9.4 Convergence Speed

Q-learning simulation using a 3-value reward function converges significantly faster as compared to Q-learning simulations using 2-value reward function (former is almost 2-3 times as fast as the later). This speed up in the performance is the result of 3-value reward function, which contains far more information about the system than the 2-value reward function. Thus, as more and more information about the system is coded in the form of a reward function, Q-learning converges faster. This additional information coded in the reward function guides the agent adaptation in a specific manner towards the goal.

4.9.5 Confidence level for the solution set

In 3-value reward function simulation, the optimal solution (set of actions for all states of the system) has a much higher value than other possible actions for all sets. This is easily seen in figures 4.12, 4.9 and 4.8. In all these simulations, for the state X = {0, 0, 2, 2} the only optimal action is action NORTH_EAST. The percentage difference between Q-values of this action and the other two is larger in case of 3-value simulation In 2-value simulations these values are fairly close to each other. Following table gives these Q-values, and the confidence level for the optimal action selection, in Random and Boltzmann exploration setting.

	NORTH	NORTH_EAST	EAST	Optimal Action	Confidence Level
3-value Boltzmann	0.518144	0.540795	0.521809	NORTH_EAST	0.3380
2-value Random	0.030205	0.0309585	0.0301545	NORTH_EAST	0.3333
2-value Boltzmann	0.033546	0.034530	0.033089	NORTH_EAST	0.3335

Table 4.6 Confidence measures of optimal actions

Even though the change in the confidence level is not dramatically large, it is an important result and shows that as we include more and more information in the reward function, better defined solutions are obtained. In all formulations, a 3 or higher-value reward function is used. For more complex formulations than the one solved in this chapter, this will

help in achieving faster convergence and well-defined solution set. The reward function thus, should be formulated such that all subtle characteristics of the system are taken care of.

4.10 Conclusion

Q-learning algorithm, using both - random and Boltzmann – exploration converged to the same optimal solution, which is a subset of the actual set of the optimal solution obtained by the DP algorithm. Random exploration shows better exploration than Boltzmann exploration. Boltzmann exploration in general produced larger Q-values than random exploration. The actual difference is not very large and hence, in all multi-agent formulations, we will use random exploration to accelerate the convergence. In next chapter we will use this Q-learning algorithm to derive a formulation of multi-agent system. This formulation is independent of the dimension of the system and also independent of the number of agents in the system.

5 MULTI-AGENT HERDING PROBLEM

5.1 Introduction

This chapter treats multi-agent formulation of the herding problem, using local Q-learning for all active agents (pursuers). The problem contains one passive agent (the evader) and a number of active agents (pursuers). Active agents are non-cooperative agents and do not communicate with each other, and do not share any information about the state of the system. For a given active agent, all other agents (passive as well as active) form a part of the system.

5.2 Dimensionality issues in multi-agent problems

As seen in chapter 3, as either the number of agents in the system or the size of the grid is increased, total number of states required to describe the system increases exponentially. In the formulation in this chapter, we use 3 active agents (pursuers) and one passive agent (evader), thus total of four agents on a 7 x 7 grid.

5.2.1 Number of states required

For a problem with 4 agents and a grid size of 7 x 7, to include positions of all agents in the state information will require $7^8 = 5.764801 \times 10^6$ state. This makes lookup table representation of the state information impossible for practical reasons. Also, learning algorithm will require a very large number of epochs to converge. This makes use of such state information impractical.

5.2.2 Non-universality of knowledge

Q-learning algorithm was used on a 5 x 5 herding problem with one pursuer and one evader, with state information similar to that used in the last chapter. The optimal solution set obtained cannot be used directly on 3 x 3 problem to obtain the optimal solution. This happens because of the boundary conditions that apply on the outer boundaries of the grid. The solution set obtained in 5 x 5 can be used to get the optimal solution if and only if evader position is between the pen and the pursuer position. Thus, if position of pursuer is (x, y) then the knowledge obtained in the 5 x 5 learning can be used as is, if position of evader is between (0, 0) and (x-1, y-1). Following figure shows this condition. For any other condition, we will get

a solution, but it will not be the optimal solution. This is true whenever learning is done in $m \times m$ grid and applied to $n \times n$ grid problem, where $m > n$.

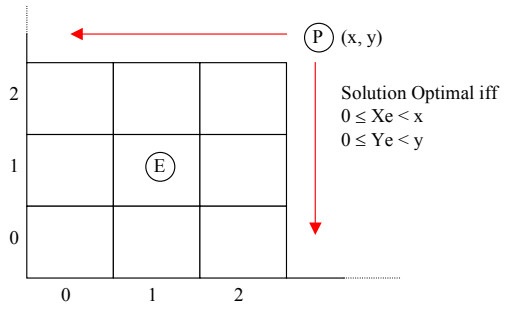


Figure 5.1 Optimality condition

5.3 Dimension-Independent Q-learning formulation

A formulation is designed which is independent of the size of the grid and the number of agents on the grid. This formulation inherently lacks significant information that is necessary to completely describe the system, and hence solutions obtained are not optimal. There is a trade-off between optimality and computational requirements of the problem.

5.3.1 Local domain for active agent (pursuer) – State and Control Set

Every active agent (pursuer) can sense (gather information about) only a small area around it and not about the whole grid. The state thus sensed contains information only in the local domain of that agent. All other agents form a part of the system.

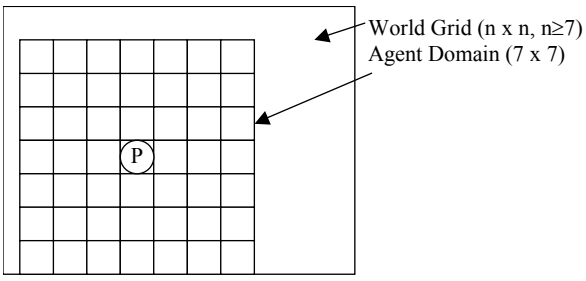


Figure 5.2 Local domain for active agent

Every active agent (pursuer) in the system has a local domain of size 7 x 7. The pursuer is at the center of this local domain. This is the only area that the pursuer can sense and collect information about. The information that the pursuer collects includes the positions of various agents in this local area. We could include positions of all other three pursuers and the evader,

but then the state-space contains $7^8 = 5764801$ states and then with 9 control actions for each state, number of Q-values becomes 51883209. It is important to note here that each pursuer now has its own state-space unlike in the 3 x 3 formulation, where the whole state space was describing the whole system. Thus, to limit the state-space size the state information is limited to the position of the evader in the local domain and the position of the pursuer that is nearest to the evader. Thus, the state-space size now is $7^4 = 2401$, and with 9 control actions, number of Q-values is 21609. Any state is thus defined by

$\mathbf{X(k) = \{x_p(k), y_p(k), x_e(k), y_e(k)\}}$, where,

X(k) = state of the system at any instant k,

$x_e(k)$, $y_e(k)$ = x and y-coordinates of the evader position at instant k,

$x_p(k)$, $y_p(k)$ = x and y-coordinates of the pursuer at instant k, nearest to the evader

All these coordinate values are local coordinates in the domain and are not actual coordinates of these agents in the grid. The pursuer at the center of the domain and has local coordinates (0, 0). All other coordinates are calculated with the position of the pursuer as the origin (0, 0).

5.3.2 Local Domain Partitioning

The local domain of the pursuer is partitioned in two parts – Visual and Influential – as shown in the following figure –

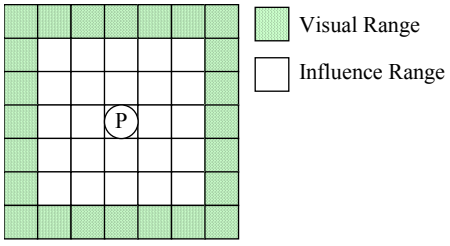


Figure 5.3 Local Domain Partitioning

If the evader is in the visual range of the pursuer, then pursuer can see the evader, but evader does not try to move away from the pursuer. If evader is in the influence range, then evader moves away from the pursuer. This is similar to equilibrium and non-equilibrium states in the 3 x 3 grid formulation.

The control set for pursuer is also the same as that in the case of 3 x 3 1 Pursuer, 1 Evader formulation. At each state pursuer can move in any one of the eight possible directions, or can stay at the current position.

5.3.3 Equilibrium States

Equilibrium and final equilibrium states are defined in the same way as in the section 3.5, with a small change. Whenever evader is not in the influence range of any of the pursuers, evader is in the equilibrium state. If evader is in equilibrium state and that its position is (0, 0), i.e. the evader is in the pen, then it is in the final equilibrium state. If evader is in the influence range of any of the pursuers, then it is not in equilibrium state and thus, has to move away from corresponding pursuer(s).

5.3.4 Evader Dynamics

Since now there are more active agents (pursuers) in the grid, evader dynamics is different from the 1 Pursuer – 1 Evader implementation.

1. Evader stays at its position if there it is not in the influence range of any pursuer.

2. Whenever there is a pursuer in any one of the adjacent cells, evader moves in a direction that takes it away from the pursuer, on the same line joining the positions of the pursuer and the evader. In this case evader takes a step of size two units. Mathematically,

$$Xe\_New = Xe + 2 \cdot (Xe - Xp),$$
$$Ye\_New = Ye + 2 \cdot (Ye - Yp)$$

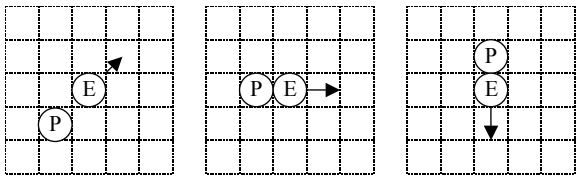


Figure 5.4 Evader dynamics – Influence area - Adjacent cells

3. Similar boundary conditions (at edges and corners) apply as in the case of 3 x 3 formulation, when the evader is in any of the adjacent cells to any of the pursuers.

4. When evader is in the influence are, but not in any of the adjacent cells with the pursuer, evader takes a step of size one unit. The directions for such a movement are shown in the following figure (each arrow shows the movement of evader from the cell where the arrow originates to the cell with the arrow head) –

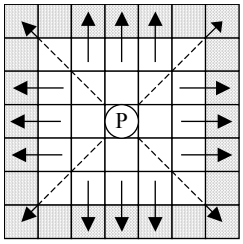


Figure 5.5 Evader dynamics – Influence area – Non-adjacent cells

Thus, mathematically above dynamics are written as –

$$
\begin{aligned}
&dx = X\_Ev - X\_Pur, \\
&dy = Y\_Ev - Y\_Pur. \\
&if \;\; abs(dx) > abs(dy) \\
&\quad X\_Ev\_New = X\_Ev + (dx/2) \\
&\quad Y\_Ev\_New = Y\_Ev \\
&if \;\; abs(dy) > abs(dx) \\
&\quad X\_Ev\_New = X\_Ev \\
&\quad Y\_Ev\_New = Y\_Ev + (dy/2)
\end{aligned}
$$

5. When there are two or more pursuers in adjacent cells of the evader position, direction of evader movement is calculated using vector mathematics. Each pursuer has its influence in a certain direction, with corresponding x and y components. Each component can have either a value 0, ±1, or ±2. The resultant vector obtained by vector addition of x and y components can also have a length of 0, ±1, or ±2 (this condition ensures that the evader can move according to the dynamics described above, at any time instant). Following figure shows vector fields for the cases shown in figure 5.3 –

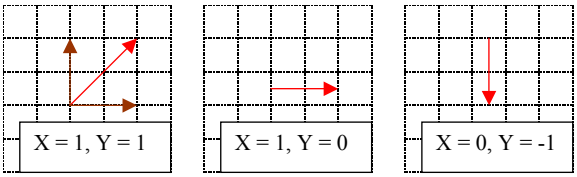


Figure 5.6 Vector field of influence

Thus, the x and y components for the resultant movement direction of the evader are given by –

$$\partial x = sign\left(\sum_i dx_i\right) \cdot \min\left[abs\left(\sum_i dx_i\right), 2\right],$$

$$\partial y = sign\left(\sum_i dy_i\right) \cdot \min\left[abs\left(\sum_i dy_i\right), 2\right]$$

where, i is the number of pursuers adjacent to the evader,

dx_i and dy_i are the x and y vector components due to individual pursuers

5.4 RL Individual and Batch processing

When we have multiple active agents that are using RL to update their knowledge about the system, we can update Q-values for these active agents in either individual processing or in batch processing. In individual processing RL update is carried out for an agent when that agent senses the world and takes an action. In case of batch processing, RL update is carried out for all agents after all agents have taken their corrosponding actions.

5.4.1 Individual RL Processing of Q-values

There are n active agents for whom Q-values are updated in the RL simulation. In individual processing, one of these n agents is randomly selected. It then senses the world and forms the state information that is relevant to itself. It then finds the best action for this state. RL algorithm then assesses this action and accordingly reinforces the Q-value for the latest state-action pair. Then another agent from the remaining $n-1$ active agents is selected randomly and RL algorithm is then executed on this agent. This process continues until all agents have undergone through the RL process. The evader is moved only after all pursuers have completed their moves. When the evader moves, RL update process is again executed on all agents for the latest state and HERE action (since only evader has moved).

In this paradigm, all other active agents form a passive part for the active agent under consideration. The reward function thus is isolated for this agent only and thus no collective reinforcement is possible.

5.4.2 Batch RL Processing

In this paradigm, all pursuers sense the world and form their corrosponding state informations at the same time. With this state information, these pursuers then find the best actions for their respective states and then they all move at the same time, unlike one at a time in case of individual processing. As all pursuers move, RL function reinforces each pursuer depending on the result of the actions of all pursuers. Thus the reinforcement reward function takes into account the actions taken by all the pursuers collectively and not actions taken by just one agent at a time.

Thus, it is expected that the batch processing will provide a better solution than the one found in the individual processing. The important design criterion in this case is that the reward function design should be such that it takes a global perspective and thus contains code that ensures that the reinforcement looks at the state of the world not in terms of individual agent state information.

The reward function for batch processing should be fundamentally different from the reward function used in individual processing. This issue is discussed in more detail in next section where actual simulation details are discussed.

5.5 Q-learning simulation – Individual RL Processing

5.5.1 Reward functions

In multi-agent Q-learning a more complex reward function is designed. It is similar to the 3-value reward function used in section 4.8. Only addition is made when the pursuer sees no evader but sees a pursuer in its local domain. If there are more than one pursuer in the local domain and no evader, then the pursuer that is nearest to the pursuer for which state is determined, is selected in the state information. In such a state, when pursuer sees no evader, it is expected that pursuers should spread out in the grid so as to catch the evader. Thus, the reward function makes sure that when no evader is seen, pursuers spread out. Following is the pseudo-code for the reward function.

X_Cur, X_New, Y_Cur, Y_New = local coordinates of pursuer of interest

Xp, Yp = local coordinates of another pursuer in the domain

Xe, Ye = local coordinates of the evader in the domain

Start Reward Funtion

Reward = 0;

IF State Info contains evader position in local domain

Cur_Dist = max[abs(Xe – X_Cur), abs(Ye – Y_Cur)];

New_Dist = max[abs(Xe – X_New), abs(Ye – Y_New)];

IF New_Dist < Cur_Dist reward := reward + 1

ELSE IF New_Dist = Cur_Dist reward := reward + 0

ELSE IF New_Dist > Cur_Dist reward := reward - 1

IF Xe = 0 AND Ye = 0

IF abs(Xe – X_New)>1 OR abs(Ye – Y_New)>1 reward := reward + 5

END IF

ELSE IF State Info contains no evader position in local domain

Cur_Dist = max[abs(Xp – X_Cur), abs(Yp – Y_Cur)];

New_Dist = max[abs(Xp – X_New), abs(Yp – Y_New)];

IF New_Dist < Cur_Dist reward := reward - 2

ELSE IF New_Dist = Cur_Dist reward := reward -1

ELSE IF New_Dist > Cur_Dist reward := reward + 2

END IF

End Reward Function

5.5.2 Learning parameters

$\#\,Active\ Agents\ (Pursuers) = 3$

$\#\,Passive\ Agents\ (Evaders) = 1$

$Grid\ Size = 7x7$

$\gamma = 0.9$

$\alpha = 1, with\ decay\ rate\ for\ \alpha = 10^{\log 0.001 / 10000} = 0.999769768,$

α *is uniformly used for all active agents.*

5.5.3 Flowchart for Individual RL Simulation

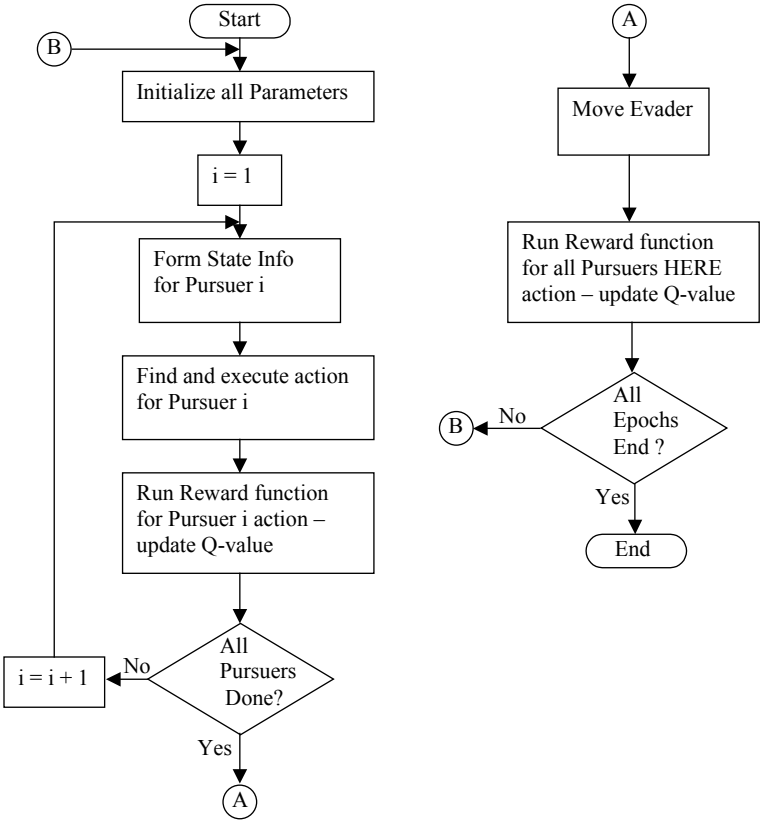


Figure 5.7 Flowchart for Q-learning simulation with local reinforcement

5.5.4 Simulation results

Q-learning alogorithm was executed for 3 pursuers and 1 evader for 30,000 epochs. Results are discussed for a representative state of the world which is shown in the following figure.

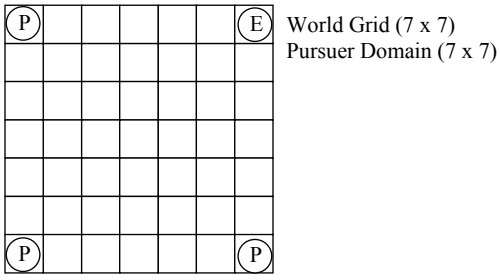


Figure 5.8 Starting State

Following figure shows the number of steps required to complete the epoch from the above state.

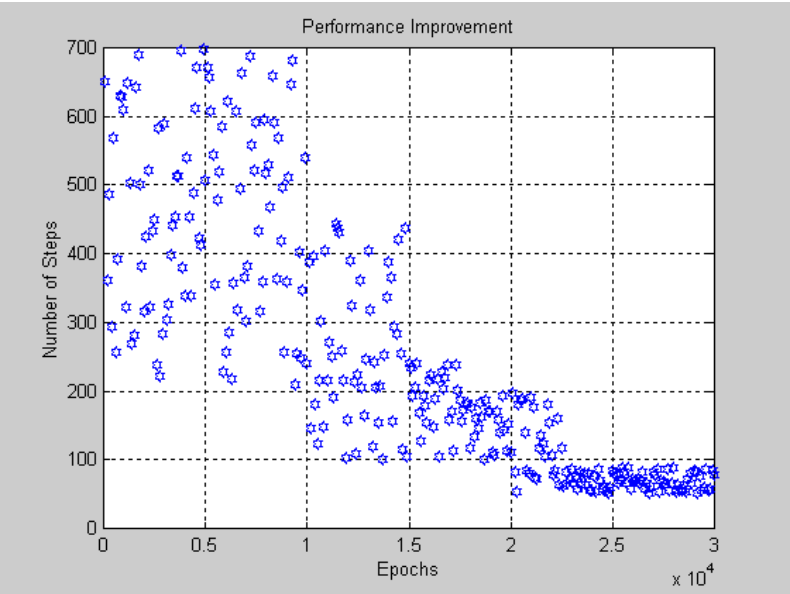


Figure 5.9 Performance improvement with epochs

Number of steps required to complete one epoch for the state shown in fig 5.5 is found by using the Q-tables generated by all the three pursuers after various epochs (100, 200, …, 30000) and initializing the system in that state. As the learning increases (i.e. actual number of epochs run by the algorithm on random initial states) the number of steps required to complete one epoch for the above state decreases. This is as per the expectation that as learning epochs increase, performance improves. Following figure shown that after 20,000 epochs, the number of steps required for the above mentioned starting state lies within a certain band, which is called as the *convergence band*.

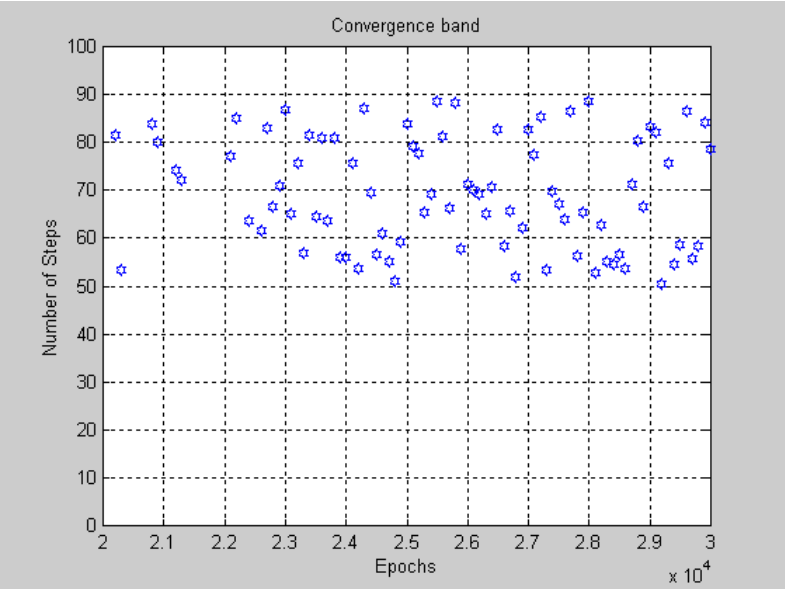


Figure 5.10 Convergence band

One important thing to note is one step mentioned above consists of moves by all agents. That is, when all pursuers and the evader move, it is considered to be one step. Thus, *steps x 4* give the total number of agent moves in an epoch.

5.5.5 Convergence band

As seen in the above figure, Q-learning algorithm (in terms of number of steps required to complete an epoch) does not converge to a single value unlike in the case of 3 x 3 formulation. Instead it converges into a band, called the *convergence band*. The most important reason for this is the local learning of the active agents – pursuers.

As all pursuers can sense only a small area around them and not all the grid, the information they collect is limited. The decisions are based on this limited knowledge of the system. Also there is no communication of information between the pursuers.

Thus, there are many states in which a pursuer has to take a random action rather than choosing the best action using the Q-learning. These states are referred to as *Blank States* henceforth. Such blank states include states when pursuer sees no evader and no pursuer in the local domain. In this situation, pursuer has to act randomly. This randomness is inherent to this formulation and it remains intact even after going through the learning algorithm

simulation. Hence, more the number of blank states in epoch, more the number of random choices made and less effective the learning algorithm.

For the state shown in fig 5.5, all pursuers are in a blank state and hence they all start by choosing their actions randomly until they see either a pursuer or an evader in their local domains. Until then it is impossible to predict the number of steps required so that at least one of the pursuers comes out of the blank state. Once any pursuer comes out of the blank state, it can pursue the optimal path unless it again goes into a blank state because of actions of other pursuers. This randomness and lack of coordination between pursuers result in a convergence band.

Performance improvement for a non-blank starting state is shown below. Figure 5.8 shows the starting state and figure 5.9 shows the performance of the agents. Note that the same Q-matrices as used in the blank starting state are used to test for non-blank starting state. These Q-matrices were obtained from the same simulation run.

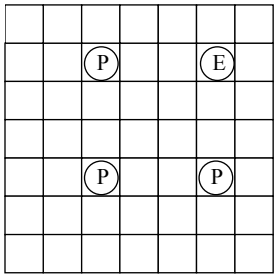
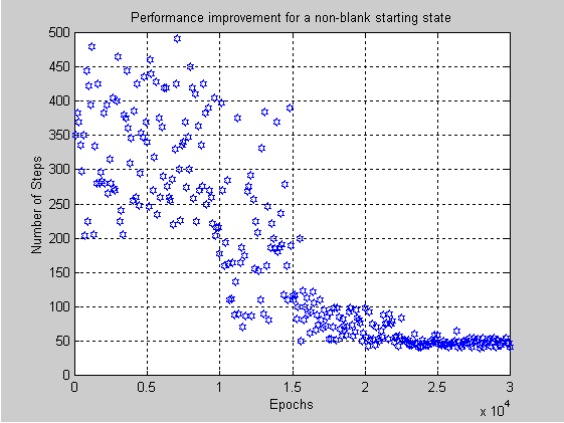


Figure 5.11 Non-blank starting state



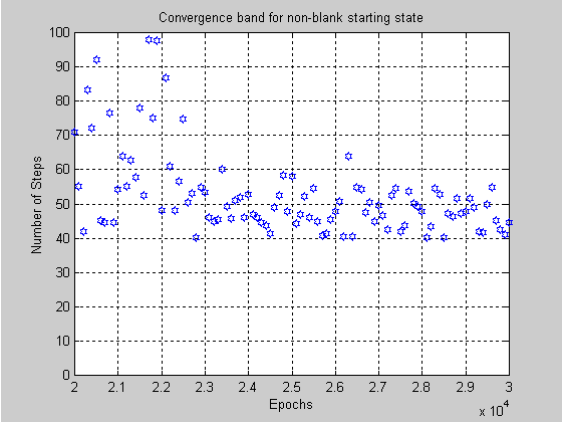


Figure 5.12 Q-Learning Convergence for a non-blank starting state (Convergence)

5.5.6 Effect of Blank States on the performance

Larger the number of blank states in an epoch more is the number of steps required for the epoch to complete. Following figure shows the percentage of non-blank states in epochs for the states shown in fig 5.5 and fig 5.8 respectively.

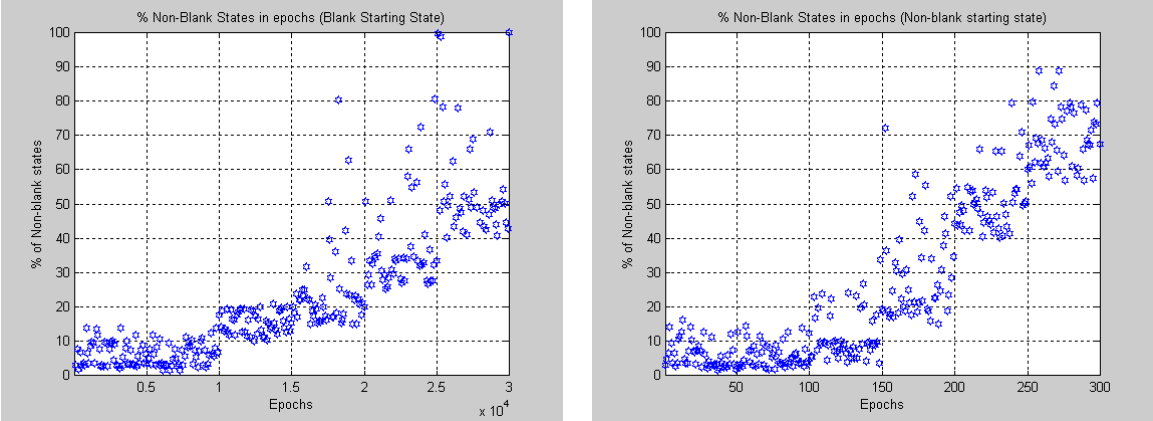


Figure 5.13 Percentage of Non-Blank states in epochs

Following table lists the percentages of non-blank states after completing learning through a number of epochs.

Percentage values (%)	Blank Starting State	Non-Blank Starting State
Overall Percentage	22.9897	31.4071
Epochs (1 – 5000)	6.1332	5.8947
Epochs (5001 – 10000)	6.2630	5.9409
Epochs (10001 – 15000)	15.3558	11.7098
Epochs (15001 – 20000)	24.1916	32.9393
Epochs (20001 – 25000)	39.4055	57.0891
Epochs (25001-30000)	70.2950	74.6143

Table 5.1 Non-Blank State percentages in simulation

The average of percentage of non-blank states for the blank starting state is 22.9897% over the simulation run of 30,000 epochs. In the convergence band this percentage value averages 70.295%. These corresponding values for the non-blank starting state are 31.4071% and 74.6143 %. This explains that the convergence band for the blank starting state is wider that the convergence band for the non-blank starting state.

5.5.7 Reducing the number of blank states

It is important to reduce the number of blank states to a great extent to improve the performance of the agents. Ways could include increasing the size of the local domain (which will increase the computational load), sharing information about the world using communication. Main consideration is to keep the additional computational requirements overhead to a minimum.

In the next section, batch-processing formulation is presented and in next two chapters a technique based on Artificial Immune Systems is presented. Batch processing is used to speed up the convergence and AIS implementation is used to decrease the number of randomness in the decision making, using the AIS modeling of the system.

5.6 Q-learning simulation – Batch RL Processing

In this formulation, in addition to the reward function used in the section 5.5 formulation, a global reward function is used. This function reinforces actions of all agents (pursuers) after each of them has chosen and executed an action, and after evader has moved.

5.6.1 Global Reinforcement Reward Function

After all active agents (pursuers) and passive agents have completed their respective actions, global reward function assesses the overall effect of all these actions. It compares the desirability of the old overall system state (not the state that a pursuer agent sees in its local domain) to the desirability of the new system state. If new state is rated better than the old state, a reward is given to all active agents (pursuers); otherwise a penalty is given.

This reward function is a three-valued reward function and it decides the desirability of system state on the change in the distance of the evader to the pen. If the distance of the evader decreases in the new state, then a reward of +1 is given, if the distance remains same from old state to the new state a reward of 0 is given, otherwise a penalty of –1 is given.

Pseudo-code for the global reward function is as follows –

Start GLOBAL Reward Funtion

Reward = 0;

Cur_Dist = max[Xevader, Yevader];

New_Dist = max[Xevader_new, Yevader_new];

IF New_Dist < Cur_Dist reward := reward + 1

ELSE IF New_Dist = Cur_Dist reward := reward + 0

ELSE IF New_Dist > Cur_Dist reward := reward - 1

End GLOBAL Reward Function

5.6.2 Flowchart for Batch RL Simulation

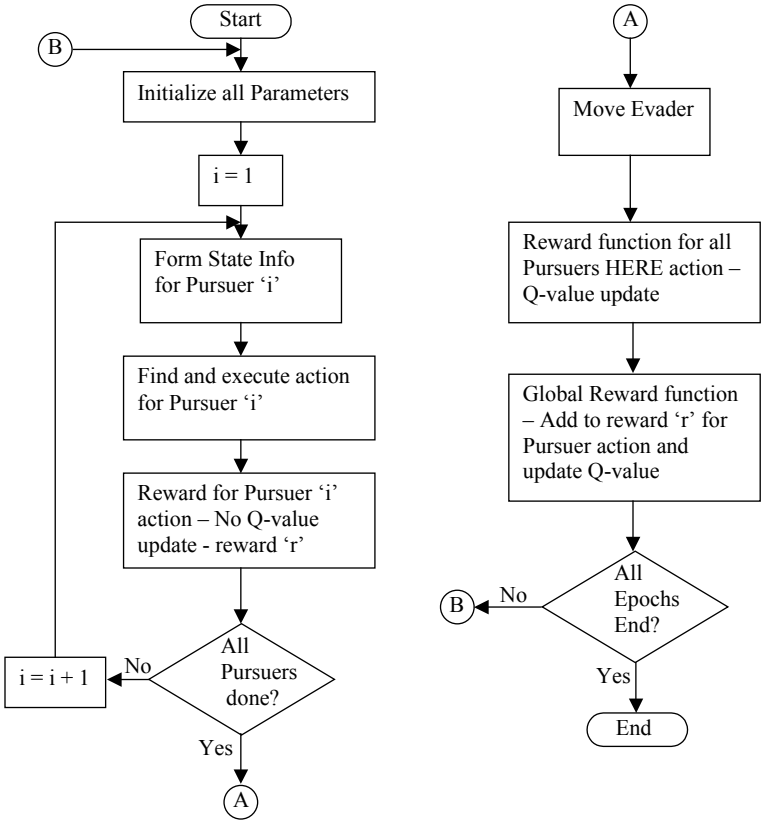


Figure 5.14 Flowchart for Q-learning using Global Reinforcement

5.6.3 Simulation results

Q-learning simulation with global reinforcement was run for 30,000 epochs. The only difference between this simulation and the simulation with only local reinforcement is the addition of the global reward function, with all other parameters being exactly same. Following figures show the results for the starting state shown in fig 5.8.

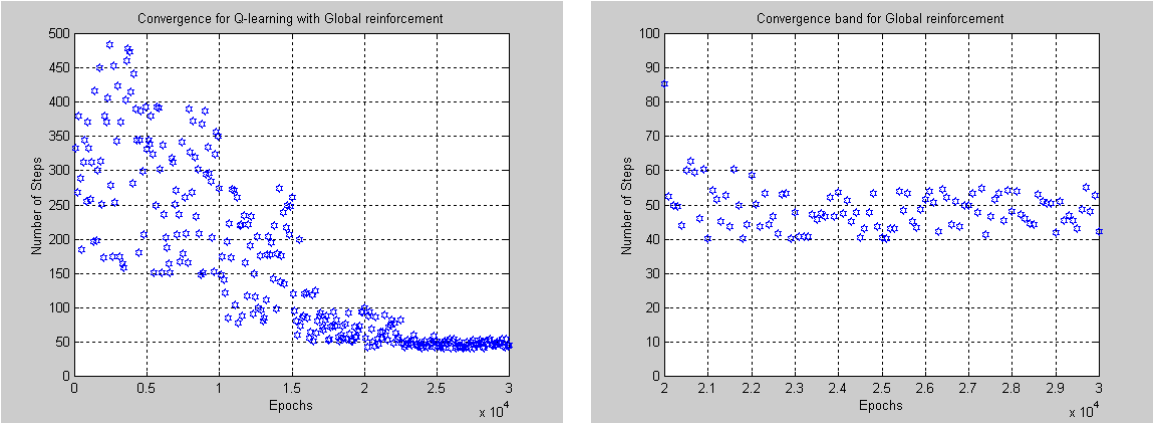


Figure 5.15 Convergence for Q-learning with Global reinforcement

As can be seen from fig 5.13 and fig 5.10, there is not much improvement in the convergence speed of the Q-learning simulation, although there is very marginal improvement. This can be a result of very simplistic nature of the global reinforcement function used. This function did not make sure that number of blank states is reduced.

5.7 Conclusions

In this chapter, two formulations of the Q-learning algorithm for solving the multi-agent herding problem are presented. All active agents do learn strategies to herd the evader to the pen, but these actions and strategies are not optimal. This owes to the fact that the local information that is available to each active agent hides a large part of the system from the agents. Also other agents form a static part in any local state formulation for active agent and thus there is no coordination between actions taken by various agents. This may lead to constructive as well as destructive agent formations, without agents being aware for this. That is, actions of one agent may herd the evader towards the pen, but those of others may move the evader away from the pen. Thus, it is important to have some coordination, knowledge sharing between these agents to increase the effectiveness of their actions.

To achieve a faster convergence of Q-learning simulation, complex global reward function can be written that take into account actions taken by all agents to determine the reward for each agent. We may also increase the state-space by including more information in the agent state information, such as the actions taken by other agents. But owing to a large state-space for each active agent (pursuer), which consists of 51883209 Q-values for each agent in this formulation, it is difficult to expand state space due to computational constraints.

In next two chapters, another approach based on Artificial Immune System is presented. Human immune system is modeled so as to describe certain characteristics of the herding problem. This paradigm can be used to completely solve the herding problem without using Q-learning. But instead, it is used only to achieve a coordination between actions of various agents.

6 ARTIFICIAL IMMUNE SYSTEM - INTRODUCTION

6.1 Introduction

Immune system is a complex of cells, molecules and organs that together form an identification mechanism capable of identifying foreign antigens invading the body and destroying them. This system is also capable of differentiating between self and non-self, that is between own cells and foreign antigens. The immune system learns through evolution to differentiate between self and non-self. Immune systems exhibit many interesting properties that can be used in engineering applications. Some of these properties are distributed processing, reinforcement learning, self-non-self discrimination, and noise tolerance.

6.2 Overview of the Immune System

There are two types of immunities – innate immunity and adaptive or acquired immunity.

6.2.1 Innate Immunity

This is the natural immunity of the body and is capable of detecting and destroying a large number of antigens. It is also capable of self and non-self discrimination. This is the first line of defense against invading antigens and it uses non-specific immune response to attack antigens. Some of the cells that take part in this immune response are monocytes, macrophages and neutrophils. They are called as phagocytes and are responsible for eliminating antigens. Other type of cells are called as lymphocytes and they are of two types – B-cells and T-cells. T-cells develop in bone marrow and travel to thymus to mature while B-cells develop and mature within bone marrow. There are organs called lymphoid organs that provide sites where lymphocytes become genetically committed towards certain types of antigens. Other types of cells are called as natural killer cells that use non-specific response towards antigens. The most important aspect of innate immunity is that it induces co-stimulatory signals in Antigen Presenting Cells that lead to T-cell activation, promoting the start of the adaptive immune response.

6.2.2 Adaptive or Acquired Immunity

Adaptive or acquired immune system uses somatically generated antigen receptors, which are clonally distributed in B-cells and T-cells. Random processes of genetic mutation

generate these receptors and hence the design of the adaptive immune response depends on clonal selection of lymphocytes. Adaptive immunity uses specific immune response towards antigens. Adaptive immune response also has immunological memory that contains information about the antigens that had earlier invaded the body.

6.2.3 Antigen and antigen structure

Antigen is a foreign living molecule that triggers an immune response. Each antigen has a specific molecular structure, called as epitope.

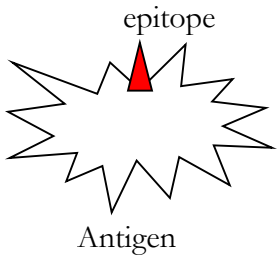


Figure 6.1 Antigen structure, epitope

B-cells and T-cells recognize antigens by checking the epitope structure. Any B-cell/antibody that has a complementary structure - called paratope - that locks with the epitope is said to recognize the antigen. This antibody then eliminates the antigen.

Figure 6.2 Antigen Recognition Mechanism

6.3 Immune cells

6.3.1 Lymphocytes

There are different types of lymphocytes – B-cells, T-cells, and Natural killer cells. B and T-cells have paratopes on their surfaces that are highly specific towards antigens.

Main functions of B-cells include production and secretion of antibodies as a response to antigens. Each B-cell produces a specific antibody. These antibodies are specific proteins that recognize and bind to a specific type of protein. This is the way by which cells are signaled to kill, ingest or remove the bound substance.

Functions of T-cells include the regulation of actions of other cells and they directly attack the infected cell. T-cells are divided into 3 categories – helper T-cells, Killer T-cells and Suppressor T-cells.

Helper T-cells activate B-cells and Natural killer cells. Killer T-cells directly invade antigens. Suppressor T-cells inhibit actions of other B-cells and thus maintain the immune response within certain limits, which otherwise results in allergies and autoimmune response. Natural killer cells directly attack antigens without first identifying them as is the case with B and T-cells. These cells also contribute to immune response regulation.

6.3.2 Phagocytes (cell eaters)

Phagocytes are white blood cells capable of ingesting and digesting antigens. Some phagocytes also have ability to present antigens to lymphocytes, and are then called as antigen presenting cells (APCs).

Important phagocytes are monocytes and macrophages. Monocytes circulate through blood and migrate to tissues where they change to macrophages. Macrophages perform a number of functions. They present antigens to T-cells after ingesting and digesting them.

6.3.3 The Complement System

The complement system constitutes a complex formed by plasma proteins that complement the antibody functions. When complement system detects an antigen, it generates a proteins complex that binds on the outer surface of the antigen, which facilitates the operation of phagocytes.

6.4 Immune System Operation

Following is the chain of actions that takes place when an antigen invades the immune system.

1. Antigens presenting cells such a macrophages ingest and digest the antigen. They then fragment the antigen into antigenic peptides.

2. Pieces of these peptides are joined to major histocompatibility complex (MHC) molecules and are presented on the cell surface.
3. T-cells have receptor molecules on their surfaces that recognize different antigens (peptide-MHC molecule combinations). When T-cells are activated by this recognition, they divide and secrete chemicals that activate B-cells.
4. B-cells also have receptor molecules on their surfaces and can recognize antigens without MHC complex. When B-cells are activated, they divide into two types. One type is called plasma cells that secrete antibody proteins. These proteins are soluble forms of the B-cell receptor molecules and they bind to the particular antigen and destroy it. B-cells of the other type (and also some T-cells) become memory cells that remain in the immune system forever. This boosts immune system's ability to eliminate same antigen that had been experienced before. This mechanism is the basis for immunization technique.
5. Antibody genes undergo random genetic mutations that result in improved response of the immune system after repeated immunizations. This is called as affinity maturation.

6.5 Clonal Selection

The principal underlying Clonal selection is that the cells that recognize an antigen proliferate while other cells are either eliminated or edited by a genetic mutation process. B-cells and T-cells both undergo Clonal selection.

When B-cells recognize a particular antigen, each B-cell secretes only one type of antibody, specific to that antigen. Also these B-cells proliferate and mature into plasma cells.

A large number of B-cells are produced in the bone marrow. Each of these B-cells has a specific receptor and can secrete a certain antibody. When an antigen invades body, only those B-cells that recognize this antigen mature into plasma and memory cells. Other B-cells that cannot recognize the antigen undergo genetic mutation by which their receptors are modified.

For this system to work, the receptor population (B-cells with various types of receptors) must be diverse enough to recognize any foreign antibody shape.

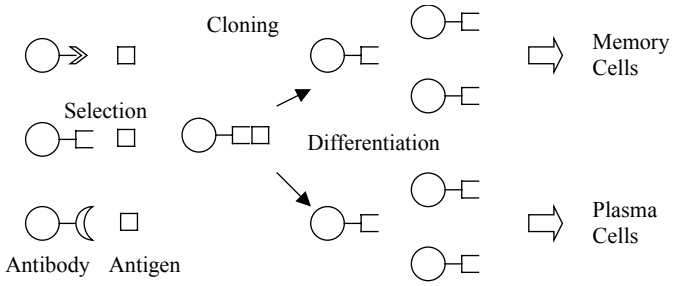


Figure 6.3 Clonal Selection Mechanism

6.6 Immune Network Theory

Jerne first introduced immune theory in 1974. This theory models the immune response in terms of a network of antibodies that stimulate and suppress each other. Following are some of the terms used in this theory. Epitope is the antigenic determinant that a B-cell looks for, while paratope is the receptor on the B-cell that locks on to a specific epitope. All B-cells also contain structures similar to antigenic epitope, displayed by various regions of antibody molecules. They are called as idiotypes. An idiotope is a single idiotypic epitope. The immune system is considered to be an enormous and complex network of paratopes that recognize sets of epitopes, and of idiotopes that are recognized by sets of paratopes. Thus each element in the network can recognize and can be recognized [27].

When an antigen invades immune system, those antibodies whose paratopes can recognize the antigenic epitope bind to the antigen. These antibodies also have idiotopes, which are recognized by other antibodies. Thus those antibodies that recognize these idiotopes, bind to the antibody, thus suppressing it. Thus, the response of the network depends on how well an antigen is recognized and on the stimulation and suppression mechanism of the network. Following figure shows the network structure.

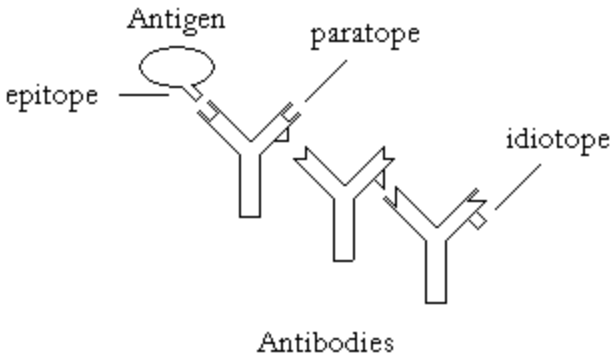


Figure 6.4 Idiotopic Network Model of Immune System

Antibodies that recognize either the antigen or other antibodies and lock on them, effectively suppress those antigens and antibodies. On the other hand, the antigens and antibodies that have idiotopic structures complimentary to paratopes of antibodies are said to stimulate those antibodies. The extent of idiotopic locking determines the mutual affinities (viz. antigen-antibody affinity and antibody-antibody mutual affinity).

6.7 Computational Aspects of the Immune System

Immune system has an inherent distributed nature and parallel processing capabilities. It also possesses powerful information processing capabilities. Following are some of the most important computational aspects of the immune system [26].

1. **Pattern Recognition** - Immune system can recognize and classify different patterns in epitopes and idiotopes to generate selective responses. Self - non-self determination is achieved by this pattern recognition capability.
2. **Feature extraction** – Antigen presenting cells (APCs) digest the antigen and extract significant features from the antigen molecules to be presented to B and T-cells. Each APC serves as a filter that filters molecular noise and as a lens that focuses on salient molecular features [26].
3. **Diversity** – a diverse set of B-cells is required for antigen recognition process, which is done in parallel way at various sites throughout the immune system.
4. **Immune Memory** – immune system possesses a content-addressable memory, which contains genetic information about antigens experienced, in the form of matured memory cells.

5. **Reinforcement Learning** – B-cells that can recognize antigens proliferate and multiply while those that cannot identify the antigen are destroyed and undergo genetic editing. This is similar to reinforce desirable actions in reinforcement learning.

Some of the other computational aspects are noise immunity, distributed detection, self-regulation (through T-cells), threshold mechanism and probabilistic detection, adaptability, specificity.

All these features are requirements to design a learning system, especially in multi-agent systems. Many important applications of artificial immune systems are computer security, computer virus detection, UNIX process monitoring, autonomous robotic systems, pattern recognition anomaly detection and fault-tolerance.

In the next two chapters, an artificial immune system modeling is used to solve the multi-agent herding problem.

7 AIS MODELING FOR GROUP BEHAVIOR ARBITRATION

7.1 Introduction

AIS is suited for systems that contain a number of independent entities – such as active agents in herding problem – that need to interact with each other to achieve the goal of the system. In this chapter, AIS (artificial immune system) is designed to solve the herding problem, using the idiotopic network model of the immune system.

7.2 Definition of the AIS components in the herding problem

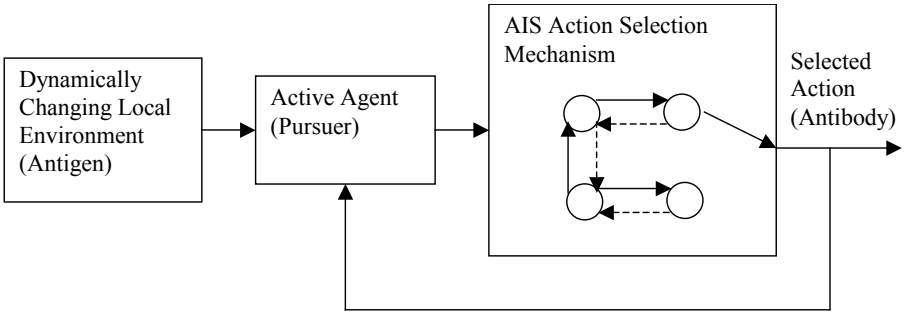


Figure 7.1 AIS System

Pursuers in the system sense the dynamic system and this state information is modeled as the active antigen(s) invading the pursuer. Depending on the definitions of antigens, there can be multiple antigens invading the pursuer at the same time. Once pursuer senses the world in its local domain, it runs through the AIS action selection mechanism (that uses idiotopic network AIS model) to select an action or a strategy from a collection of a number of possible actions/strategies. These actions are called as antibodies. Pursuer then executes the selected action and the cycle repeats.

7.3 Group behavior control in the herding problem

7.3.1 Introduction

In this section AIS for group behavior arbitration is designed. AIS is used to decide a strategy for all pursuers instead of deciding exact action (actions as were defined and used in Q-learning). Implementation of AIS involves following steps –

1. Environment detection,

2. Behavior strategy arbitration using AIS,

3. Strategy execution.

The objective of the system is to find strategies by interaction among active agents (pursuers) and then carry out corresponding tasks independently. If concentration of a selected strategy is above certain threshold, that strategy is broadcast to all other agents (pursuers) and thus this strategy forms the group behavior [33].

AIS model contains B-cells and T-cells. Each pursuer is modeled as a B-cell and T-cell concentration is modeled as a factor dependent on the inter-agent distance.

7.3.2 Antigens Definition for the Pursuer

Every antigen has a characteristic epitope, which is the unique signature of that antigen. This epitope is checked by B-cells to identify the antigen. Information sensed by a Pursuer in its local domain is translated into an epitope, which is a 2-tuple. The local domain of Pursuer is divided into two parts as shown in the figure below.

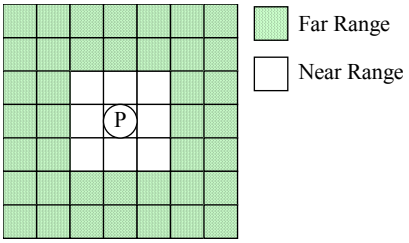


Figure 7.2 Local Domain of Pursuer (Antigen Definition)

Any antigen includes information about the type of any external agent (pursuer or evader) in the local domain, and the information about whether this external agent is in the far or the near area of the local domain.

Agent ID	Range

Figure 7.3 Antigen definition

Agent ID	P → Pursuer
	E → Evader
Range	F → Far
	N → Near

Table 7.1 Antigen values

In the following discussion, the pursuer for which antigens are referred will be called as "the pursuer" and all other agents (pursuers and evader) are referred to as "external agent(s)".

In a given system state, the pursuer will identify the type of agent in its local domain and accordingly select either P or E as the value for the agent id field in the antigen. If the external agent is in the near range the value chosen is N (Near). If the external agent is in the far range the value chosen is F (Far). When there is neither a pursuer nor an evader in the local domain then active antigen is defined as XX.

For each external agent present in the local domain of the pursuer, there will be a corresponding antigen that will be active on the pursuer. There is however a possibility that more than one agent of the same type may be present, for example two pursuers in the Far range. In that case, the antigen active for both these pursuers will be the same ("PF") and thus, even though there are more than one pursuer, only one antigen is active. Following table lists all the possible antigens in the system.

EN	EF
PN	PF
XX	

Table 7.2 Possible Antigens

7.3.3 Antibody Definitions

Four strategies are defined as the antibodies. These strategies are similar to those defined in [32] and [33]. Strategies are defined in a way appropriate to the herding problem. Each of these antibodies is a formation of pursuers in a particular way. These antibodies are listed below.

1. **AB1 - Herding -** strategy to find and go to a particular location in a formation that will result in herding the evader in the best possible way. Following figure shows the formation strategy for Herding antibody.

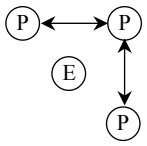


Figure 7.4 Herding Strategy Formation

2. **AB2 - Aggregation –** strategy to move in a particular area, keeping some maximum distance between them. A triangular formation as shown in the following figure is the desired result of the aggregation strategy. The maximum distance between any two pursuers in x and y-directions is 3 units so that they remain in the local domain of each other.

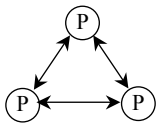


Figure 7.5 Aggregation Strategy Formation

3. **AB3 - Dispersion –** strategy to spread out in the grid, keeping some minimum distance between them. The minimum distance between any two pursuers in x and y-directions is 3 units so that they may remain in the local domain of each other, but is not guaranteed.
4. **AB4 - Random Search –** strategy to move randomly in the grid until there is at least one agent in the local domain. This is the basic strategy of pursuers.

7.3.4 AIS Dynamics

With every antibody associated is a variable, called concentration of that antibody. This variable represents the actual level of stimulation and suppression for that antibody in the network. Antibody with maximum concentration is selected, in a "winner-takes-all" fashion. For i^{th} antibody, stimulus and concentration are calculated by the following equation.

$$S_i(t) = S_i(t-1) + \left[\alpha \cdot \frac{\sum_{j=0}^{N-1} m_{ij} s_j(t-1)}{N} + \beta g_i - c_i(t-1) \right] \cdot s_i(t-1) \qquad (7.1)$$

$$s_i(t) = \frac{1}{1+\exp[0.5 - S_i(t)]} \qquad (7.2)$$

$$c_i(t) = \eta \cdot (1 - d_i) \cdot S_i(t) \qquad (7.3)$$

where, i, j = 0, 1, …, N-1, and α, β, η are constants,
N = number of antibody types,
$S_i(t)$ = Stimulus of antibody i,
$s_i(t)$ = Concentration of antibody i,
$s_j(t)$ = Concentration of antibody j for other pursuers,
$c_i(t)$ = Concentration of T-cell for antibody i, d_i is the distance between pursuers
m_{ij} = Mutual stimulus coefficient between antibody i and j,

g_i = affinity of antibody i towards antigen.

First term in the square bracket on the right hand side of equation (7.1) is the term that gives stimulation and suppression of the antibody by other agent's antibodies. The second term gives the stimulation of the antibody by all active antigens. Following figure shows this interaction that results in the first two terms in equation (7.1).

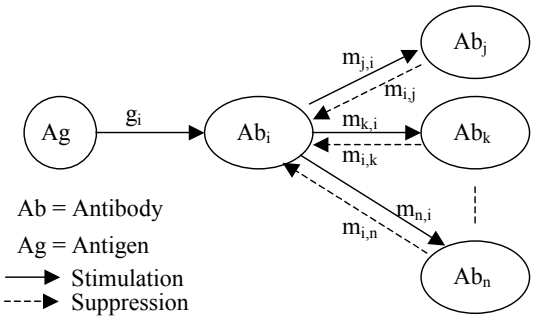


Figure 7.6 AIS Dynamics for one agent

The third term in equation (7.1) signifies the T-cell influence on the B-cell antibodies. Equation (7.2) gives the concentration of antibody from the antibody stimulus. Equation (7.3) gives the T-cell concentration for the B-cell (active agent pursuer).

7.3.5 T-cell modeling

T-cell modeling is added to control the extent of influence these agents have on each other. For every B-cell, there is a T-cell associated with it. The term $c_i(t)$ in the dynamics equation (7.3) gives the effect of T-cell concentration on the interaction between B-cells. Concentration of the T-cell is the distance between the two pursuers that are interacting. Thus, farther the pursuers, more is the T-cell concentration $c_i(t)$, and lesser is the effect of interactions. As the distance between pursuers decreases, T-cell concentration decreases and thus the interactions become stronger. Following table gives qualitative behavior of the T-cell modeling in the network.

Distance	**T-cell Concentration $c_i(t)$**	**Role of T-cell**
Small	Small	Helper T-cell
Medium	Medium	Helper T-cell
Large	Large	Suppressor T-cell

Table 7.3 Qualitative effects of T-cell modeling

7.3.6 Simulated Artificial Immune Network

Each pursuer is considered to be a B-cell with a number of antibodies available to it. Antigens are as defined in section 7.3.2. The distance between pursuers is considered to be the T-cell that controls the stimulation of antibodies. All antibodies of a B-cell are stimulated or suppressed by all antibodies of other B-cells. This can be graphically shown as follows. All 3 B-cells (Pursuers) are interconnected to each other in the network.

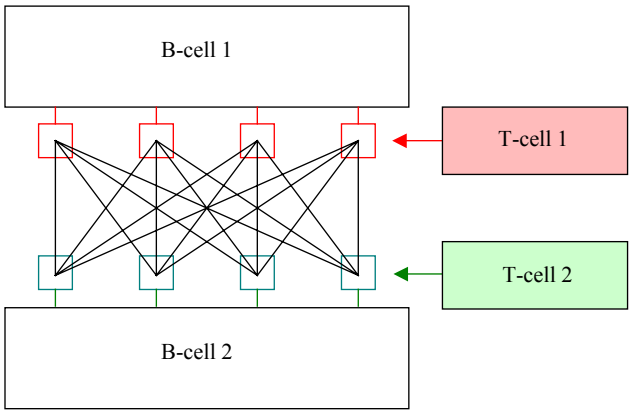


Figure 7.7 The Immune Network

Small squares connected to B-cells are the antibodies. The lines connecting antibodies of one B-cell to those of the other are bi-directional. They stimulate or suppress antibodies depending on the mutual affinity between these antibodies. T-cells affect concentrations of all antibodies of the corresponding B-cells.

As can be seen from the above network that concentrations of all antibodies are dependent not only on the current antigen but also on concentrations of antibodies in other B-cells. Thus the strategy (antibody) a pursuer (B-cell) selects is influenced by the strategies of other pursuers. This in turn results in a group behavior as decisions are taken not in isolation, but through the immune arbitration network.

7.4 Artificial Idiotopic Immune Network Simulation

Immune network simulation is done is two stages. In one stage the evader is stationary, i.e. it does not move. Evolution of group strategy is studied in this stage. In the next stage, herding problem is simulated as the immune network. Results in the first stage show how a group strategy is evolved through interaction. Results in the second stage show how these

interactions give better results to the herding problem, as compared to those obtained in Q-learning.

7.4.1 Immune Network Parameters

The important parameters used in the immune network are the mutual affinity coefficients between antibodies (m_{ij}) and the affinity coefficients between antigens and antibodies (g_i). Following tables give these affinity coefficients. They are decided by trial-and-error.

1. Mutual affinity coefficients between antibodies (m_{ij})–

	Herding	**Aggregation**	**Dispersion**	**Random**
Herding	1.0	-0.2	-0.2	-0.4
Aggregation	-0.2	1.0	-0.2	-0.4
Dispersion	-0.75	-0.75	1.0	-0.1
Random	-0.4	-0.2	-0.2	1.0

Table 7.4 Antibody Mutual affinity coefficients (m_{ij})

2. Affinity of antibody towards antigen (g_i) –

	Herding	**Aggregation**	**Dispersion**	**Random**
PN	0.2	0.2	0.6	0.0
PF	0.2	0.4	0.4	0.0
EN	0.7	0.2	0.1	0.0
EF	0.6	0.3	0.1	0.0
XX	0.1	0.1	0.4	0.4

Table 7.5 Antibody Affinities towards Antigens (g_i)

7.4.2 Group Strategy Evolution Simulation – All Stationary Agents

In this stage, simulation is started with agents at certain locations. All agents are stationary. Evolution of group strategy is studied in this stage, particularly the effect of interactions between agents. This simulation shows how a certain strategy is reinforced for a particular agent, depending on the mutual affinity of antibodies and antigens. Simulation is run for following starting state of the system.

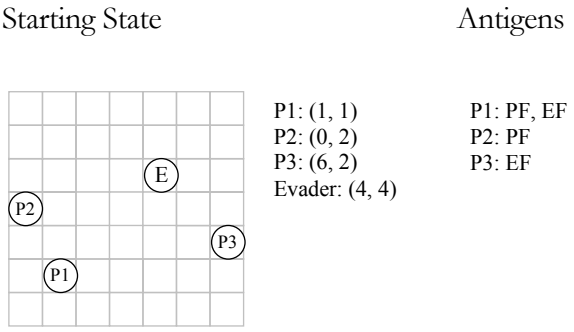


Figure 7.8 Starting state for static agents simulation

Following figure shows antibody concentrations for Pursuer 1.

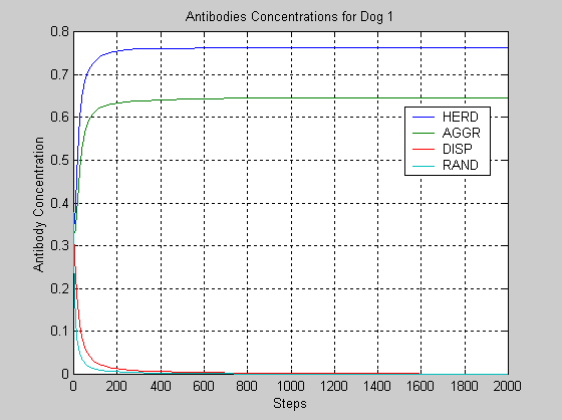


Figure 7.9 Antibody Concentrations for Pursuer 1 – Stationary agents

As seen from the mutual affinity tables, initially the two antigens active on Pursuer 1 (DF and DF) stimulate strategies to particular extent - herding (0.2 + 0.6), aggregation (0.4 + 0.3), and dispersion (0.4 + 0.1). We expect from the antigens invading Pursuer 1 that finally it should choose either Herding or Aggregation as these two are stimulated to a higher extent. The graph for concentrations shows that this is the result we get, with Herding being the most stimulated antibody (due to interactions between antigens and antibodies of other pursuers).

For Pursuer 2, the only antigen active is DF and we expect that it will choose either aggregation or dispersion depending on interactions with other pursuers. For Pursuer 3, only

antigen is SF and we expect it to choose Herding after interactions. Following figure shown results for Pursuer 2 and Pursuer 3.

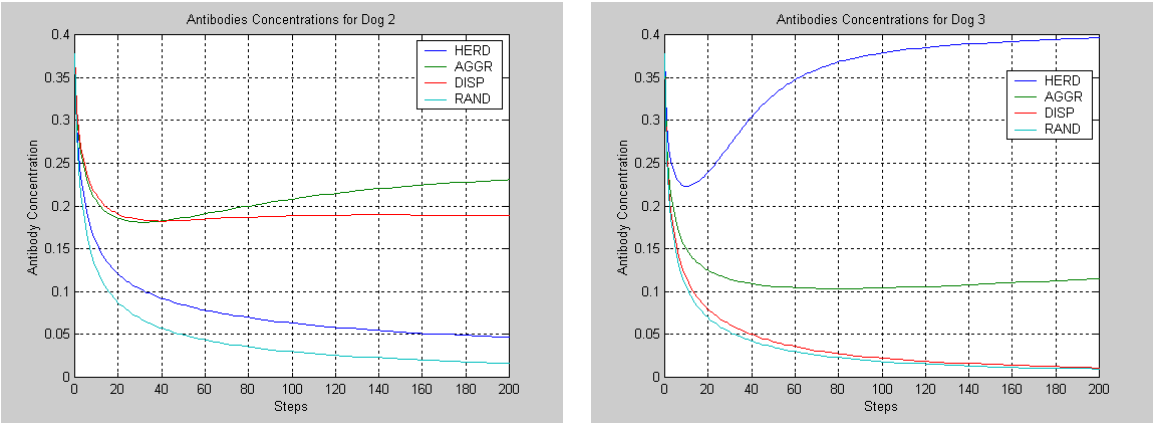


Figure 7.10 Antibody concentrations for Pursuer 2 and Pursuer 3 – Stationary agents

It can thus be safely concluded that the artificial immune network simulation can give results as per expectations and that the modeling of B-cell and T-cell is able to reinforce strategies through interactions within different B-cells.

7.4.3 Group Strategy Evolution Simulation – Stationary Evader

In this simulation, only evader is stationary while all pursuers move, depending on the strategy they choose. Starting state for this simulation is exactly the same as in the simulation with all stationary agents (fig. 7.8). Following figure shows the ending state of the system.

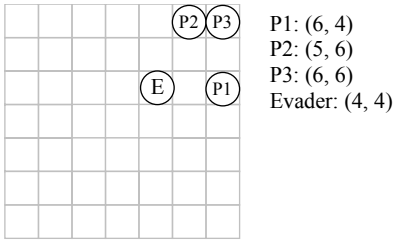


Figure 7.11 End State for static evader simulation

Above figure shows that at the end of the simulation, all pursuers were in the herding formation. This implies that the strategy selected by all agents after a certain number of steps was Herding. Following figure shows concentrations for all strategies for all three pursuers.

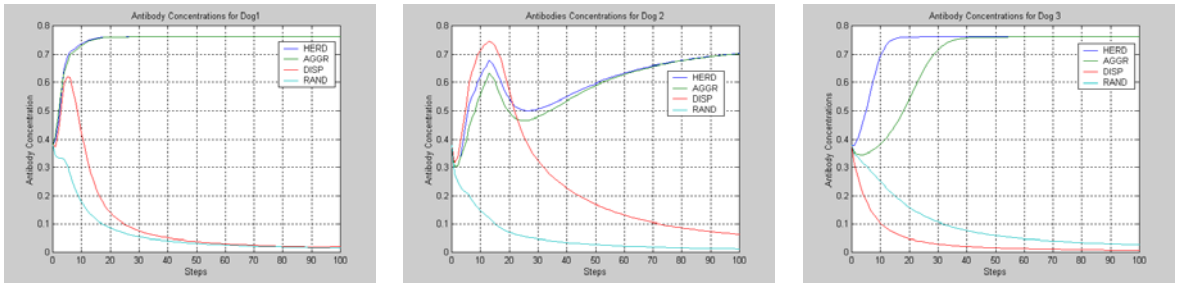


Figure 7.12 Antibody Concentrations for Pursuers – Static Evader Simulation

As we can see from above figure, concentrations of Herding and Aggregation antibodies eventually reach a maximum level, whereas concentrations of Dispersion and Random antibodies go to zero exponentially. Increase in concentrations of Herding and Aggregation antibodies for all pursuers can be explained similar to that in section 7.4.2 (simulation for all static agents). In this simulation important observations are regarding the concentrations of Dispersion antibody for all pursuers.

For Pursuer 1 and Pursuer 2 that have DF antigen active at the start, Dispersion antibody is stimulated and hence its concentration starts rising. At the same time, concentrations for Herding and Aggregation antibodies for Pursuer 1 and Pursuer 3 increase. As these two antibodies tend to suppress the Dispersion antibody, after a certain time, Dispersion antibodies for Pursuer 1 and Pursuer 2 are suppressed and hence their concentrations decrease. For Pursuer 3, concentration for Dispersion antibody decreases from start as it is not stimulated by any active antigen on Pursuer 3 and is suppressed by antibodies of other pursuers, concentrations of which are increasing monotonically.

These concentrations of antibodies for all pursuers together represent the knowledge gained by the system, for herding the evader to the pen. Unlike in Q-learning, this knowledge is stored in a very small memory, viz. concentrations of all antibodies in all pursuers.

In the next section, artificial immune network is used to completely solve the herding problem. No agents – either pursuers or the evader – are stationary. Pursuers move according to the strategy selected while evader follows the evader dynamics, which is same as in the multi-agent Q-learning simulation.

7.4.4 Group Strategy Evolution Simulation – Herding

In this section, herding problem is solved for 10 epochs. Starting state for all epochs is the same, and is as shown in the fig. 7.13. Same starting state was selected to study the performance of the adaptation of the immune network. In the next section, simulation with random starting states is discussed.

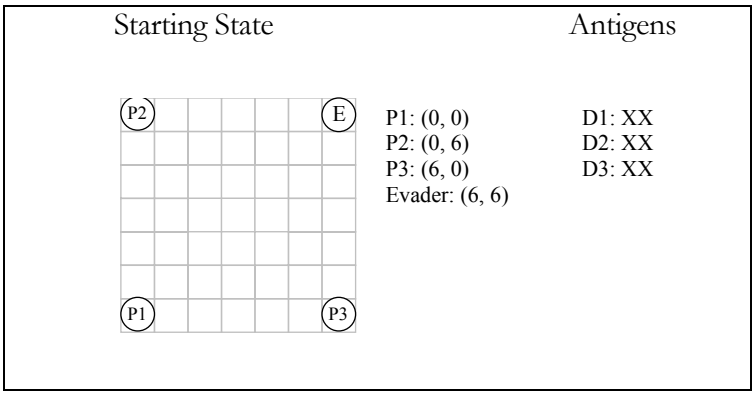


Figure 7.13 Starting state for herding simulation

Following figures show some of the ending states of the system and steps per epoch.

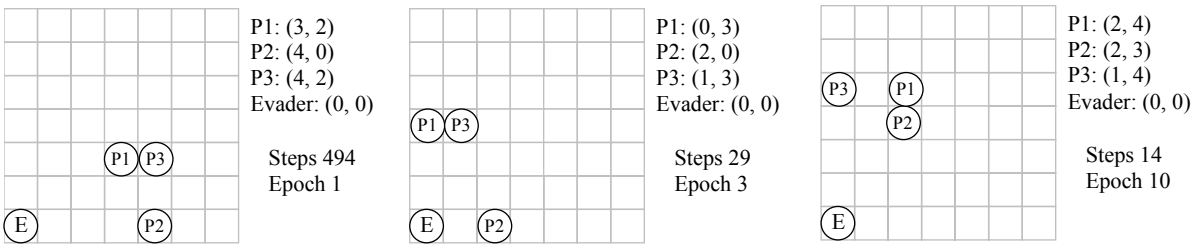


Figure 7.14 End States for herding simulation

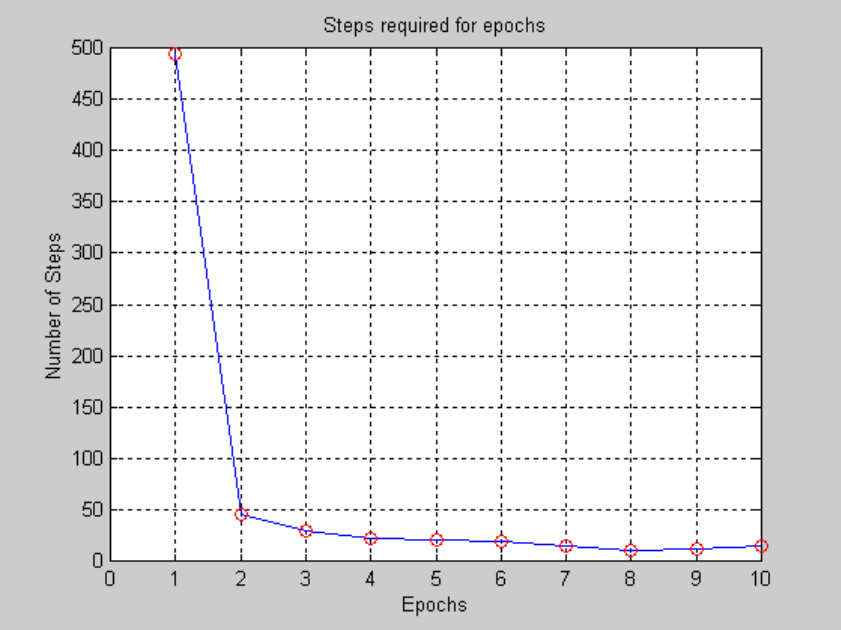


Figure 7.15 Steps per epoch

Following figures show the antibody concentrations for all pursuers.

Pursuer 1 Pursuer 2 Pursuer 3

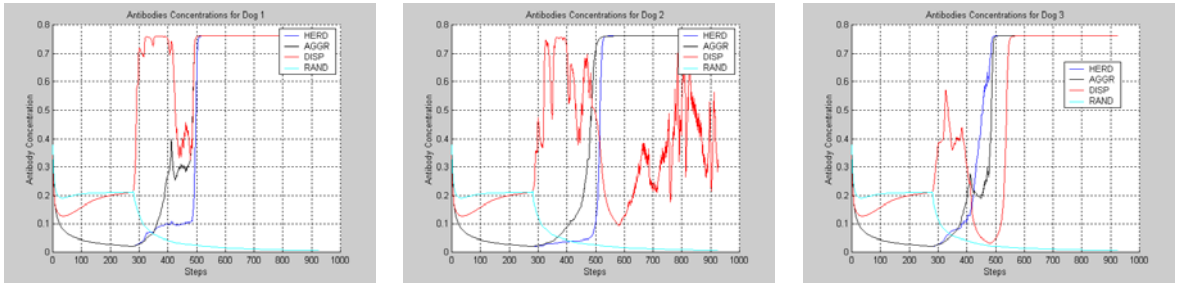


Figure 7.16 Antibody concentration - herding simulation

Above figures show that the arbitration mechanism as simulated using the artificial immune network adapts itself very fast as compared to Q-learning. Only the first epoch took a large number of steps (494), and then the number of steps reduced drastically. Numbers of steps after epoch 5 are within a small band of 10 to 15 steps, which indicates reduced randomness in the system.

Another important observation is that in no epoch the system was forced into a state of no escape, such as a state in which evader is trapped by all agents in a corner other than the pen (0, 0).

In the following section, immune network is trained for a number of epochs that start in random starting states and then it is tested on the starting state used in this section.

7.4.5 Group Strategy Evolution Simulation – Herding

AIS network is trained for 100 epochs, which have random starting states, and then is tested on the starting state as shown in fig. 7.13. This testing is done for 15 epochs.

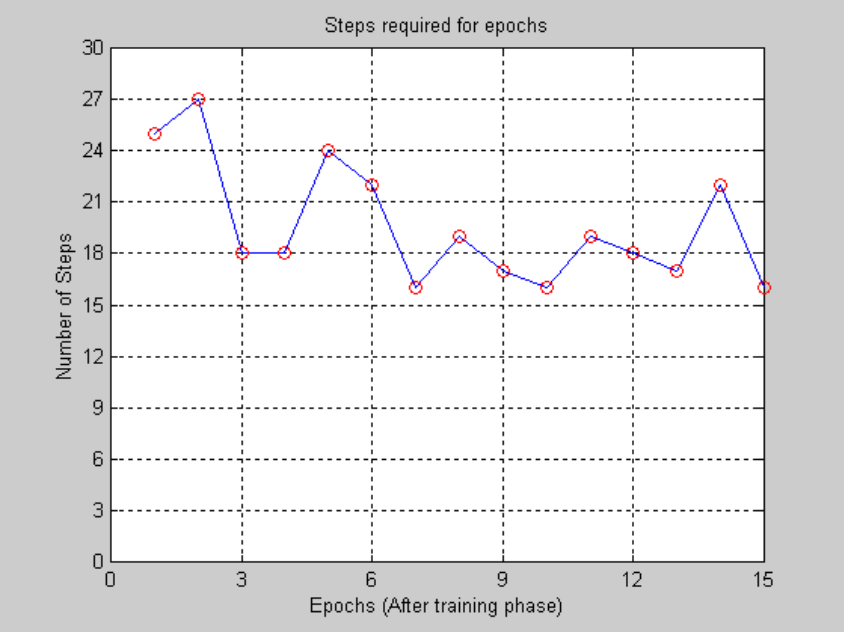


Figure 7.17 Steps per epoch after training

The number of steps required per epoch after training of just 100 epochs, in AIS simulation is much smaller that the number of steps required in the Q-learning simulation, after 30,000 epochs. This result reflects the speedy convergence of AIS learning.

7.4.6 Group Strategy Evolution Simulation – Dimension Independence

AIS network as used in above section in 7 x 7 grid was used to test the same starting state in a 10 x 10 grid. Following figure shows the comparison between performances of the

system. No additional training than that in the above section was carried out on the network when tests were done on the 10 x 10 grid.

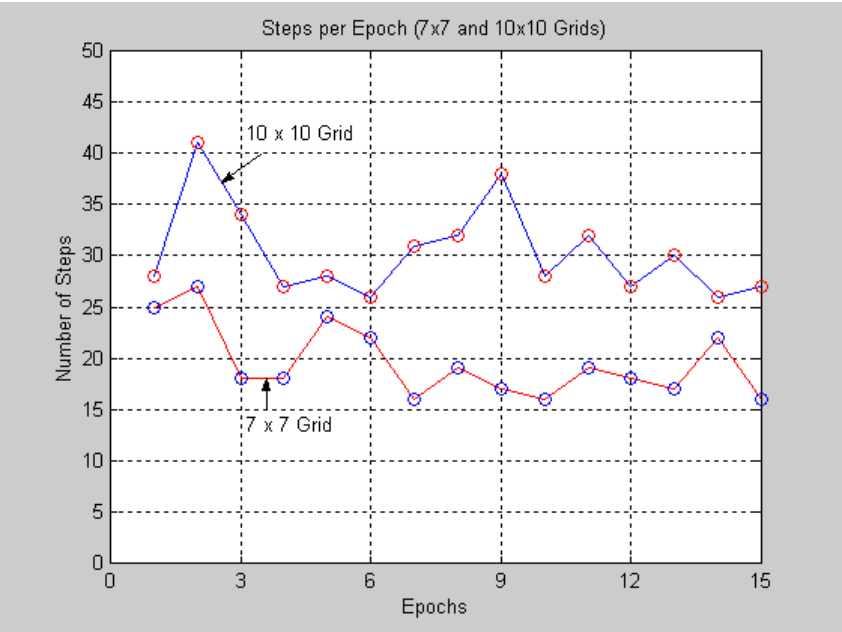


Figure 7.18 Performance comparison for different sized grids

Thus the knowledge gained by the network, in terms of concentrations of antibodies of all agents is independent of the dimension of the system. It is also independent of the number of agents in the system. All the antigens are based on the features in the local domain of pursuers and contain no information about the number of agents in the system. But it is necessary that we have more than one active agents (pursuers) as the decision making is based on the interactions of active agents unlike in multi-agent Q-learning in chapter 5 where each pursuer decided actions in isolation.

7.5 Conclusion

In this chapter, an artificial immune network was proposed to solve the herding problem using group behavior arbitration. The network learns various strategies very fast, compared to Q-learning algorithm. Also, it is independent of the dimension of the grid and the number of agents in the grid. In the next chapter another design of artificial immune system is proposed to solve the herding problem using individual action selection rather than by group strategy arbitration.

8 AIS MODELING FOR INDIVIDUAL ACTION SELECTION

8.1 Introduction

In this section AIS is designed for individual action selection by pursuer, that is, AIS is used to decide exact individual actions instead of deciding strategy for all pursuers. The objective of the immune network is to find individual actions by interaction among active agents (pursuers).

8.2 Immune network design

8.2.1 Antigens Definition for the Pursuer

Information sensed by a Pursuer in its local domain is translated into an epitope, which is a 3-tuple. This information contains the type of external agent in the local domain, range in which that agent is present and the direction of the external agent. The local domain of Pursuer is divided into two ranges – far and near - as shown in the figure below. Also, local domain is partitioned in four cells, each representing one direction.

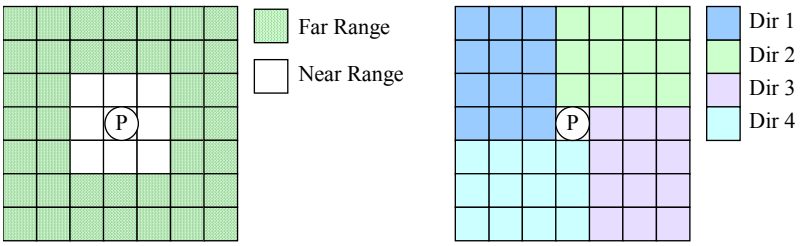


Figure 8.1 Local Domain of a Pursuer – Antigen Definition

Any antigen includes information about the type of any external agent (pursuer/pursuer or evader/evader) in the local domain, and the information about whether this external agent is in the far or the near area of the local domain.

Agent ID	Range	Direction

Figure 8.2 Antigen definition

Agent ID	P → Pursuer
	E → Evader
Range	F → Far
	N → Near
Direction	Dir 1 → 1
	Dir 2 → 2
	Dir 3 → 3
	Dir 4 → 4

Table 8.1 Antigen values – Individual Action Selection

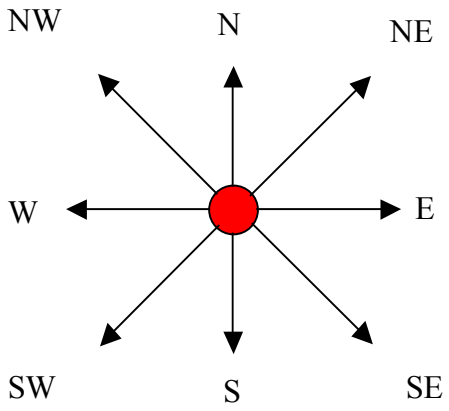
Following table lists all the possible antigens in the system.

PF1	PN1	EF1	EN1
PF2	PN2	EF2	EN2
PF3	PN3	EF3	EN3
PF4	PN4	EF4	EN4
No Agent in Local Domain - XX			

Table 8.2 Possible Antigens – Individual Action Selection

8.2.2 Antibody Definitions

Antibodies are defined as actions that a pursuer can take from any position. They are the 9 control actions in 9 directions, as defined for the Q-learning simulations. These are reproduced below.



Antibodies for Pursuer

E = EAST
NE = NORTH EAST
N = NORTH
NW = NORTH WEST
W = WEST
SW = SOUTH WEST
S = SOUTH
SE = SOUTH EAST
HERE = Stay at the same position

Figure 8.3 Antibody definitions for Active Agent (Pursuer)

8.2.3 AIS Dynamics

AIS dynamics are exactly the same as used in the immune network for strategy arbitration. For i^{th} antibody, stimulus and concentration are calculated by the following equation.

$$S_i(t) = S_i(t-1) + \left[\alpha \cdot \frac{\sum_{j=0}^{N-1} m_{ij} s_j(t-1)}{N} + \beta g_i - c_i(t-1) \right] \cdot s_i(t-1) \qquad (8.1)$$

$$s_i(t) = \frac{1}{1 + \exp[0.5 - S_i(t)]} \qquad (8.2)$$

$$c_i(t) = \eta \cdot (1 - d_i) \cdot S_i(t) \qquad (8.3)$$

where, i, j = 0, 1, …, N-1, and α, β, η are constants,

N = number of antibody types,

$S_i(t)$ = Stimulus of antibody i,

$s_i(t)$ = Concentration of antibody i,

$s_j(t)$ = Concentration of antibody j for other pursuers,

$c_i(t)$ = Concentration of T-cell controlling antibody i, d_i is the distance between pursuers

m_{ij} = Mutual stimulus coefficient between antibody i and j,

g_i = affinity of antibody i towards antigen.

8.2.4 T-cell modeling

In the network for individual action selection, concentration of the T-cell is the distance between the pursuer and the evader. Thus, if pursuer is nearer to the evader, then the stimulation of all antibodies will be more than if the evader were farther away. As the distance between the pursuer and the evader decreases, T-cell concentration decreases and thus the interactions become stronger. Following table gives qualitative behavior of the T-cell modeling in the network.

Distance	**T-cell Concentration $c_i(t)$**	**Role of T-cell**
Small	Small	Helper T-cell
Medium	Medium	Helper T-cell
Large	Large	Suppressor T-cell

Table 8.3 Qualitative effects of T-cell modeling

8.3 Artificial Idiotopic Immune Network Simulation

Immune network simulation is done under exactly same conditions as the multi-agent Q-learning. This setup gives a better ground for comparison between the immune network performance and Q-learning performance for the herding problem. Simulation contains 3 pursuers, 1 evader and the grid size is 7 x 7.

8.3.1 Immune Network Parameters

1. Mutual affinity coefficients between antibodies (m_{ij})–

	HERE	NW	N	NE	E	SE	S	SW	W
HERE	-0.1	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5
NW	-0.1	1.0	-0.1	-0.2	-0.3	-0.4	-0.3	-0.2	-0.1
N	-0.1	-0.1	1.0	-0.1	-0.2	-0.3	-0.4	-0.3	-0.2
NE	-0.1	-0.2	-0.1	1.0	-0.1	-0.2	-0.3	-0.4	-0.3
E	-0.1	-0.3	-0.2	-0.1	1.0	-0.1	-0.2	-0.3	-0.4
SE	-0.1	-0.4	-0.3	-0.2	-0.1	1.0	-0.1	-0.2	-0.3
S	-0.1	-0.3	-0.4	-0.3	-0.2	-0.1	1.0	-0.1	-0.2
SW	-0.1	-0.2	-0.3	-0.4	-0.3	-0.2	-0.1	1.0	-0.1
W	-0.1	-0.1	-0.2	-0.3	-0.4	-0.3	-0.2	-0.1	1.0

Table 8.4 Antibody Mutual affinity coefficients (m_{ij}) – Individual Action Selection

2. Affinity of antibody towards antigen (g_i) –

	HERE	NW	N	NE	E	SE	S	SW	W
PF1	0.0	0.0	0.0	**0.25**	**0.25**	**0.25**	**0.25**	0.0	0.0
PF2	0.0	0.0	0.0	0.0	0.0	**0.25**	**0.25**	**0.25**	**0.25**
PF3	0.0	**0.25**	**0.25**	0.0	0.0	0.0	0.0	**0.25**	**0.25**
PF4	0.0	**0.25**	**0.25**	**0.25**	**0.25**	0.0	0.0	0.0	0.0
PN1	0.0	0.0	0.0	0.0	**0.25**	**0.5**	**0.25**	0.0	0.0
PN2	0.0	0.0	0.0	0.0	0.0	0.0	**0.25**	**0.5**	**0.25**
PN3	0.0	**0.5**	**0.25**	0.0	0.0	0.0	0.0	0.0	**0.25**
PN4	0.0	0.0	**0.25**	**0.5**	**0.25**	0.0	0.0	0.0	0.0
EF1	0.0	**0.4**	**0.2**	**0.2**	0.0	0.0	0.0	0.0	**0.2**
EF2	0.0	**0.2**	**0.2**	**0.4**	**0.2**	0.0	0.0	0.0	0.0
EF3	0.0	0.0	0.0	**0.2**	**0.2**	**0.4**	**0.2**	0.0	0.0
EF4	0.0	0.0	0.0	0.0	0.0	**0.2**	**0.2**	**0.4**	**0.2**
EN1	**0.3**	0.0	**0.7**	0.0	0.0	0.0	0.0	0.0	0.0
EN2	**0.3**	**0.7**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
EN3	**0.2**	0.0	**0.4**	**0.4**	0.0	0.0	0.0	0.0	0.0
EN4	**0.3**	0.0	**0.7**	0.0	0.0	0.0	0.0	0.0	0.0
XX	0.0	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**

Table 8.5 Antibody Affinities towards Antigens (g_i) – Individual Action Selection

8.3.2 Simulation results

Immune network was trained for 100 epochs, using random starting states. Tests were done on the network after 0, 10, 20, …, 100 epochs on the starting state shown below.

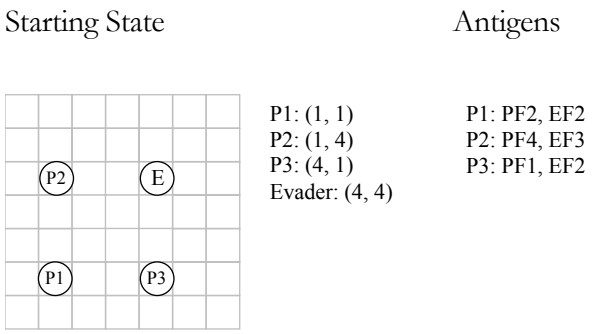


Figure 8.4 Starting state for testing the network

Following figure shows the number of steps required after training for various numbers of epochs.

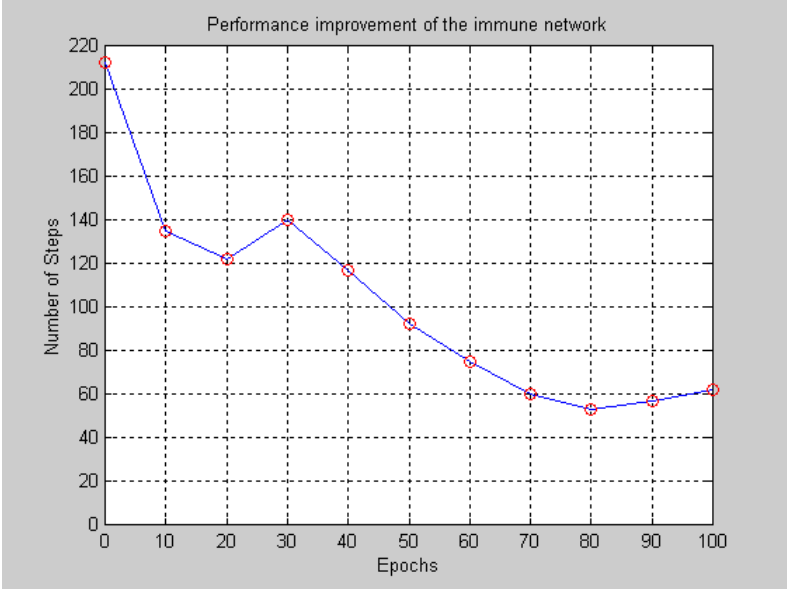


Figure 8.5 Performance improvement of the immune network

Following figures show the changes in antibody concentrations for all the three pursuers.

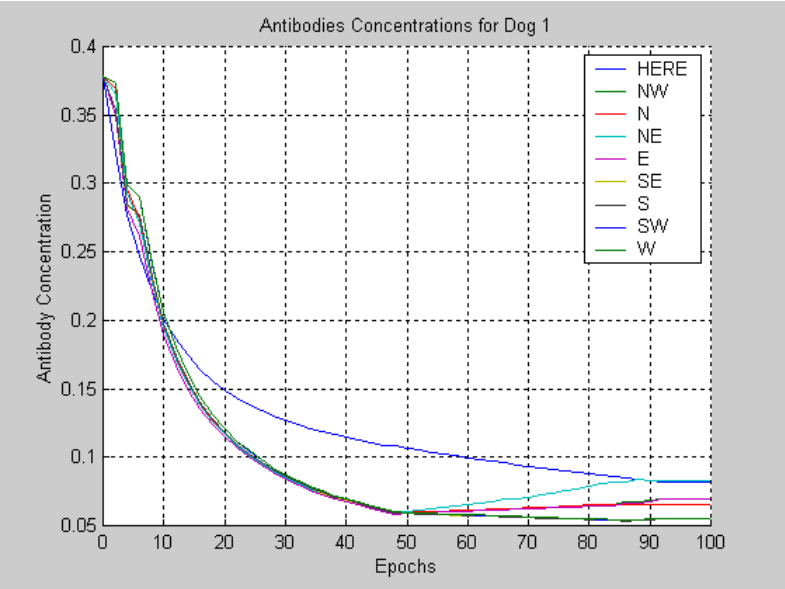


Figure 8.6 Antibody concentrations for Pursuer1

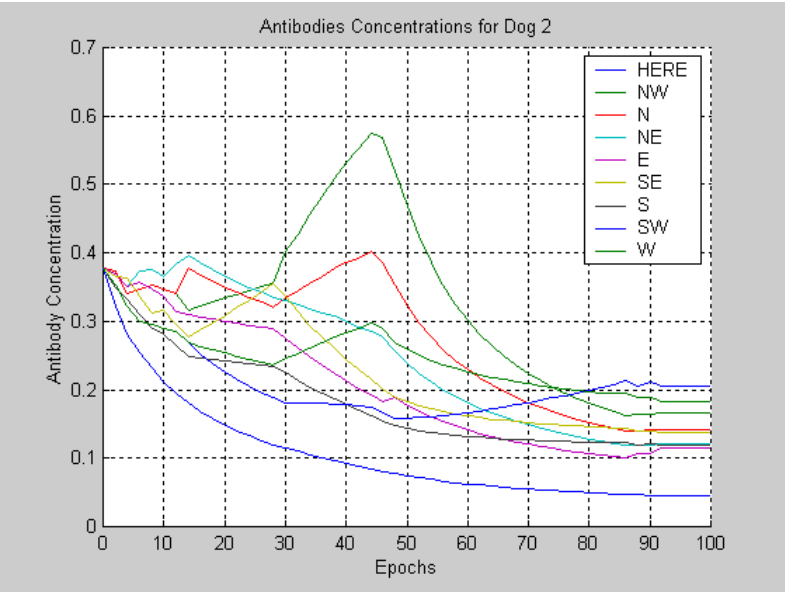


Figure 8.7 Antibody concentrations for Pursuer 2

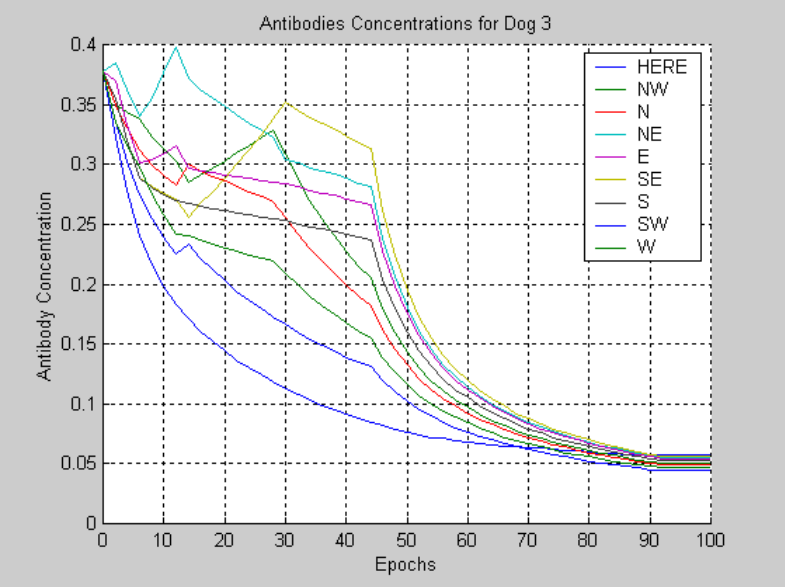


Figure 8.8 Antibody concentrations for Pursuer 3

Following figures show paths taken by all agents – all pursuers and the evader.

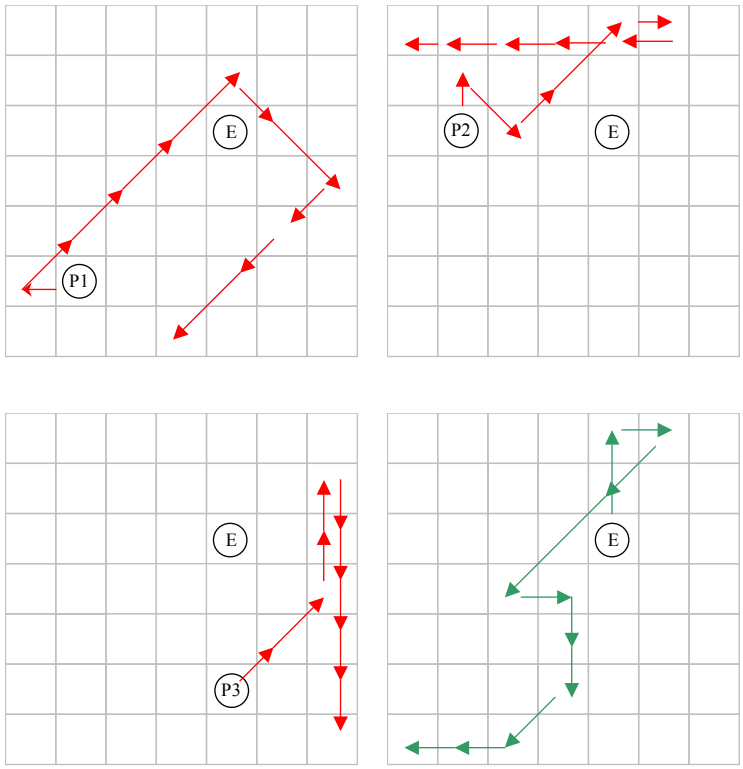


Figure 8.9 Agent trajectories

8.3.3 Action selection and strategy evolution by active agents

Fig. 8.9 shows trajectories of all agents. These trajectories give an insight into the coordination of behavior among active agents (pursuers). It is important to mention that these actions may not be optimal and that are probabilistic. For different runs we get trajectories that are overall similar to the one that we have in these figures, but they show some minor variations.

Pursuer-trajectories patterns show that each pursuer works in a specific domain. Pursuer 1 is moving so as to reach the evader. This can be seen from trajectories of Pursuer 1 and the evader. Pursuer 2 and Pursuer 3 move in the top and right flank so as to stop evader from moving in directions other than those taking it to the quadrant of the pen.

All pursuers do not try to move in a direction of the evader. This behavior will move the evader away from the pen. Instead pursuers attempt to move around the evader so that evader remains between pursuers and the pen. This strategy is evolved during the network training and was not defined *a priori*. Once evader is in this area between pursuers and the pen, it is herded to the pen without allowing exiting this area. The final formation of pursuers is similar to the Herding formation defined in the last chapter. It is important here that this formation was specified in simulations in the last chapter, whereas in this simulation, no such formation was specified. Interactions between all pursuers have resulted in this formation taking shape.

8.4 Conclusions of network modeling for individual action selection

1. Fig. 8.6, 8.7 and 8.8 show antibody concentrations for all pursuers. Concentrations for the HERE antibody monotonically decrease for all pursuers. This makes sure that all pursuers take actions proactively, without waiting for the evader to move.

2. All actions taken by pursuers are not completely deterministic given a state, as is the case in Q-learning after convergence. Action selections depend on the interactions, which depend on antibody concentrations and T-cell modeling. These values are constantly changing after every interaction unlike in Q-learning where after convergence Q-values remain unchanged. Thus, action selection is probabilistic as compared with Q-learning.

3. Pursuers formulate strategies dynamically given a state and these strategies cannot be predicted depending on the antibody concentrations only. This is similar to fuzziness that is inherent to neural networks.

9 CONCLUSIONS AND FUTURE WORK

Following conclusions can be drawn from the experimental results obtained in this research work.

For the 3 x 3 1 pursuer – 1 evader herding problem formulation, Q-learning algorithm resulted in exactly same values as were obtained by the dynamic programming solution. The model of the system was explicitly known in this case. With no model available, Q-learning converged to the same optimal solution as in case of the dynamic programming solution.

This formulation, however, is not independent of the dimension of the system and number of agents in the system and hence is not very useful for larger and more complex systems.

A formulation that is independent of the problem dimension was proposed using region-based Q-learning. This formulation did find a solution, but since a lot of information was not available to the active agent, this solution is not optimal. Thus, there is a trade-off between implementing dimension independent algorithms and the optimality of the solution.

Q-learning algorithms is computationally extensive. As system becomes more and more complex, computational requirements increase exponentially.

The artificial immune network designed for group behavior arbitration produced expected results with a lot less computation as compared to Q-learning. Similarly, the network proposed for individual action selection produced better results than Q-learning implementation with substantially less computational overhead.

Future work includes designing an artificial life system in which agents live for a specific length of time and follow population dynamics. Also, designing an immune network

for systems that contain agents with conflicting goals is a challenge. Such a system may be an intrusion-prevention system on networks.

REFERENCES

[1] R. E. Bellman, "Dynamic programming", *Princeton University Press, 1957.*

[2] D. P. Bertsekas, J. N. Tsitsiklis, "Neuro-Dynamic Programming", *Athena Scientific, 1996.*

[3] Pushkin Kachroo, Samy Shedied, John S. Bay, Hugh VanLandingham, "Dynamic Programming Solution for a Class of Pursuit Evasion Problems: The Herding Problem", *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Volume 31 - Issue 1, Feb. 2001, pp. 35 –41.*

[4] Roland Johnson, Albert Esterline, "Strategies in Multi-Agent Systems", *Proceedings of the IEEE - Southeastcon 2000, pp. 91 –94.*

[5] Andrew Garland, Richard Alterman, "Learning Cooperative Procedures", *AIPS workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments, June 1998.*

[6] Andrew Garland, Richard Alterman, "Multi-Agent Learning through Collective Memory", *AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multi-agent Systems, 1996, pp. 33-38.*

[7] Andrew Garland, Richard Alterman, "Pragmatic Multi-Agent Learning", *AAAI Doctoral Consortium, 1998.*

[8] Ming Tan, "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents", *Proceedings of the Tenth International Conference on Machine Learning, 1993, pp. 330-337.*

[9] Michael L. Littman, "Markov Games as a Framework for Multi-agent Reinforcement Learning", *11th International Conference on Machine Learning, 1994, pp. 157-163.*

[10] Sachiyo Arai, Katia Sycara, "Multi-Agent Reinforcement Learning for Planning and Conflict Resolution in a Dynamic Domain", *Proceedings of the fourth international conference on Autonomous agents, 2000, pp. 104-105.*

[11] Sachiyo Arai, Katia Sycara, T. R. Payne, "Multi-agent reinforcement learning for planning and scheduling multiple goals", *Proceedings of Fourth International Conference on Multi-Agent Systems, 2000, pp. 359-360.*

[12] Isamu Kawaishi, Seiji Yamada, "Experimental Comparison of a Heterogeneous Learning Multi-Agent System with a Homogeneous One", *IEEE International Conference on Systems, Man and Cybernetics, Volume 1, 1996, pp. 613-618*.

[13] Mark Humphrys, "Action Selection methods using Reinforcement Learning", *Ph.D. Dissertation, University of Cambridge, June 1997, Chapters 2, 3, 4*.

[14] Ou Haitao, Zhang Weidong, Xu Xiaoming, "A Novel Multi-Agent Q-learning Algorithm in Cooperative Multi-Agent System", *Proceedings of the 3rd World Congress on Intelligent Control and Automation, Volume 1, 2000, pp. 272 –276*.

[15] I. H. Suh, J. H. Kim, S. R. Oh, "Region-based Q-learning for Intelligent Robot Systems", *IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1997, pp. 172 –178*.

[16] Dimitri P. Bertsekas, "New Value Iteration and Q-learning Methods for the Average Cost Dynamic Programming Problem", *Proceedings of the 37th IEEE Conference on Decision and Control, Volume 3, Dec. 1998, pp. 2692 –2697*.

[17] George H. John, "When the Best Move Isn't Optimal: Q-learning with Exploration", *Proceedings, Tenth National Conference on Artificial Intelligence, 1994, AAAI Press, pp. 1464-1470*.

[18] Norihiko Ono, Osamu Ikeda, Kenji Fukumoto, "Acquisition of Coordinated Behavior by Modular Q-learning Agents", *IEEE/RSJ Proceedings on Intelligent Robots and Systems, International Conference 1996, Volume 3, pp. 1525-1529*.

[19] J. H. Kim, I. H. Suh, S. R. Oh, Y. J. Cho, Y. K. Chung, "Region-based Q-learning using Convex Clustering Approach", *IEEE Proceedings of the Intelligent Robots and Systems, International Conference, Volume 2, 1997, pp. 601-607*.

[20] Yoichi Hirashima, Youji Iiguni, Akira Inoue, Shiro Masuda, "Q-learning Algorithm Using an Adaptive-Sized Q-table", *Proceedings of the 38th IEEE Conference on Decision and Control, Volume 2, 1999, pp. 1599 –1604*.

[21] David J. Finton, "An Application of Importance-Based Feature Extraction in Reinforcement Learning", *IEEE Proceedings of Neural Networks for Signal Processing, 1994, pp. 52 –60*.

[22] Chi-Hyon Oh, Tomoharu Nakashima, Hisao Ishibuchi, "Initialization of Q-values by Fuzzy Rules for Accelerating Q-Learning", *The 1998 IEEE International Joint Conference on Neural Networks and Computational Intelligence, Volume 3, 1998, pp. 2051 –2056*.

[23] Chi-Hyon Oh, Tomoharu Nakashima, Hisao Ishibuchi, "Fuzzy Q-learning for a Multi-Player Non-Cooperative Repeated Game", *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems, Volume 3, 1997, pp. 1573 –1579*.

[24] Hamid R. Berenji, Sujit K. Saraf, "Competition and Collaboration among Fuzzy Reinforcement Learning Agents", *The 1998 IEEE International Conference on Fuzzy Systems, Volume 1, 1998, pp. 622 –627*.

[25] Jan Paredis, "Learning at the Crossroads of Biology and Computation", *IEEE Proceedings of Intelligence in Neural and Biological Systems, First International Symposium, 1995, pp. 56-63*.

[26] D. Dasgupta (Editor), "Artificial Immune Systems and Their Applications", *Springer-Verlag Publication, Berlin Heidelberg, 1999*.

[27] Lernardo Nunes de Castro, Fernando José Von Zuben, "Artificial Immune Systems: Part I – Basic Theory and Applications", *Technical Report, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Capinas, Brazil, Dec 1999*.

[28] Lernardo Nunes de Castro, Fernando José Von Zuben, "Artificial Immune Systems: Part II – A Survey of Applications", *Technical Report, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Capinas, Brazil, Dec 1999*.

[29] Dong-Wook Lee, Hyo-Byung Jun, Kwee-Bo Sim, "Artificial Immune System for Realization of Cooperative Strategies and Group Behavior in Collective Autonomous Mobile Robots", *Proceedings of Fourth International Symposium on Artificial Life and Robotics, 1999, pp. 232-235*.

[30] Dong-Wook Lee, Hyo-Byung Jun, Kwee-Bo Sim, "Realization of Cooperative Strategies and Swarm Behavior in Distributed Autonomous Robotic Systems Using Artificial Immune System", *IEEE Conference Proceedings of Systems, Man, and Cybernetics, Volume 6, 1999, pp. 614 –619*.

[31] John E. Hunt, Denise E. Cooke, "An Adaptive Distributed Learning System based on the Immune System", *IEEE International Conference on Systems, Man and Cybernetics, Volume 3, 1995, pp. 2494 –2499*.

[32] Akio Ishiguro, Toshiyuki Kondo, Yuji Watanabe, Yoshiki Uchikawa, "Dynamic Behavior Arbitration of Autonomous Mobile Robots Using Immune Networks", *IEEE International Conference on Evolutionary Computation, Volume 2, 1995, pp. 722 –727*.

[33] Akio Ishiguro, Toshiyuki Kondo, Yuji Watanabe, Yoshiki Uchikawa, "Decentralized Consensus-making Mechanisms Based on Immune System – Application to a Behavior Arbitration of an Autonomous Mobile Robot", *Proceedings of IEEE International Conference on Evolutionary Computation, 1996, pp. 82 –87*.

[34] Hossam Meshref, Hugh VanLandingham, "Artificial Immune Systems: Application to Autonomous Agents", *IEEE International Conference on Systems, Man, and Cybernetics, Volume 1, 2000, pp. 61 –66*.

VITA

Aditya Gadre

Aditya Gadre received his Bachelor of Engineering in Instrumentation and Controls in 1998 from University of Pune, India. After receiving his Bachelor degree, he went to Virginia Tech to pursue a Master of Science in Electrical Engineering with a concentration in Controls. Aditya is now continuing at the Virginia Tech for his doctoral studies in control systems.