Cooperative Coevolution of Multi-Agent Systems: Demonstrated in Prey Capture Task

By

Md. Shohrab Hossain
Roll No. 040305041P

Submitted for the partial fulfillment of the requirements for the degree of M. Sc. Engineering in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh

January, 2007

The thesis titled "**Cooperative Coevolution of Multi-Agent Systems: Demonstrated in Prey-Capture Task**" submitted by **Md. Shohrab Hossain**, Roll No. 040305041P, Session April 2003 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Science and Engineering in Computer Science and Engineering held on January 22, 2007.

Declaration

I, hereby, declare that the work presented in this thesis is done by me under the supervision of Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000. I also declare that neither this thesis nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

[illegible] 22/1/07

Md. Shohrab Hossain

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

All praises are for the Almighty Allah who is the most beneficent and the most merciful.

Firstly, I would like to express my deep respect, honor and gratitude to Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, for his continuous supervision, sincere advice and overall support. His keen interest in research, deep sense of responsibility towards his students and dynamic supervision influenced me to carry out and finish this difficult work.

Secondly, I would like to thank Dr. Muhammad Masroor Ali, Professor and Head, Department of CSE, BUET for his encouragement, fruitful advice and suggestions.

I would also like to thank all other faculty members and staffs of Department of CSE, BUET, for their support and cooperation.

Finally, I would like to express my deep respect and gratitude to my family members, especially my parents, without whose encouragement, this work would have been impossible.

Abstract

The basic principles of evolution have been applied to the solution of technical problems in a wide range of domains. To successfully apply evolutionary algorithms to these increasingly complex problems, we must develop effective techniques. Coevolution refers to maintaining and evolving individuals for different roles in a common task, either in a single population or in multiple populations. This thesis work is on cooperative coevolution of multi-agent systems. A novel model, the CCMAS model has been developed for evolving multi-agent systems which have added a new dimension in the research of cooperative coevolution. This model addresses major limitations of traditional evolutionary algorithms such as the issues of problem decomposition, credit assignment to the cooperating agents, maintaining interdependencies among the agents, keeping the population diversified, considering the cost of communication etc by giving suitable solutions to these problems.

In order to justify the efficacy and accuracy of the model, we have applied the model in *predator-prey problem* which is a very complex task that is yet to be solved. The CCMAS model has been found to be advantageous over other evolutionary method in solving prey capture task. The success rates seem to be almost 95% or more. The number of generations required to evolve the predators are also much less and the less number of moves are required to catch the prey. Two predator strategies are also tested: predators without any communication and predators with communication. It is found that communicating predators perform much better than non-communicating predators. Our work also performs the sensitivity analysis of different system and environment parameters and on some of the characteristics of *predator-prey problem* likely to affect the performance of the model. Finally the resulting graphs comparing these analyses are presented.

Chapter 1 Introduction

1.1 Introduction

The basic principles of evolution have been applied to the solution of technical problems in a wide range of domains. There are a number of advantages of evolutionary algorithms (EAs) over traditional algorithms. First, they can be applied with limited knowledge about the problem. This minimal knowledge requirement is the ability to approximate the relative worth of alternative problem solutions. In EAs the term *fitness* is used to mean the relative worth of a solution as defined by some objective function. Second, EAs are less susceptible to be trapped in local optima. This is because EAs maintain a population of alternative solutions and strike a balance between exploiting regions of the search space that have previously produced fit individuals and continuing to explore uncharted territory. Third, EAs can be applied in noisy or non-stationary environments. As a result, it can be used to construct a system that exhibits some of the characteristics of biology, such as intelligence or the ability to adapt to change.

Co-evolution is the process that is observed in relation of two (or more) populations, which are both in evolutionary process, and they influence each other in some way. It is the simultaneous evolution of two or more species with coupled fitness. It refers to maintaining and evolving individuals for different roles in a common task, either in a single population or in multiple populations. Considerable work on coevolution has been done from a competitive point of view [1-2], but there are only few works on cooperative coevolution. In competitive coevolution, the roles of different individuals involve conflict or opposition and competition is introduced among individuals. One agent's loss is another one's gain and vice versa. Though the fitness computation is quite efficient (fitness of one agent is the complement of other), it may form cycles of alternative classes of strategies. Moreover, some complex task may not be solved by competitive approach. In cooperative coevolution, the agents share the rewards and penalties of successes and failures respectively. Due to the cooperation among the agents, more effective, optimum solution is possible.

In multi-agent problem solving environment, several agents work together to achieve a common objective. Due to their distributed nature, multi-agent systems can be more efficient, more robust and more flexible than centralized problem solvers. The agents need to communicate and behave cooperatively rather than greedily to accomplish a common task. Although there have been many models for cooperative coevolution, no single model addresses all the issues for better cooperation. A generalized model is therefore necessary for evolving multi-agent systems.

1.2 Literature Review

This section is organized in two parts. In the first part, previous works on competitive and cooperative coevolution have been discussed. In the latter part, previous works on prey capture task have been pointed out.

1.2.1 Work on Competitive and Cooperative Coevolution

In competitive coevolution, the fitness of one species is evaluated through competition with other competing species, rather than through an absolute fitness measure. In other words, fitness signifies only the relative strengths of the solutions; an increased fitness in one solution leads to a decreased fitness for another. Ideally, competing solutions will continually outdo one another, leading to an "**arms race**" of increasingly better solutions [3-5].

Competitive coevolution in its simplest state has just two individuals, a champion and a challenger. If the challenger is better than the champion then the challenger becomes the champion and a new challenger is chosen. Samuel's work on automated learning of checkers strategies [6] is an early example. In his work, the challenger learnt over the course of a checkers game and if its learning made it a better player then it became the champion. If the challenger was a weaker player then a new and fairly drastically different challenger would be found. After just 28 games the learnt strategy was considered better than an average human player.

Competitive coevolution may be naturally applied to evolutionary systems with single population. All that is necessary is a concept of "better": when is one individual better than another? In Juille and Pollack's work on the intertwined spirals problem [7], they used a single population for coevolution. Diversity among individuals in a population was encouraged by a competitive fitness function.

Angeline and Pollack (1993) [2] compared the solutions of *Tic-Tac-Toe* evolved by using competitive fitness against those evolved by learning to play against an "expert". The "experts" were *perfect*, *nearly perfect* and *random*, none of which offered sufficient positive feedback for evolving players. The *perfect* player could not be beaten and thus provided no positive feedback at all. The *nearly perfect* player allowed only one type of win, so provided only limited feedback. The *random* player was too easily beaten. Competitive fitness was shown to produce superior individuals.

Hillis (1991) [1] applied a model of hosts and parasites (known as *Host-parasite model*) to the evolution of sorting networks using genetic algorithm. Technically, it is an example of a competitive model. One species (the host population) represents sorting networks and the other species (the parasite population) represents test cases in the form of sequences of numbers to be sorted. The members of a species interbreed with one another by using a Gaussian displacement rule. The host and the parasite species are genetically isolated and only interact through their fitness functions.

A competitive model was used by Rosin and Belew (1995) [9] to solve a number of game learning problems, including tic-tac-toe, nim and go. Two species represent opponents in this game. In their algorithm called *shared sampling*, each member of one species is matched against a sample of opponents from the previous generation of the other species. The sample of opponents is chosen based on their ability to win games. It was found that perfect solutions to Tic-Tac-Toe were easier to evolve than equivalent solutions for Nim. Both of these competitive models [1, 9] showed that this form of interaction between species helps to preserve genetic diversity and resulted in better final solutions. It can also result in a more computational efficient fitness function. But these two models have two limitations: they involve a hand-decomposition of the problem and they have a narrow range of applicability.

Competitive coevolution has traditionally been used in two kinds of problems. First, it can be used to evolve interactive behaviors that are difficult to evolve in terms of an absolute fitness function. For example, Sims (1994) [10] evolved simulated 3D creatures that attempted to capture a ball before an opponent did, resulting in a variety of effective interactive strategies. Second, coevolution can be used to gain insight into the dynamics of game-theoretic problems. For example, Lindgren & Johansson (2001) [11] coevolved iterated Prisoner's Dilemma strategies in order to demonstrate how they correspond to stages in natural evolution.

Whatever the goal of a competitive coevolution, interesting strategies will only evolve if the arms race continues for a significant number of generations. In practice, it is difficult to establish such an arms race. Evolution tends to find the simplest solutions that can win, meaning that strategies can switch back and forth between different idiosyncratic yet uninteresting variations (Darwen, 1996; Floreano and Nolfi, 1997; Rosin and Belew, 1997) [12-14]. Several methods have been developed to encourage the arms race (Angeline and Pollack, 1993; Ficici and Pollack, 2001; Noble and Watson, 2001; Rosin and Belew, 1997) [2, 14-16]. For example, a "hall of fame" or a collection of past good strategies can be used to ensure that current strategies remain competitive against earlier strategies. Recently, Ficici and Pollack (2001) [15] and Noble and Watson (2001)[16] introduced a promising method called Pareto coevolution, which finds the best learners and the best teachers in two populations by casting coevolution as a multi-objective optimization problem.

Although such techniques allow sustaining the arms race longer, they do not directly encourage continual coevolution, i.e. creating new solutions that maintain existing capabilities. For example, no matter how well selection is performed, or how well competitors are chosen, if the search space is fixed, a limit will eventually be reached. Also, it may occasionally be easier to escape a local optimum by adding a new dimension to the search space than by searching for a new path through the original space.

A genetic algorithm for coevolving multiple cooperative species was applied to job-shop scheduling by Husbands (1991). Typically, job-shop scheduling systems generate an optimum schedule given a set of descriptions for machining parts called process plans. In

[17], each species represents alternatives for a single process plan, and is genetically isolated from the other species through the use of multiple populations.

Peredis (1995) [18] used a cooperative two-species model (referred to as symbiotic model) for the solution of a deceptive problem introduced by Goldberg (1989). One species represents problem solutions as 30-bit vectors and the other species represents permutations on the order of the bits in the solutions. Since it was known beforehand which permutations would be helpful, they were coevolved along with the solutions. But this system involves a hand-decomposition of the problem.

Gomez and Miikkulainen [19] developed a method called Enforced Subpopulations (ESP) to evolve populations of neurons to form a neural network. A neuron was selected from each population to form the hidden-layer units of a neural network, which was evaluated on the problem; the fitness was then passed back to the participating neurons. Potter and De Jong [20] outlined an architecture and process of cooperative coevolution that is similar to ESP model. They focused on methodological issues such as how to automatically decompose the problem into an appropriate number of subcomponents and roles.

Haynes and Sen explored various ways of encoding, controlling, and evolving predators that behave cooperatively in the predator-prey domain in a series of papers [21-23]. In the first of these studies [22], Genetic Programming was used to evolve a population of strategies. The predators were thus said to be homogeneous, since they each shared the same behavioral strategy. In follow-up studies [21, 23], they developed heterogeneous predators: each chromosome in the population was composed of different programs. They showed that the heterogeneous predators were able to perform better than the homogeneous ones.

Wagner [26] suggested that even in domains where communication does contribute towards solving a task, communicative traits may not evolve if they involve a significant cost. Balch [27] examined the behavioral diversity learned by robot teams using reinforcement learning. He found that when the reinforcement was local, i.e. applied separately to each agent, the agents within the team learned identical behavior; global reinforcement shared by all agents, on the other hand, produced teams with more heterogeneous behavior.

In summary, the coevolutionary approach seems to be a good match for multi-agent tasks. The predator-prey domain is a simple but effective domain to test this hypothesis. So our model of cooperative coevolution for multi-agent systems (CCMAS) is applied on predator-prey domain. Previous studies on that domain are mentioned in the next section.

1.2.2 Work on Predator-Prey Problem

The predator-prey pursuit problem is a general and well studied multi-agent problem that still has not been solved. It was first introduced by Benda, Jagannathan, and Dodhiawalla [28]. This problem has been used to study phenomena like competitive coevolution, multi-agent strategies and multi-agent communication.

Jim and Giles (2001)[29] showed that introducing noise to the decision process of human-designed predator strategies allows the predators to overcome deadlock and thus outperform predator strategies both programmed and learned in all previously published work. They also showed that a genetic algorithm can evolve communicating predators that outperform the best evolved non-communicating predators, and that increasing the language size for communication among the predators improves performance [30].

Haynes and Sen [22] used genetic programming to evolve predator strategies and showed that a linear prey (pick a random direction and continue in that direction for the rest of the trial) was impossible to capture reliably in their experiments because such prey avoids locality of movement. Korf [32] studied a version of the predator-prey problem in which the predators were allowed to move diagonally as well as orthogonally and the prey moved randomly. Tan [33] used reinforcement learning and showed that cooperating agents that share sensations and learned policies with each other significantly outperform non-cooperating agents in a version of the predator-prey problem.

Nishimura and Ikegami [34] observed random swarming and other collective predator motions in a predator-prey game. Stephens and Merx [35] studied a simple non-communicating predator strategy in which predators move to the closest capture position and showed that this strategy is not very successful because predators can block each other by trying to move to the same capture position. Stephens and Merx also present another strategy in which three predators transmit all their sensory information to one central predator agent who decides where all predators should move. This central single-agent strategy succeeded for 30 test cases, but perhaps the success rate would be much lower if the agents were to move simultaneously instead of taking turns. Our study uses an implementation that is probably more difficult for the predators than those used in all previous works.

1.3 Aim of the Thesis

As mentioned in the previous section, no single cooperative coevolutionary model addresses all the issues for better performance in a team work. So we try to develop a generalized model for evolving multi-agent systems. The aims of this thesis work are as follows:

- To develop a generalized model for Cooperative Coevolution
- Applying the proposed model for a complex task of predator-prey problem
- Find out the advantages and drawbacks of the proposed model
- Compare the performance of the proposed model with the conventional models in solving the complex problem by using the following comparison criteria.
    - Time to complete the task
    - Number of generations required
    - Average success rate
    - Average number of moves to capture

1.4 Thesis Organization

The rest of the thesis is organized as follows.

In Chapter 2, a brief description of different kinds of evolutionary algorithms is given. Different components and terms related to evolutionary computation are also explained.

In Chapter 3, terms related to coevolution, competitive coevolution and cooperative coevolution are explained with examples. Some advantages and drawbacks of competitive and cooperative co-evolution along with the major design issues have been pointed out.

In Chapter 4, the generalized model of cooperative coevolution for multi-agent systems (CCMAS) is described in details along with the key features of the model.

In Chapter 5, we present the results of our experiment on predator-prey domain using our cooperative coevolutionary model. We show that our new model is capable of finding better solutions of the prey capture task.

Finally, in Chapter 6, we summarize our work and give directions for future works.

Chapter 2
Evolutionary Algorithms

2.1 Introduction

The chapter introduces different types of Evolutionary Algorithms. This description is deliberately based on a unifying view presenting a general scheme that forms the common basis of all Evolutionary Algorithm variants. The main components of Evolutionary Algorithms are also discussed, along with their roles and related issues of terminology.

2.2 Concepts of Evolutionary Algorithms

Let us consider the conventional approach of solving a problem. Let X be a problem with N possible solutions. Conventional algorithm picks one solution at a time. If the performance of that solution is not satisfactory, it is thrown away. The cost, efforts, partial knowledge, experience in finding that solution- all are discarded. Thus the final optimal solution only contains the result of the best one.

But in EAs, a *subset* of *candidate solutions* is selected simultaneously. If the performance of that subset of solutions is not good enough, they are not dropped altogether. The experience and efforts are reused. Thus all the partial solutions have an impact on the final solutions. Since the evolutionary approach deals with a subset of solutions all the time, it is somewhat slower but still can produce more experienced, effective result. There are many different variants of EAs. They are all based on the same fundamental **principles of Darwinian evolution**. These principles are as follows:

- Organisms have a finite lifetime; therefore, propagation is necessary for the continuation of the species.
- Offspring vary to some degree from parents.
- The organisms exist in an environment in which survival is a struggle and the variations among them will enable some to better adapt to this difficult environment.
- Through natural selection, the better-adapted organisms will tend to live longer and produce more offspring.

- Offspring are likely to inherit beneficial characteristics from their parents, enabling members of the species to become increasingly well adapted to their environment over time.

The common underlying idea behind all the evolutionary techniques is quite similar: given a population of individuals, the environmental pressure causes natural selection (survival of the fittest) and this causes a rise in the fitness of the population. Based on this fitness, some of the better candidates are chosen to seed the next generation by applying *recombination* and/or *mutation* to them. This process can be iterated until a candidate with sufficient quality (a solution) is found or a previously set computational limit is reached. In this process, there are two fundamental forces that form the basis of evolutionary systems.

- Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty, while
- Selection acts as a force pushing quality.

The general scheme of an Evolutionary Algorithm is given below in a pseudo-code fashion.

```
INITIALIZE population with random candidate solutions;
EVALUATE each candidate;
REPEAT UNTIL (TERMINATION CONDITION is satisfied )
        1 SELECT parents;
        2 RECOMBINE pairs of parents;
        3 MUTATE the resulting offspring;
        4 EVALUATE new candidates;
        5 SELECT individuals for the next generation;
```

Algorithm 2.1 The general scheme of an evolutionary algorithm

It can be noted that this scheme falls in the category of generate-and-test algorithms. The fitness evaluation function represents a heuristic estimation of solution quality and the search process is driven by the variation and the selection operators. Evolutionary Algorithms (EA) posses a number of features that can help to position them within in the family of generate-and-test methods:

- EAs are population based, i.e., they process a whole collection of candidate solutions simultaneously.
- EAs mostly use recombination to mix information of more candidate solutions into a new one.
- EAs are stochastic.

2.3 Components of Evolutionary Algorithms

EAs have a number of components, procedures or operators that must be specified in order to define a particular EA. The most important components are:

- Representation
- Evaluation function (or fitness function)
- Population
- Parent selection mechanism
- Variation operators, recombination and mutation
- Survivor selection mechanism

Each of these components must be specified in order to define a particular EA. Furthermore, to obtain a running algorithm the initialization procedure and a termination condition must be also defined.

2.3.1 Representation

The first step in defining an EA is to link the "real world" to the "EA world", that is to set up a bridge between the original problem context and the problem solving space where evolution will take place. Objects forming possible solutions within the original problem context are referred to as *phenotypes*, their encoding, the individuals within the EA, are called *genotypes*. The first design step is commonly called representation, as it amounts to specifying a mapping from the phenotypes onto a set of genotypes that are said to represent these phenotypes. For instance, given an optimization problem on integers, the given set of integers would form the set of phenotypes. Then one could decide to represent them by their binary code, hence "18" would be seen as a phenotype and "10010" as a genotype representing it. It is important to understand that the phenotype space can be very different from the genotype space, and that the whole evolutionary search takes place in the genotype space. A solution (i.e., a good phenotype) is obtained by decoding the best genotype after termination. To this end, it should hold that the (optimal) solution to the problem at hand - a phenotype - is represented in the given genotype space.

2.3.2 Evaluation Function (Fitness Function)

The role of the ***evaluation function*** is to form the basis for selection, and thereby it facilitates improvements. More accurately, it defines what improvement means. From the problem solving perspective, it represents the task to solve in the evolutionary context. Technically, it is a function or procedure that assigns a quality measure to genotypes. Typically, this function is composed of a quality measure in the phenotype space and the inverse representation. To remain with the above example, if we were to maximize x^2 on integers, the fitness of the genotype 10010 could be defined as the square of its corresponding phenotype: $18^2 = 324$. The evaluation function is commonly called the ***fitness function.***

2.3.3 Population

The role of the population is to hold (the representation of) possible solutions. A population is a multi-set of genotypes. Individuals are static objects not changing or adapting, it is the population that does. Given a representation, defining a population can be as simple as specifying how many individuals are in it, that is, setting the population size. As opposed to variation operators that act on the one or two parent individuals, the selection operators work at population level. For instance, the best individual of *the given population* is chosen to seed the next generation, or the worst individual *of the given population* is chosen to be replaced by a new one. In almost all EA applications the population size is constant, not changing during the evolutionary search.

2.3.4 Parent Selection Mechanism

The role of parent selection or mating selection is to distinguish among individuals based on their quality, in particular, to allow the better individuals to become parents of the next generation. Together with the survivor selection mechanism, parent selection is responsible for pushing quality improvements. Parent selection is typically probabilistic. Thus, high quality individuals get a higher chance to become parents than those with low quality. Nevertheless, low quality individuals are often given a small, but positive chance; otherwise the whole search could become too greedy and get stuck in a local optimum.

There are many parent selection methods, for example, roulette wheel selection, rank selection, steady state selection, Boltzman selection, tournament selection and some others. Some of them are described in this section.

2.3.4.1 Roulette Wheel Selection

Parents are selected according to their fitness: the better the fitness values, the more the chances for them to be selected. Imagine a *roulette wheel* where all chromosomes in the population are placed. Every chromosome has its place as big accordingly to its fitness function, like those in figure 2.1 with four chromosomes. The wheel is rotated and a marble is thrown into it to select a chromosome. Chromosomes with higher fitness have higher chances to be selected. In fact, an individual will have a probability of f(i)/ ∑f(i) to be chosen.

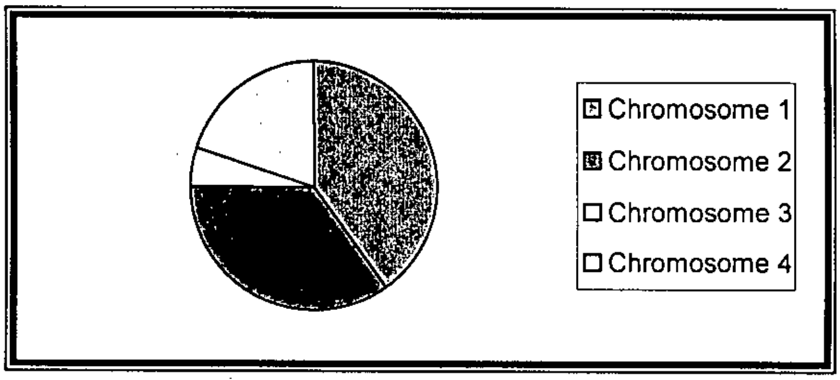


Figure 2.1: Roulette wheel selection

2.3.4.2 Rank Selection

The Roulette wheel selection will have problems when the fitness values differ very much. For example, if the best individual's fitness is 90% of the roulette wheel, then the other individuals will have very few chances to be selected. Rank selection first ranks the population and then every individual receives fitness from this ranking. The worst will have fitness *1*, second worst *2* etc. and the best will have fitness *N* (where N is the number of chromosomes in population).

2.3.4.3 Steady-State Selection

This is not a particular method of selecting parents. Main idea of this selection is that big part of chromosomes should survive to the next generation. GA then works in the following way. In every generation, a few good (with higher fitness) chromosomes are selected for creating a new offspring. Then some bad (with lower fitness) chromosomes are removed and the new offspring is placed in their place. The rest of the population survives to the new generation.

2.3.4.4 Elitism

When creating the new population by crossover and mutation, there is a good chance that the best individuals will be lost. Elitism is a method that prevents such phenomena. It first copies the best individual (or a few best individuals) to the new population. The rest is done in classical way. Elitism can very rapidly increase performance of EA, because it prevents losing the best found solution.

2.3.5 Variation Operators

The role of variation operators is to create new individuals from old ones. In the corresponding phenotype space this amounts to generating new candidate solutions. From the *generate-and-test* search perspective, variation operators perform the "generate" step. Variation operators in EC are divided into two types based on their arity: recombination and mutation.

2.3.5.1Recombination

A binary variation operator is called ***recombination*** or ***crossover***. Crossover exchanges information between two parent chromosomes and introduces new information for the child chromosomes. Each crossover operator takes the characteristics of the parents to form the chromosomes of the children. The main purpose of crossover is to exchange information between randomly selected parent chromosomes with the aim of not losing any important information. Actually, it recombines genetic material of two parent chromosomes to produce offspring for the next generation. Recombination is a stochastic operator: the choice of what parts of each parent are combined, and the way these parts are combined, depends on random drawings.

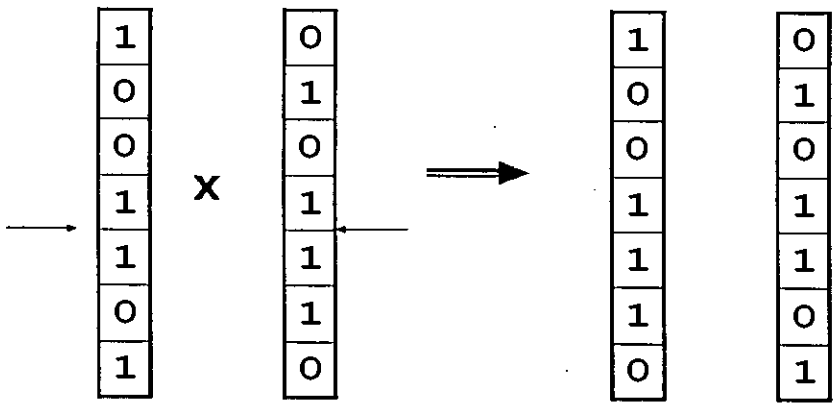


Figure 2.2 Single point crossover

The most important trait of a crossover operator is how offspring differ in, how many children are produced and the method in which they are produced. The crossover operator has the property of inheriting the good traits from parents and discovering the new traits by its own in the next generation. It also keeps the algorithm out of local minimum.

2.3.5.2 Mutation

Mutation is the second way an evolutionary algorithm explores the fitness surface. It is a unary variation operator. It is applied to one genotype and delivers a (slightly) modified mutant, the child or offspring of it. A mutation operator is always stochastic: its output the child depends on the outcomes of a series of random choices. It should be noted that an arbitrary unary operator is not necessarily seen as mutation. A problem specific heuristic operator acting on one individual could be termed as mutation for being unary. However, in general, mutation is supposed to cause a random, unbiased change.

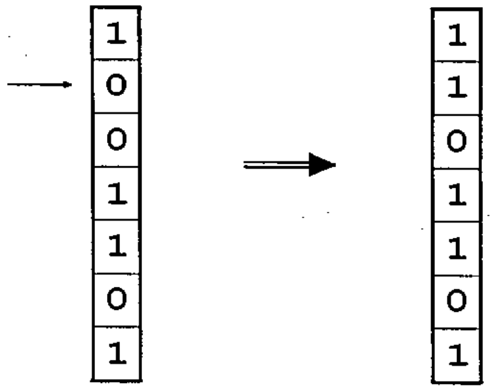


Figure 2.3 Mutation

Mutation causes the search to seek out alternate routes, ideally leading to the global optimum. In the computer, mutation is simulated by flipping randomly selected bits (in binary genetic algorithms) or replacing an entire floating-point parameter by a new random value (in continuous genetic algorithms). The role of mutation is to insert new diversity into a population. Homogenous populations may suffer from over-specialization; inserting random genes may create individuals that vary wildly from the norm, allowing the overall process to move away from sub-optimal solutions. Mutation rates should diminish as the population converges towards a satisfactory solution.

2.3.6 Survivor Selection Mechanism

The role of survivor selection is to distinguish among individuals based on their quality. It is similar to parent selection, but it is used in a different stage of the evolutionary cycle. The survivor selection mechanism is called after having created the offspring of the selected parents. As mentioned in section 2.2.3, in EC the population size is (almost always) constant, thus a choice has to be made on which individuals will be allowed in the next generation. This decision is usually based on their fitness values, favoring those with higher quality, although the concept of age is also frequently used. As opposed to parent selection (which is typically stochastic), survivor selection is often deterministic, for instance ranking the unified multi-set of parents and offspring and selecting the top segment (fitness biased), or selecting only from the offspring (age-biased).

2.3.7 Initialization

Initialization is kept simple in most EA applications. The first population is seeded by randomly generated individuals. In principle, problem specific heuristics can be used in this step aiming at an initial population with higher fitness. Whether this is worth the extra computational effort or not is very much depending on the application at hand. There are, however, some general observations concerning this issue based on the so-called anytime behavior of EAs.

2.3.8 Termination Condition

As for a suitable termination condition we can distinguish two cases. If the problem has a known optimal fitness level, probably coming from a known optimum of the given objective function, then reaching this level (perhaps only with a given precision $\varepsilon > 0$) should be used as stopping condition. However, EAs are stochastic and mostly there are no guarantees to reach an optimum, hence this condition might never get satisfied and the algorithm may never stop. This requires that this condition is extended with one that certainly stops the algorithm. Commonly used options for this purpose are the following:

- Maximally allowed CPU time elapses
- Total number of fitness evaluations reaches a given limit
- For a given period of time (i.e, for a number of generations or fitness evaluations), the fitness improvement remains under a threshold value
- Population diversity drops under a given threshold

The actual termination criterion in such cases is a disjunction: optimum value hit or condition x satisfied. If the problem does not have a known optimum, then we need no disjunction, simply a condition from the above list or a similar one that is guaranteed to stop the algorithm.

2.4 Types of Evolutionary Algorithm

A variety of EAs have been proposed. The major ones are: genetic algorithms, evolutionary programming, evolution strategies and genetic programming. They all share a common conceptual base of simulating the evolution of individual structures via processes of selection, mutation, and reproduction. They differ only in technical details (such as, representation of a candidate solution). Typically, the candidates are represented by (i.e., the data structure encoding a solution has the form of) strings over a finite alphabet in Genetic Algorithms (GA), real-valued vectors in Evolution Strategies (ES), finite state machines in classical Evolutionary Programming (EP) and trees in Genetic Programming (GP). It is important to note that the recombination and mutation operators working on candidates must match the given representation. Thus for instance in GP, the recombination operator works on trees, while in GAs it operates on strings. As opposed to variation operators, selection takes only the fitness information into account; hence it works independently from the actual representation. Differences in the commonly applied selection mechanisms in each stream are therefore rather a tradition than a technical necessity.

2.5 Genetic Algorithm

Genetic Algorithm was developed by John Holland in 1975 [8]. Genetic Algorithms (GAs) are search algorithms based on the mechanics of the natural selection process (biological evolution). The most basic concept is that the strong tend to adapt and survive while the weak tend to die out. That is, optimization is based on evolution, and the "Survival of the fittest" concept.

2.5.1 The Steps of Genetic Algorithm

Genetic Algorithms have the ability to create an initial population of feasible solutions, and then recombine them in a way to guide their search to only the most promising areas of the state space. In GAs, the parameters of the search space are encoded in the form of

```
0 START   : Create random population of n chromosomes
1 FITNESS : Evaluate fitness f(x) of each chromosome in the population
2 NEW POPULATION
            0 SELECTION      : Based on f(x)
            1 RECOMBINATION  : Cross-over chromosomes
            2 MUTATION       : Mutate chromosomes
            3 ACCEPTATION    : Reject or accept new one
3 REPLACE : Replace old with new population: the new generation
4 TEST    : Test problem criterium
5 LOOP    : Continue step 1 – 4 until criterium is   satisfied
```

Algorithm 2.2: The basic genetic algorithm

strings (called *chromosomes*). A group of individuals that interact (breed) together is called a *population*. Initially, a random population is created, which represents different points in the search space. *Fitness* is a value associated with a chromosome that assigns a relative merit of that chromosome. An objective and fitness function as associated with each string that represents the degree of goodness of the string. Based on the principle of survival of the fittest, a few of the strings are selected and each is assigned a number of copies that go into the mating pool. Biologically inspired operators like *crossover* and *mutation* are employed to form a new chromosome from parent chromosomes by combining part of the information from each. The process of selection, crossover and mutation continues for a fixed number of generations or until a termination condition is satisfied.

Most important parameters in GAs are population size, evaluation function, crossover method and mutation rate. Determining the size of the population is a crucial factor. Choosing a population size too small increases the risk of converging prematurely to local minima, since the population does not have enough genetic material to sufficiently cover the problem space. On the other hand, a larger population has a greater chance of finding the global optimum at the expense of more CPU time. The population size remains constant from generation to generation.

2.5.2 Advantages of Genetic Algorithm

Genetic Algorithm is a robust search technique. It produces "close" to optimal results in a "reasonable" amount of time. It is suitable for parallel processing and can use a noisy fitness function. Moreover it is fairly simple to develop and makes no assumptions about the problem space. GA is blind without the fitness function. The fitness function drives the population toward better solutions and is the most important part of the algorithm.

2.5.3 When to Use Genetic Algorithm?

Genetic Algorithms can be used when

- An acceptable solution representation is available.
- A good fitness function is available.
- A near-optimal, but not optimal solution is acceptable.
- The state-space is too large for other methods.

2.5.4 Difference with Traditional Algorithm

The main differences of Genetic Algorithms with Traditional Algorithm are:

- The GA works with a coding of the parameter rather than the actual parameter.
- The GA works from a population of strings instead of a single point.
- Application of GA operators causes information from the previous generation to be carried over to the next.
- The GA uses probabilistic transition rules, not deterministic rules.

2.6 Evolutionary Programming

Evolutionary programming (EP), originally conceived by Lawrence J. Fogel in 1960, is a stochastic optimization strategy similar to GA, but instead places emphasis on the behavioral linkage between parents and their offspring, rather than seeking to emulate specific genetic operators as observed in nature. EP is similar to evolution strategies (ES). EP does not use any crossover as a genetic operator [36].

EP is a useful method of optimization when other techniques such as gradient descent are not possible. Combinatoric and real-valued function optimization, where the optimization surface or fitness landscape is *rugged*, possessing many locally optimal solutions, are well suited for EP. In EP, there is an underlying assumption that a fitness landscape can be characterized in terms of variables, and that there is an optimum solution (may be multiple) in terms of those variables. For example, if one were trying to find the shortest path in a Traveling Salesman Problem, each solution would be a path. The length of the path could be expressed as a number that would serve as the solution's fitness. The goal would be to find the globally shortest path in that space.

2.6.1 Steps of Basic Evolutionary Programming

The basic EP method involves three steps.

- Choose an initial population of solutions at random. The number of solutions in a population is highly relevant to the speed of optimization.
- Each solution is replicated into a new population. Each of these offspring solutions are mutated according to a distribution of mutation types, ranging from minor to extreme with a continuum of mutation types between.
- Each offspring solution is assessed by computing its fitness. Typically, a stochastic tournament is held to determine N solutions to be retained for the population of solutions, although this is occasionally performed deterministically.

Repeat these steps until an adequate solution is obtained or a predefined number of iterations.

2.6.2 Differences with Genetic Algorithms

There are two important ways in which EP differs from GAs.

First, there is no constraint on the representation. The typical GA approach involves encoding the problem solutions as a string of representative tokens, the genome. In EP, the representation follows from the problem. A neural network can be represented in the same manner as it is implemented.

Second, the mutation operation simply changes aspects of the solution according to a statistical distribution which weights minor variations in the behavior of the offspring. Further, the severity of mutations is often reduced as the global optimum is approached.

2.7 Evolution Strategy

Evolution strategies (ES) were invented to solve technical optimization problems. It is more or less similar to EP. Self-adaptation within ES depends on randomness, population size, cooperation and deterioration [37].

2.7.1 Differences with Evolutionary Programming

The main differences between Evolutionary Programming (EP) and Evolution strategy (ES) are:

- EP typically uses *stochastic selection* via a tournament. Selection then eliminates those solutions with the least wins. In contrast, ES typically uses *deterministic selection* in which the worst solutions are purged from the population based directly on their function evaluation.

- EP is an abstraction of evolution at the level of reproductive populations and thus no recombination mechanisms are typically used because recombination does not occur between species. In contrast, ES is an abstraction of evolution at the level of individual behavior. When self-adaptive information is incorporated this is purely genetic information (as opposed to phenotypic) and thus some forms of recombination are reasonable and many forms of recombination have been implemented within ES.

2.8 Genetic Programming

Genetic programming (GP) is the extension of the genetic model of learning into the space of programs. That is, the objects that constitute the population are not fixed-length character strings that encode possible solutions to the problem at hand, they are programs that, when executed, are the candidate solutions to the problem. These programs are expressed in genetic programming as parse trees, rather than as lines of code. The programs in the population are composed of elements from the function set and the terminal set. These are typically fixed sets of symbols selected to be appropriate to the problem solution.

In GP the crossover operation is implemented by taking randomly selected sub-trees in the individuals (selected according to fitness) and exchanging them. It should be pointed out that GP usually does not use any mutation as a genetic operator.

Chapter 3
Coevolution

3.1 Introduction

Coevolutionary algorithms have received increased attention in the past few years within the domain of evolutionary computation. Coevolution means simultaneous evolution of two or more species — "A trait in one species evolving in response to a trait of another species and vise versa". In coevolution, species influence one another's evolution. Here the fitness of individuals in one species cannot be evaluated independently of those in the other species. As the species evolve, the fitness landscape of each is changing [40]; this has been referred to as "coupled fitness landscapes" [41].

The co-evolutionary algorithm consists of an iterative loop of transmission, variation, and selection of individuals with certain traits. Because co-evolution of good design obviously occurs naturally, the co-evolutionary algorithm is equated to the formalization of design synthesis. Modification to the co-evolution for use on computational substrate results in the co-evolution framework that can be and is used for automatic design synthesis. The problem of the cooperative design chosen for either competitive or cooperative is subject to a dynamic environment. They must be able to change with the environment. Those that have a better ability to co-evolve can evolve more quickly in concert with the dynamic environment and thus have a better chance for continued survival. In addition, dynamic environments in nature have resulted in the evolution of adaptability, or the ability of an individual to adapt itself to an environment within its life span. A population can coevolve competitively or cooperatively or both. Competitive coevolution has relative fitness measures rather than absolute fitness. Individuals compete against each other for dominance in the population. Cooperative coevolution requires that a number of individuals work together to solve the problem.

3.2 Competitive Coevolution

In a competitive coevolutionary algorithm, the fitness of an individual is based on direct competition with individuals of other species, which in turn evolve separately in their own populations. Increased fitness of one of the species implies a reduction in the fitness of the other species. This evolutionary pressure tends to produce new strategies in the populations involved so as to maintain their chances of survival. This “arms race” ideally increases the capabilities of each species until they reach an optimum level.

Competitive coevolution has relative fitness measures rather than absolute fitness. Individuals compete against each other for dominance in the population. Both cooperation and competition occur when teams of separately evolved individuals compete against each other. Two examples of this are robot soccer (team against team) and some *predator-prey* domains where the predators are groups of individuals (team against individual). *Predator-prey* domain is the most well-known example of competitive coevolution. There is a strong evolutionary pressure for prey to defend themselves better (e.g., by running faster, growing bigger shields, better camouflage, etc.) in response to which future generations of predators develop better attacking strategies (e.g., stronger claws, better eye-sight, etc.). In such *arms races*, success on one side is felt by the other side as failure to which one must “respond” in order to maintain one’s chances of survival. This, in turn, calls for a reaction of the other side. This process of coevolution can result in a stepwise increase in complexity of both predator and prey.

3.2.1 Benefits of Competition

Competitive co-evolution has several features that may potentially enhance the adaptation power of artificial evolution. First, competing population may produce increasingly complex evolving challenges. Competing populations may reciprocally drive one another to increasing levels of behavioral complexity by producing an evolutionary “arm race” (Dawkins and Krebs, 1979) [3]. Second, artificial evolution may fail to find a solution to a complex task if starting from scratch, but it is very likely to succeed if it is asked first to find a solution to a simple task and later to progressively more complex tasks. Let us consider the case of predators and prey. At the beginning of the evolutionary process, both predators and prey display, on average, very poor behaviors. Therefore, for some predators it might be relatively easy to catch several preys and similar situation holds for some prey. Gradually, both populations and their evolving challenges will become progressively more and more complex.

Another potentially beneficial factor regards the variety of tasks faced by every single individual. During its lifetime, an organism might encounter a certain number of opponents both from the same generation and from the earlier generations. As a result, the ability for which individuals are selected is more general. Finally, competing coevolutionary systems are computationally appealing as the ever changing fitness landscape caused by changes in the co-evolving species is potentially useful in preventing stagnation in local minima.

3.2.2 Limitations

Competitive coevolution has a number of drawbacks. Competition is not always the major parameter to produce better coevolutionary results.

First, a continuous increase in complexity is not guaranteed. Co-evolving population might in fact drive one another along twisting pathways where each new solution is just good enough to counter balance the current strategies implemented by the co-evolving population, but is not necessarily more complex than solutions discovered some generations earlier. Moreover, the new solution may be ineffective against strategies which previous solutions were able to defeat. Consider the following example.

Here A strategies are for predators and B strategies are for prey. It is found that the predator population is generating A_1 and A_2 strategies and on the other hand, the prey population is generating the B_1 and B_2 strategies which are just good enough to counter balance the current strategies of the opponent population. Thus a cycle of alternative classes of strategies are formed and the net progress is in fact stuck in this infinite cycle.

$$A_1 > B_1$$
$$B_2 > A_1$$
$$A_2 > B_2$$
$$B_1 > A_2$$

A_1: Track prey and try to catch it by approaching it.

B_1: Stay Still or hidden near a wall

A_2: Track prey while remaining more or less in same place

B_2: Move fast in the environment avoiding predators and wall

A_1, A_2: predator strategies; B_1, B_2: prey strategies

Figure 3.1 Competitive approach forming a cycle of alternative strategies

Another, major problem is that the quality of competitor may hamper the overall coevolutionary process. If the competitor is very weak or inefficient, competitive coevolution will generate very poor or suboptimal solution. Finally, in competitive coevolution, it is very difficult to monitor the progress of competing species since it is dependent on the fitness of other species. This is sometimes referred to as the *Red Queen effect* (Van Valen 1973, Ridley 1993) [5, 38]

3.3 Cooperative Coevolution

Cooperative coevolution requires that a number of individuals work together to solve a common problem. It refers to the simultaneous evolution of two or more species with coupled fitness. Such coupled evolution favors the discovery of complex solutions whenever complex solutions are required and widen the range of applications of evolutionary computation. Examples of cooperative coevolution are evolving robotic soccer teams, evolving a group of predators, or evolving two separate species that must work together to solve a problem. Cooperation implies communication; this communication can be explicit or implicit.

It should be noted that cooperative coevolution can be used in those kinds of problems, which can be naturally modularized into subcomponents that interact or cooperate to solve the problem. Each subcomponent can then be evolved in its own population, and each population contributes its best individual to the solution. The fitness of an individual depends on its ability to collaborate with individuals from other species. In this way, the evolutionary pressure stemming from the difficulty of the problem favors the development of cooperative strategies and individuals [39].

3.3.1 Benefits of Cooperative Coevolution

There are a number of advantages of cooperative coevolution.

First, Cooperative coevolutionary algorithms involve a number of independently evolving species which together form complex structure, well suited to solve difficult problems. Some difficult problems can only be solved by cooperating agents, not competitive agents. In competitive coevolution, a common phenomenon is that progress becomes stuck at some level if the competing agent is inefficient. This phenomenon is very rare in cooperative coevolution since the cooperating agents always try to improve the overall performance of the team.

3.4 Major Design Issues of Cooperative Coevolution

There are some major issues that need to be addressed in designing a cooperative model. These are the issues of problem decomposition, interdependencies between subcomponents, credit assignment problem, maintenance of diversity, considering the role and cost of communication and parallel processing.

3.4.1 Problem Decomposition

Problem decomposition consists of determining an appropriate number of subcomponents and the role each will play. For some problems an appropriate decomposition may be known *a priori*. Consider the problem of optimizing a function of k independent variables. It may be reasonable to decompose the problem into k subtasks, with each assigned to the optimization of a single variable. However, there are many problems for which we have little or no information pertaining to the number or roles of subcomponents that ideally should be in the decomposition. For example, if the task is to learn classification rules for a multimodal concept from pre-classified examples, we probably will not know beforehand how many rules will be required to cover the examples or which modality of the concept each rule should respond to. Given that an appropriate decomposition is not always obvious, it is extremely important that the EA addresses the decomposition task either as an explicit component of the algorithm or as an emergent property.

3.4.2 Inter Dependency of Subcomponents

The second issue concerns the evolution of interdependent subcomponents. If a problem can be decomposed into subcomponents without interdependencies, clearly each can be evolved without regard to the others. Graphically, one can imagine each subcomponent evolving to achieve a higher position on its own fitness landscape, disjoint from the fitness landscapes of the other subcomponents. Unfortunately, many problems can only be decomposed into subcomponents exhibiting complex interdependencies. The effect of changing one of these interdependent subcomponents is sometimes described as a deforming or warping of the fitness landscapes associated with each of the other interdependent subcomponents (Kauffman and Johnsen 1991) [41]. Given that EAs are adaptive, it would seem that they would be well suited to the dynamics of these coupled landscapes. However, natural selection will only result in co-adaptation in such an environment if the interaction between subcomponents is modeled.

3.4.3 Credit Assignment Problem

The third issue is the determination of the contribution each subcomponent is making to the solution as a whole. This is called the *credit assignment* problem, and it can be traced back to early attempts to apply machine learning to the game of checkers by Arthur Samuel (1959) [6]. For example, if we are given a set of rules for playing a two-player game such as checkers, we can evaluate the fitness of the rule set as a whole by letting it play actual games against alternative rule sets or human opponents while keeping track of how often it wins. However, it is far from obvious how much credit a single rule within the rule set should receive given a win, or how much blame the rule should accept given a loss.

One of the fundamental principles of Darwinian evolution is that the likelihood of an individual successfully passing its characteristics on to future generations is based on the fitness of the individual. If our goal is to use a cooperative model of multi-agent systems, there must be a process by which credit or blame is assigned to each of the agents for their role in the overall task. That is, the agents need to be evaluated based on their contribution to the problem solving effort as a whole.

3.4.4 Maintenance of Diversity

The fourth issue is the maintenance of diversity in the population. If one is using an EA to find a single individual representing a complete solution to a problem, diversity only needs to be preserved in the population long enough to perform a reasonable *exploration* of the search space. Typically, once a good solution is found, the EA terminates and all but the best individual is discarded. Contrast this to evolving a solution consisting of co-adapted subcomponents. In the coevolutionary paradigm, although some subcomponents may be stronger than others with respect to their contribution to the solution as a whole, all subcomponents are required to be present in the final solution.

There is a continuous pressure in an evolutionary algorithm driving the population to convergence. This results from the Darwinian principle of natural selection, in which the more highly fit individuals produce a greater number of offspring than the less fit individuals. With each new generation, the average fitness of the population will rise, making above average individuals increasingly exclusive. This, along with the effects of genetic drift, will eventually lead to a population consisting mostly of clones of a single highly fit individual.

3.4.5 Role and Cost of Communication

In a cooperative task, the agents trying to achieve a common objective must communicate with others. The role of communication in cooperative behavior has been studied in several Artificial Life experiments [24, 25]. These studies have shown that communication can be highly beneficial, allowing the communicating individuals to outperform the non-communicating ones. However, most of these studies did not take into account the cost of communication- such as, the danger of attracting the opponents, the energy required for signaling, the means of communication, the equipment required etc.

3.4.6 Parallelism

Although parallel computing in the sense of utilizing multiple processors is not necessary for the evolution of co-adapted agents, it becomes a critical issue if we apply our algorithm to the solution of increasingly difficult problems. Because evolutionary algorithms try to evolve a population of solutions rather than a single solution, parallelism is required to maintain a reasonable evolution time. Fitness can be computed parallelly on separate processors. Moreover, each of the agents can be evolved separately in different processors in order to have greater parallelism. However, some amount of inter-processor communication is required because of the interdependencies among the agents.

Chapter 4
The Cooperative Coevolutionary Model

4.1 Introduction

In this chapter, we present the new model of cooperative coevolution of multi-agent system (CCMAS). This model has tried to address major limitations of traditional evolutionary algorithms such as the issues of problem decomposition, credit assignment to the cooperating agents, maintaining interdependencies among the agents, keeping the population diversified, considering the cost of communication, making it parameter independent etc by giving suitable solutions to these problems. In order to justify the efficacy and accuracy of the model, we have applied the model in *predator-prey problem* which is a very complex task that is yet to be solved. The variation of *predator-prey problem* on which CCMAS model has been applied is quite different and much difficult from the existing ones. The details of our implementation are also pointed out here.

4.2 Key Features of the Model

The proposed model of cooperative coevolution of multi-agent system (CCMAS) is a generalized model which is found to be quite robust and efficient in predator-prey domain. The model has the following features:

4.2.1 Problem Decomposition

First, the CCMAS model addresses the major issue of problem decomposition. For solving a complex problem, one has to determine the number of sub-problems and the precise role of each of them. In some complex task, appropriate decomposition (number of subcomponents) is known a priori. But in other cases, exact number or the role of the subcomponents are not known precisely. Good decomposition has a selective advantage over poor decomposition.

In our model of multi-agent system, problem decomposition is done by keeping several agents. The role is determined by optimizing the state of one agent while keeping the state of other agents fixed. Thus a single complex task is being converted into N significantly easier subtasks.

4.2.2 Maintaining Interdependencies

Second, the CCMAS model has the property of maintaining interdependencies among the agents. If a problem can be decomposed into independent subcomponents, each can be solved without any regard to the others. But unfortunately, many problems can only be decomposed into subcomponents with complex interdependencies. The fitness landscape of such subcomponent is not only related to its own phenotypes (or genotypes), but also linked to some other subcomponents. So each subcomponent must constantly adopt just to remain in parity with others.

In our model of multi-agent system, subcomponents (that is, agents) are not totally independent. Optimizing the state (or fitness) of one agent may hamper other agents' states (or fitness). Interdependencies among the agents must be maintained. So each agent must adopt itself in every step.

Moreover, in order to model the method of interaction among the agents (or subcomponents), current best individual from each agent (or subcomponent) is used to interact with other agents. This implies that all the individuals of one agent (subcomponent) are to be evaluated with the context of the current best individuals of each of the other agents (or subcomponents). The benefit of choosing such approach is that it is easier, simpler and requires less time to evaluate since minimal number of combinations is to be checked.

4.2.3 Maintaining Diversity in the Population

The CCMAS model has the property of maintaining diversity in the population. For a simple single-agent system, population diversity is only required to explore all the regions of the search space. But for a complex problem with multiple agents, it may not be possible to find a solution without proper population diversity.

In our CCMAS model, population diversity is to be maintained by keeping separate populations for different agents with reasonable population size and using standard genetic operators.

4.2.4 Assigning Credit to the Agents

The CCMAS model also gives a suitable solution to the credit assignment problem in multi-agent system. The contribution of each agent to the partial solution of the problem is termed as credit assignment problem. For a simple task (e.g., a single agent system), credit assignment is very straight-forward. But for a complex task involving several agents, important crucial decisions have to be made about how much credit an agent should get for a success or how much blame to be given for a failure.

In our cooperative coevolutionary model of multi-agent system, credit assignment is done at two different levels:

- Local credit assignment
- Global credit assignment

Local credit is assigned for each individual in an agent considering the improvement (of fitness) it have made from its previous state. This credit value helps in making the decision of selecting the current best individual of that agent which is to be used for reproduction. Local credit assignment is done considering the context of other agent to be constant.

Global credit assignment, as the name implies, occurs at the upper level for all the cooperating agents. It is based on the estimate of the overall improvement (of fitness) or progress the agents have made towards the goal (or towards solving the problem). These credit values play an important role in selection scheme as well as the replacement scheme.

4.2.5 Role and Cost of Communication

In many cooperative models of multi-agent systems, the role and cost of communication is not considered at all. But communication is very much essential in complex tasks involving multi-agent systems where agents work together to achieve a common goal. Without communication, most complex task may not be solved at all. In our CCMAS model, the main role of communication is to generate better results by the agents. Moreover, in order to consider the cost of communication, some amount of noise is introduced to the fitness landscape of the communicating agents while they communicate with other.

4.2.6 Parallel Processing

The CCMAS model can facilitate parallel computation to speed up the coevolution process. Since evolutionary algorithms try to evolve a **population of solutions** rather than a single solution, parallelism is required to maintain a reasonable evolution time. Fitness can be computed parallelly on separate processors. Moreover, each of the agents can be evolved separately in different processors in order to have greater parallelism. However, some amount of inter-processor communication is required because of the interdependencies among the agents.

4.2.7 Accuracy

The success rate of the predator agents to capture the prey is found to be almost 100%, the average number of move to catch the prey is less and less number of generations is required to evolve the predators. Communicating agents are found to perform better than non-communicating agents.

4.2.8 Parameter Independence

The CCMAS model shows good performance irrespective of different system and environment parameters. This is evident from different graphs presented in chapter 5.

4.2.9 Scalability

The model can be scalable to larger problem with dimension of the environment being made two or even four times of the original. This is also evident from the fact that the environment is toroidal. So the prey can escape from any side of the grid to the other side both vertically and horizontally.

4.3 Difference with Other Cooperative Models

There are many coevolutionary models [24, 25] that did not take care of cost of communication which may be needed by the cooperative agents. But our CCMAS model always pays attention to this overhead cost. Some models [1, 9] require hand-decomposition of the complex problem. But in our CCMAS model, the number of subcomponents is trivial, that is, the number of cooperative agents.

In summary, no single model addresses all the issues regarding multi-agent systems. Our CCMAS model tries to do so and produces better result. Our study uses an implementation that is quite different and more difficult from all the existing works. The main differences are pointed out here.

- In our configuration, all agents have the same speed. The predators do not usually move faster than the prey (unlike Hayes and Sen's model [31].
- Neither the Predators nor the prey stop moving in the whole cycle. (unlike [31] where prey does not move 10% of the time; this effectively makes the predators travel faster than the prey]
- All agents (both the predators and the prey) are allowed to move in both orthogonal as well as diagonal directions unlike many other studies. [Jim and Giles [30].
- All the agents move simultaneously; the agents do not take turns (like Stephens and Merx' model) [35]. So conflict resolution is necessary since the predators are not allowed to occupy the same position at the same time unlike (Benda et. al.) [28] (Javier A. Alcazar) [42] where the predators can occupy same position.
- The "arena" or "world" is two-dimensional toroidal which is not the case in Javier's model [42] where predators and preys bounce off the wall using the reflection angle law: angle of incidence is equal to angle of reflection. This toroidal property (both in vertical and horizontal direction) makes the arena as an infinite region.
- The vision powers of the predators and the prey are kept equal and limited. That is, a predator can only see the prey if it is within its visual range (which is kept as a system parameter). Similar situation holds for prey. So local perception is allowed by all the agents. This configuration is quite different from other models (Hayes

and Sen [31], Korf [32]) where a predator can always see the prey and the prey can see all the predators. This property is inconsistent with biological systems since no creature has infinite sensing range.

- When one of the predators sees the prey, it informs other predators about the position of the prey which is not the case in most of the studies (e.g. in Hayes and Sen [31], two predators cannot communicate to resolve conflicts or negotiate a capture strategy).
- Finally in our configuration, there exists multiple predator and single prey unlike some study (Javier A. Alcazar) [42] where several preys exist in the environment.

4.4 The CCMAS Model

This section explains the generalized model of cooperative coevolutionary algorithm for multi-agent systems (CCMAS). First, the steps are explained in brief. The next section explains it in details.

```
Step 1. Initialize different parameters from the input file.
Step 2. Set a counter value to zero.
Step 3. Initialize the population zero for prey.
Step 4. Initialize the population zero of each of the predators.
Step 5. Evaluate the fitness of the individuals of each of the populations.
Step 6. Assign credit locally and globally.
Step 7. Select individuals based on fitness from each population for reproduction.
Step 8. Apply crossover and mutation on these selected individuals to produce offspring.
Step 9. Evaluate the fitness of the offspring.
Step 10. Select individuals from the old generation to be replaced by the newly generated offspring.
Step 11. If the goal is reached, go to step 12. Otherwise, go to step 5.
Step 12.Repeat the cycle for a maximum number of iterations. If counter reaches the maximum value, terminate the program. Otherwise, go to step 3.
```

Algorithm 4.1: Steps of cooperative coevolution of multi-agent systems (CCMAS)

Step 1. *Initialize different parameters from the input file:*

In the first step, all the useful environment parameters as well as the system parameters are to be set from the input file. These parameters play a vital role in the result of a program run. These parameters are kept same for different program run of the model. But for the comparison with other strategies, the environment and the system parameters are kept similar with those strategies.

Here is a listing of these parameters.

- Maximum number of trials in each program run
- Population size of prey
- Population size of predator
- The length of the toroidal environment
- The width of the toroidal environment
- Random Initial position vs Fixed initial position
- Vision power of Prey
- Vision Power of each Predator
- Predator speed and Prey Speed
- No of Predators
- Mutation probability
- Maximum number of moves allowed to catch the prey

Step 2. *Set a counter value to zero.*

A counter is used to count the number of trials (which is a system parameter). It is initialized to zero. The program is usually run for a maximum number of trials (usually 1000).

Step 3. *Initialize the population zero for prey.*

The population zero of prey is initialized in this state. An individual can be initialized in two methods: random initialization and fixed initialization. Two things are to be initialized in every case:

- Position of the prey (x, y coordinate)
- Direction of movement

The details of the prey initialization process are described in the section 4.5.2.

Step 4. *Initialize the population zero of each of the predators.*

The population zero of each of the predators is initialized in this state. Like the prey, an individual for a predator can be initialized in two methods: random initialization and fixed initialization. Again two things are to be initialized in every case:

- Position of the predator (x, y coordinate)
- Direction of movement

The details of the predator initialization process are described in the section 4.5.2.

Step 5. *Evaluate the fitness of the individuals of each of the populations.*

In order to evaluate how good or bad is the current state of a predator or a prey, a fitness function is used. For the predators, the average Euclidean distance from each predator to the prey is computed first; the lower the average distance, the higher is the fitness value. For the prey, it is just the opposite. The higher the average distance, the higher is the fitness value.

$$d_{avg} = \sum_{i=1}^{N} \frac{di}{N} \qquad \text{..................(4.1)}$$

Where di is the Euclidean distance between prey and the ith predator and N is the number of predators. More details on the Fitness function is described in the section 4.5.3

Step 6. *Assign credit locally and globally.*

Local credit is assigned for each individual in an agent considering the improvement (of fitness) it have made from its previous state. This credit value helps in making the decision of selecting the current best individual of that agent which is to be used for reproduction. Local credit assignment is done considering the context of other agent constant.

Global credit assignment, as the name implies, occurs at the upper level for all the cooperating agents. It is based on the estimate of the overall improvement (of fitness) or progress the agents have made towards the goal (or towards solving the problem). Global credit is same for all the cooperating agents.

Step 7. *Select individuals based on fitness from each population for reproduction.*
The best individuals from each of the current populations are selected based the highest fitness value. From the whole population, a fraction (may be 20%) of individuals are selected. The details of the selection process are described in the section 4.5.4.

Step 8. *Apply crossover and mutation on these selected individuals to produce offspring.*
In order to produce offspring from the selected individuals, genetic operators such as crossover and mutation are used. So the offspring is a variation of the best individual selected. The details of the crossover and mutation processes are described in the section 4.5.5 and 4.5.6 respectively.

Step 9. *Evaluate the fitness of the offspring.*
The fitness of the offspring is computed in this step using the evaluation function mentioned before.

Step 10. *Select individuals from the old generation to be replaced by the newly generated offspring.*
In this step, the offspring produced replace few worse (may be 20%) individuals in the current population. Thus the new population for predators and the prey is formed.

Step 11. *If the goal is reached, go to step 12. Otherwise, go to step 5.*
In this step, the goal condition is checked. If the goal is reached, the next epoch is started using random initialization of the predators and the prey populations. Otherwise, this current epoch of coevolution continues and the program control goes to step 5

Step 12.*Repeat the cycle for a maximum number of iterations. If counter reaches the maximum value, terminate the program. Otherwise, go to step 3*
This step checks the counter value whether it has reached the value of maximum number of iterations. If so, the program is terminated. Otherwise, the program control goes to step 3 to start the next epoch.

```
t = 0
for each agent A do begin
    Initialize Population $P_t(A)$ to random individuals
    evaluate fitness of each individual in $P_t(A)$
    end
while termination condition is false do begin
    for each agent A do begin
        Apply credit locally and globally
        evaluate fitness of each individual in $P_t(A)$
        select individual for reproduction based on fitness from $P_t(A)$
        apply crossover, mutations to produce offsprings
        evaluate fitness of offspring
        Replace members of $P_t(A)$ with offspring to produce $P_t(A)$
        end
    t = t + 1
    end
```

Algorithm 4.2: Pseudo-code for cooperative coevolution of multi-agent system

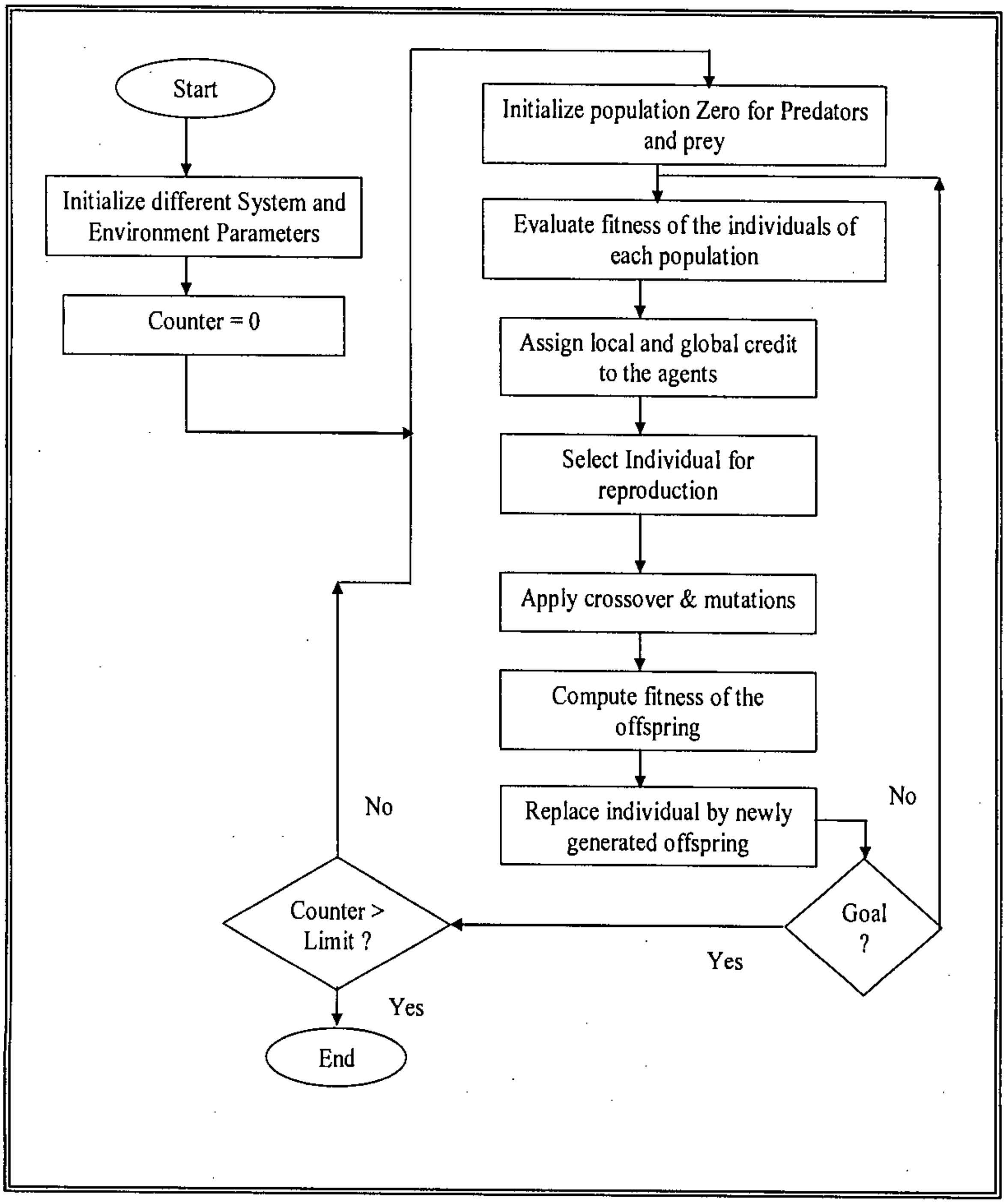

Start
Initialize different System and Environment Parameters
Counter = 0
Initialize population Zero for Predators and prey
Evaluate fitness of the individuals of each population
Assign local and global credit to the agents
Select Individual for reproduction
Apply crossover & mutations
Compute fitness of the offspring
Replace individual by newly generated offspring
No
Goal ?
Yes
No
Counter > Limit ?
Yes
End

Figure 4.1: Flow diagram of the CCMAS model

4.5 Detailed Description of the Model

In this section, we describe the model from experimental point of view, that is, how different steps have been implemented in the proposed model is explained in details.

4.5.1 Parameter Initialization

All the useful environment parameters as well as the system parameters are initialized in this step from the input file. Usually, the grid length and width are set 100 X 100, the number of predators are four (sometimes three), maximum 150 moves allowed to catch the prey, number of directions allowed for predator and prey movement are nine (four orthogonal, four diagonal and one stand still situation), prey and the predator speed are kept equal (sometimes varied depending on the requirement), equal vision power set for predators and prey (usually five), mutation probability is set 0.1, usually 20% of the population are selected for reproduction, maximum 1000 trials in each program run.

4.5.2 Population Initialization

The population zero of predators and the prey is to be initialized in the beginning of each epoch. The population size is usually used 100 for each of the populations.

Prey State Initialization

An individual can be initialized in two methods: random initialization and fixed initialization. Two things are to be initialized in every case:

- Position of the prey (x, y coordinates)
- Direction of movement

In both methods of initialization, direction of prey movement is initialized to one of the nine directions (four orthogonal directions, four diagonal directions and one stand still situation). In the random method of initialization, prey x location and y location is initialized to one of the squares of the 2D toroidal grid environment (Figure 4.3). In the fixed method of initialization, prey is placed in the centre location of the environment, that is, in the coordinate (GRID_LENGTH / 2, GRID_WIDTH / 2) as illustrated in figure 4.2.

Predator State Initialization

For predators, two things (position and direction of movement) are to be initialized. Again it is done in two methods: random initialization and fixed initialization. Like prey, in both methods of initialization, direction of predator movement is initialized to one of the nine directions (four orthogonal directions, four diagonal directions and one stand still situation). In the random method of initialization, predator x location and y location is initialized to one of the squares of the 2D toroidal grid environment (figure 4.3). In the fixed method of initialization, illustrated in figure 4.2, the four predators are placed in the four corner locations of the environment, that is, in the following four coordinates:

- (1,1)
- (1, GRID_LENGTH)
- (GRID_WIDTH ,1)
- (GRID_LENGTH, GRID_WIDTH)

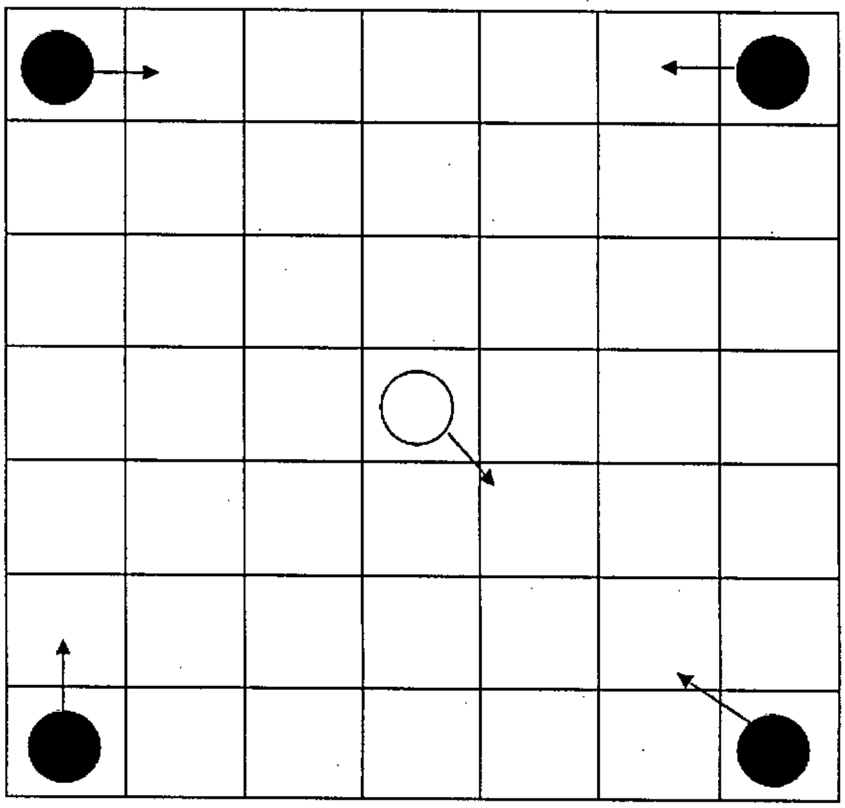


Figure 4.2: Fixed initial state

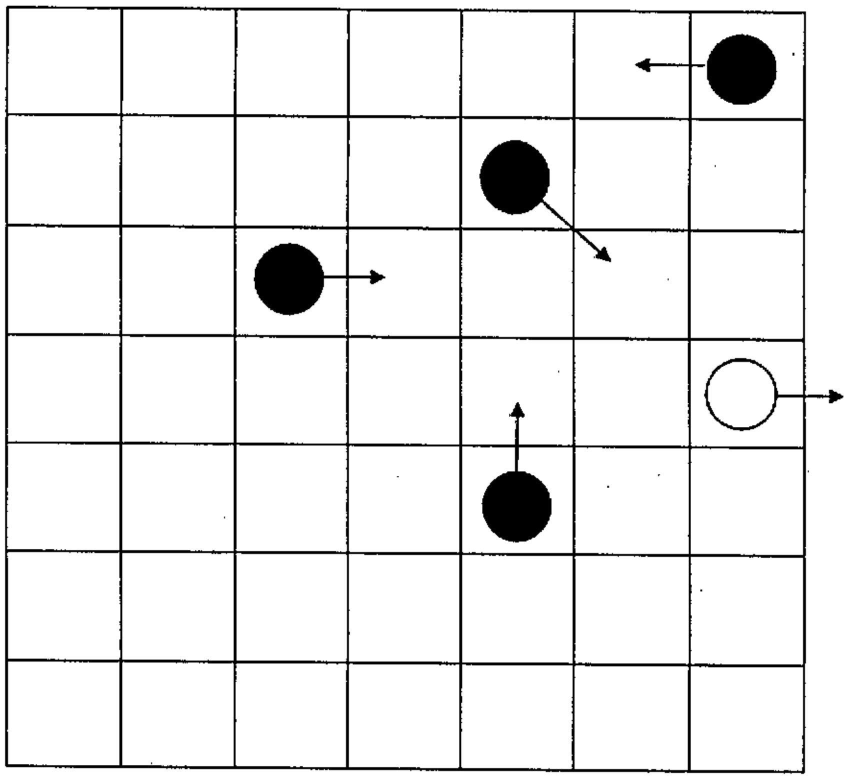


Figure 4.3: Random initial state

4.5.3 Fitness Functions

Fitness or overall success rate can be calculated by considering the number of successful trial to catch the prey using the following formula:

$$CR = \frac{N_s}{N_t} \quad \text{..................(4.2)}$$

Where CR = Capture rate (sometimes termed as success rate),

N_s = Number of successful capture of prey,

N_t = Total number of trials

If the total number of moves in catching the prey N_s times is T_m, then average number of moves to catch the prey, M_{avg} is

$$M_{avg} = \frac{T_m}{N_s} \quad \text{..................(4.3)}$$

This M_{avg} is the average move to successfully catch the prey (only considering successful trials).

Let us now consider all the trials, that is, both successful and unsuccessful trials. In that case, the expected number of moves to catch the prey, Em can be calculated by the following formula:

$$Em = CR \times Mavg + (1 - CR) \times 2M\text{max} \quad \cdots\cdots\cdots\cdots\cdots\cdots(4.4)$$

Expected move = success rate × average no of moves + failure rate × 2 × 150

Here (1-CR) is the failure rate to catch the prey and $\mathbf{M_{max}}$ is the maximum allowed number of moves to catch the prey beyond which the trial is considered to be a failure. In a failure epoch, it is assumed that the average number of moves is twice the maximum allowed number of moves (that is, 2Mmax)

4.5.4 Selection of Parents

The selection procedure used in the experiment is *steady-state selection.* In this method, from every generation a few good (with higher fitness) individuals are selected for creating a new offspring. These individuals are selected based on their fitness values. In our experiment, 20% of the whole population is selected as parents to produce new offspring.

4.5.5 Crossover

Crossover is a binary variation operator that is applied between two parents to produce one or two offspring. In our experiment, crossover is done by interchanging the x and y coordinates of selected parents. Thus the new offspring produced is a variation of the two best individuals of the older generation.

4.5.6 Mutation

Mutation is a unary variation operator used on the offspring produced after the crossover process. This operator is applied with a probability p set by the value of mutation probability which is a system parameter. In our experiment, mutation is applied on the direction of movement of the offspring generated by crossover process. The mutation probability is usually set to 0.1 at the initialization step

4.6 Complexity Analysis

The time and the space complexity of our model are analyzed in this section.

4.6.1 Time Complexity

The time needed for the cooperative coevolution (for a single trial) is proportional to number of agents, population size of each agent, number of generation required and the sum of fitness computation time, selection time and crossover and mutation time.

Coevolution time = O (no of agents × population size × no of generation required ×
(fitness computation + selection time + crossover and mutation time))

And the total time required to execute the algorithm 4.1 is:
Total Time required = No of Trials × Maximum move allowed × Coevolution time

4.6.2 Space Complexity

The space required in implementing the model for *predator prey problem* is proportional to the population size of the predators and/or prey, space required to represent each individual in the predator or the prey population, and the sum of the number of predators and the prey.

In our implementation, population size of the predators and the prey are kept equal. The number of prey is kept one.

Let, the Population Size = p
Number of Predators = n1
Number of Prey = n2 = 1
Space required for each individual = e

So, Total space required = (No of predators + No of Prey) × Population size × Space required for each individual
$= (n1+n2) \times p \times e$
$= (n1 + 1) \times p \times e$

Space required for each individual, e = 2 bytes (for storing the (x,y) coordinates) + 1 byte for storing the direction + 4 bytes for control purpose

Chapter 5
Experiment

5.1 Introduction

The model of cooperative coevolution of multi-agent system (CCMAS) was tested on Predator-prey problem. It is a general and well studied multi-agent problem that is yet to be solved. This problem is being used to study phenomena like competitive and cooperative coevolution, multi-agent strategies and multi-agent communication.

5.2 Prey Capture Task

The *predator-prey problem* was introduced in [28] as an example of multi-agent system. An *agent* is a computational mechanism that exhibits a high degree of autonomy, performing actions in its environment based on information (sensors, feedback) received from the environment. A *multi-agent* environment is one in which there is more than one agent, and further, there are constraints on that environment such that agents may not at any given time know *everything* about the world that other agents know.

The *predator-prey problem* comprises of four predator agents whose goal is to capture the prey agent. A prey is captured if one predator is in the same square as the prey (individual capture) or if two or more predators are in squares adjacent to the prey (cooperative capture). The overall objective of the team is to catch the prey as many times as possible. At the beginning of each epoch the predators and prey are randomly placed on different squares. An epoch is a predefined number of steps or turns. In each step or turn, an agent observes the current state, acts upon it and receives a reward. Each epoch continues until either the prey is captured or until k (usually 150) steps have occurred without a capture. One cycle is composed of predefined number of epochs (usually 1000). After a trial (successful or failure), predators and the prey are randomly relocated.

5.2.1 The Environment

The environment for the predator-prey domain is a two-dimensional torus discretized into a 100 × 100 grid. Since the world is toroidal, if an agent runs off the left edge of the grid, it will reappear on the right edge of the grid, and a similar behavior would be observed vertically. This is explained in figure 5.1. Here four predators (represented by red circles) are approaching the prey (represented by green circle) which is in the rightmost cell of

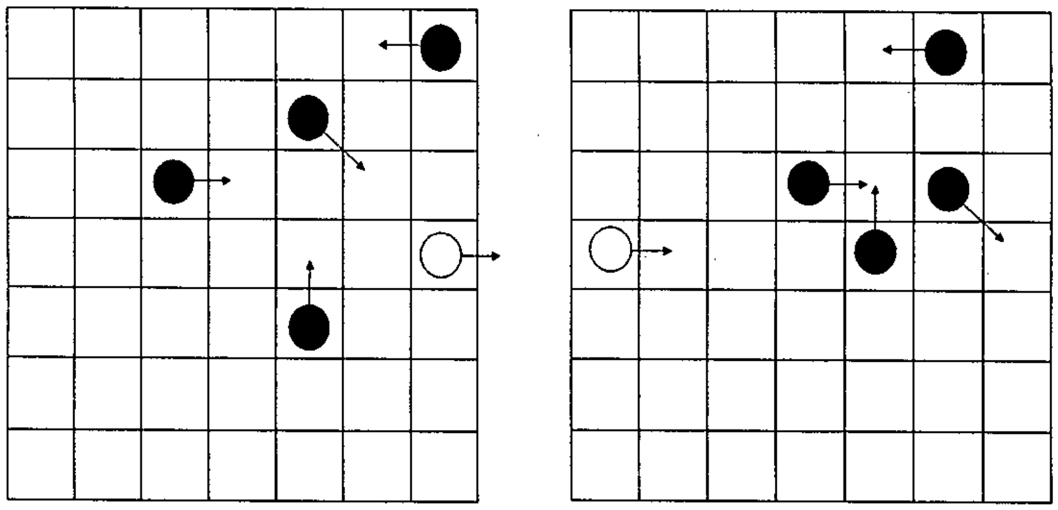

(a) Predators approaching the prey (b) Prey escapes from right to left

Figure 5.1 Toroidal environment

the environment. Due to the toroidal nature of the environment, the prey (in figure 5.1(b)) moves from the rightmost cell to the leftmost cell and is able to escape for the time being.

5.2.2 Directions of Movements

All agents (both the predators and the prey) are allowed to move in both orthogonal as well as diagonal directions. Another choice is the stand still option. So total nine possible options.

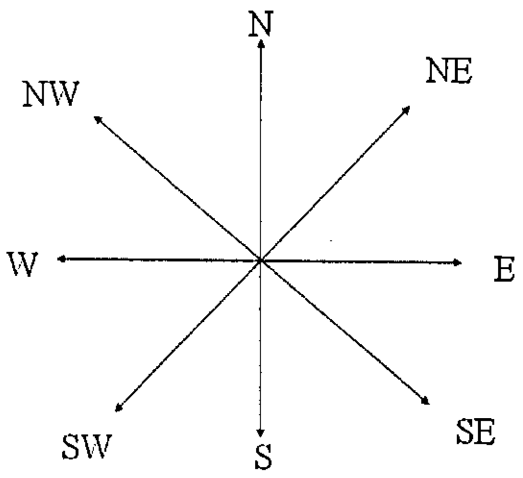
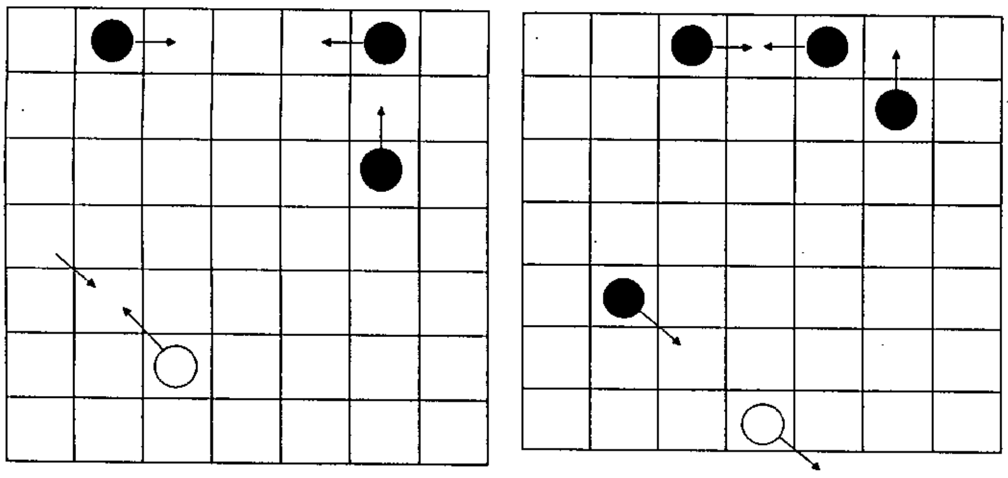


Figure 5.2 Directions of movements

5.2.3 Prey Move Strategy

In our experiment, the main task of the prey is to escape from the moving predators for a specified number of moves (usually 150). So the prey looks around to find out the locations of the predators. There is a limit of the vision power of the prey which is set at the initialization step. If the prey cannot find any predator around it, it moves along its own previous direction. If it finds any predator(s) within its eyesight, it changes its direction of movement opposite to the direction of its nearest predator. If two or more predators are equally close to the prey (within its visual range), the prey picks a random direction and continues to move along that direction even though this may lead to approaching another predator.



(a) Prey sees a predator within its eyesight (b) Prey is moving opposite to that predator

Figure 5.3 Prey move strategy

Two prey strategies are used in the simulations. The ***random* prey** chooses its next direction of movement at each time step randomly from one of the nine choices. The ***linear* prey** picks a random direction at the beginning of a trial and continues in that direction for the duration of the scenario. It has been found that the linear prey is more difficult to capture which was also found in [22, 43] because it does not stay localized in an area. In our simulations, the prey capture task was even more difficult since the prey and predators move at the same speed.

5.2.4 Predator Move Strategy

In our experiment, predator agents cannot move through each other. No two agents are allowed to occupy the same cell at the same time. If two or more agents try to move to the same square, only one of them are allowed to move to that square, others stay still in its older cell for the time being. This phenomenon is explained in figure 5.4. Two predators (represented by red circles) are heading towards the same cell in figure 5.4 (a). Only one of them are allowed to do so; the other predator stay in its previous cell for that move only and after that it continues to move along its own direction (figure 5.4(b)).

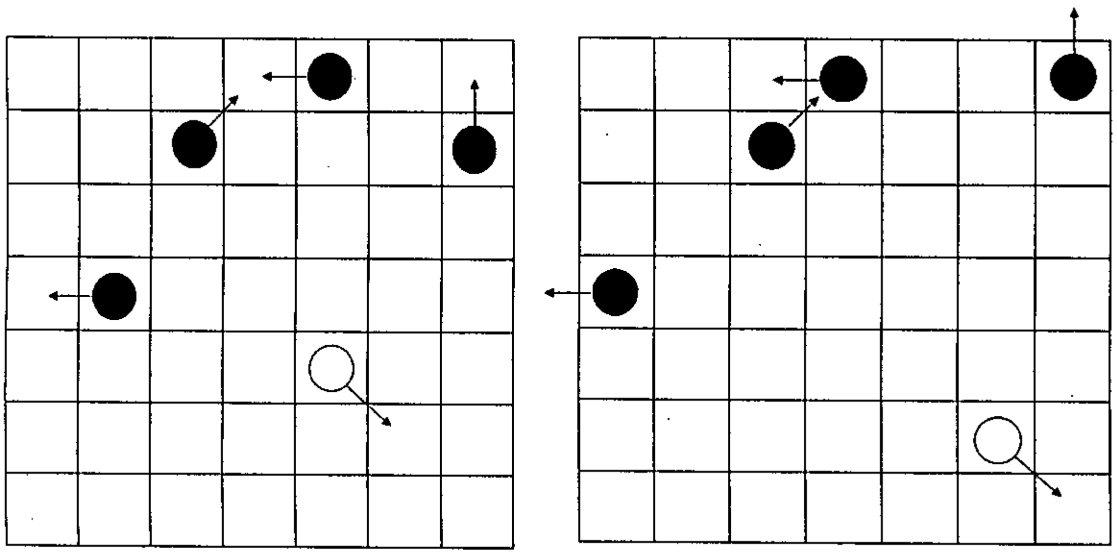

(a) Two predators approaching same cell (b) Only one is allowed, other waits

Figure 5.4 Conflict resolution

Two different strategies are used for predators in the simulation: predators with communication and predators without communication. In the non-communicating strategy, if any predator finds the prey within its eyesight, it does not inform the position of the prey to the other predators (figure 5.5 (a)). No information about the prey position is shared among the predators.. As a result, all the predators move along its own previous direction (figure 5.5 (b)).

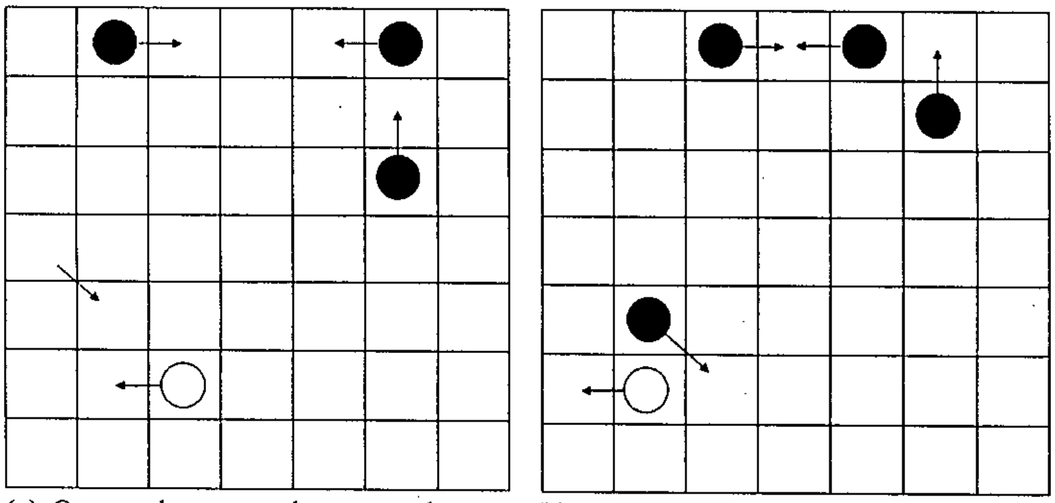
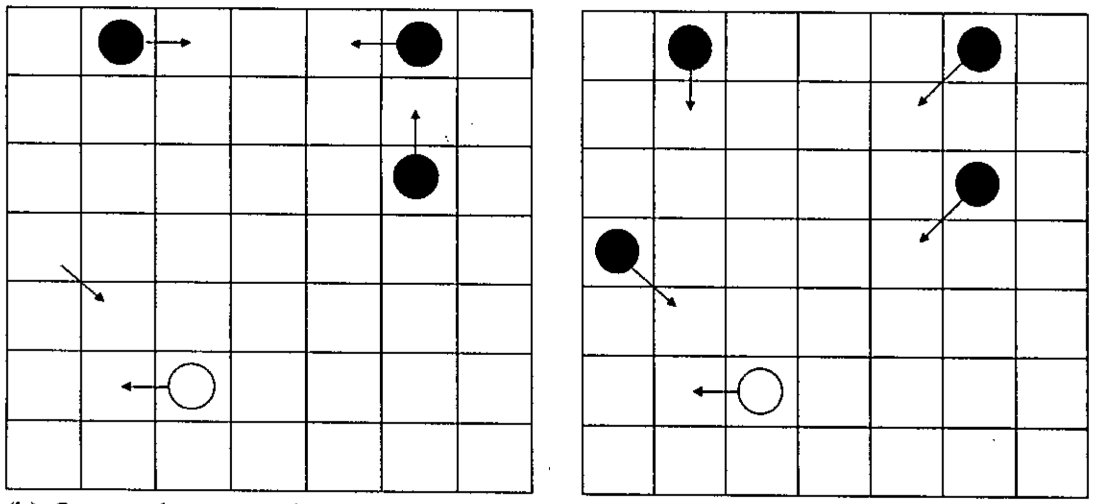

(a) One predator sees the prey; others not informed
(b) So other predators are moving along their previous directions

Figure 5.5 Non-cooperative predators

In the communicating predator strategy, if any predator finds the prey within its eyesight, it immediately informs the position of the prey to the other predators (figure 5.6 (a)). As a result, the other predators change their direction of movement in order to converge towards the prey (figure 5.6 (b)). In order to simulate the cost of communication, some amount of noise (proportional to the distance between these communicating predators) is introduced to their fitness values.



(b) One predator sees the prey; others are informed

(b) So other predators change their directions towards the prey

Figure 5.6 Cooperative predators

For the predators, mutation is applied on their direction of movement with some probability p (usually 0.1) at the beginning. This mutation probability is kept higher at the beginning when no predator has seen the prey so that the predator teams can search and explore different portion of the environment in order to find the prey. Once the prey is seen by any predator (in communicating predator strategy), the mutation probability is gradually reduced to a lower value (usually 0.05).

5.3 System Settings

First the dimension of the grid is to be decided. In our experiment, different grid dimensions are used, (such as, 100 X 100, 150 X 150, 200 X 200 etc.) in which one prey and multiple predators (usually four or three) are to be situated.

Then the initial states of the prey and the predators are to be decided. Two approaches are used: random initial positions and fixed initial positions. Moreover the initial directions of movement for the predators and the prey are also to be set from one of the total nine initial directions of movement. Now the *Euclidean distance* between each of the predators and the prey is calculated using the following formula:

$$d = \sqrt{ \{ (x_1 - x_2)^2 + (y_1 - y_2)^2 \} } \qquad \ldots\ldots\ldots\ldots\ldots\ldots(5.1)$$

In calculating the distance, the toroidal property of the environment is taken into consideration. That is, the minimum of the distances between each (kth) predator and the prey is computed in the **calculate_distance (k)** function.

```
double calculate_distnace(int k){
                double d, d1_x, d2_x, min_x;
                double d1_y, d2_y, min_y;
                d1_x = (predator_x[k] - prey_x)^2;
                d1_y = (predator_y[k] - prey_y)^2;
                d2_x = (GRID_LENGTH-predator_x[k] + prey_x)^2;
                d2_y = (GRID_WIDTH-predator_y[k] + prey_y)^2;
                min_x = (d1_x < d2_x) ? d1_x : d2_x;
                min_y = (d1_y < d2_y) ? d1_y : d2_y;
                d = min_x + min_y;
                d = sqrt(d);
                return d;
}
```

Figure 5.7 Pseudo-code of calculate_distance function

In the simulation, some system and environment parameters are kept similar while some are varied. Usually the maximum moves allowed to catch the prey is set 150, prey and the predator speed are kept equal, equal vision power set for predators and prey (usually five), mutation probability is set 0.1, usually 20% of the population are selected for reproduction, maximum 1000 trials in each program run

5.4 Results

The model is tested varying different system and environment parameters. Success rate, expected number of moves are obtained and plotted to compare different strategies. In figure 5.8, success rate is plotted against generations. In this run (of simulation), speed of predators and prey were kept equal, the vision power were also kept equal, the environment was 100 X 100 toroidal grid, one prey and four predators in the environment and average results of 1000 trials are shown in the graph.

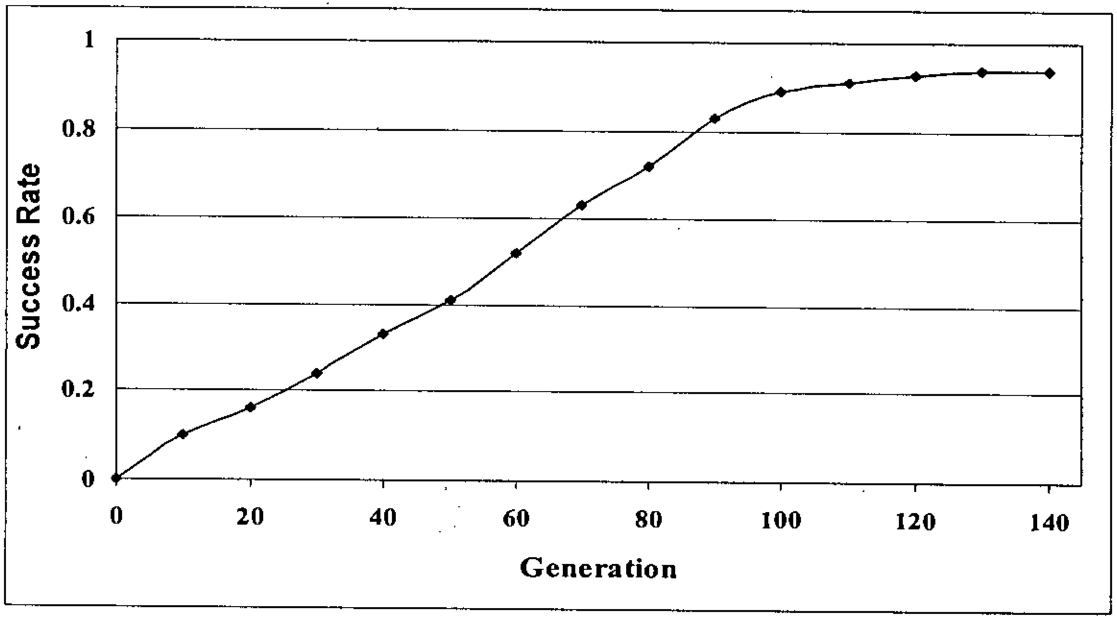


Figure 5.8 Generation vs success rate for CCMAS

In figure 5.9, Expected number of moves is plotted against generations for the same setting as of figure 5.8. Here since the success rate at generation zero is usually zero, the expected number of move at generation zero is showing 300 (which is assumed to be twice the maximum number of moves allowed to capture the prey explained in section 4.5.3).

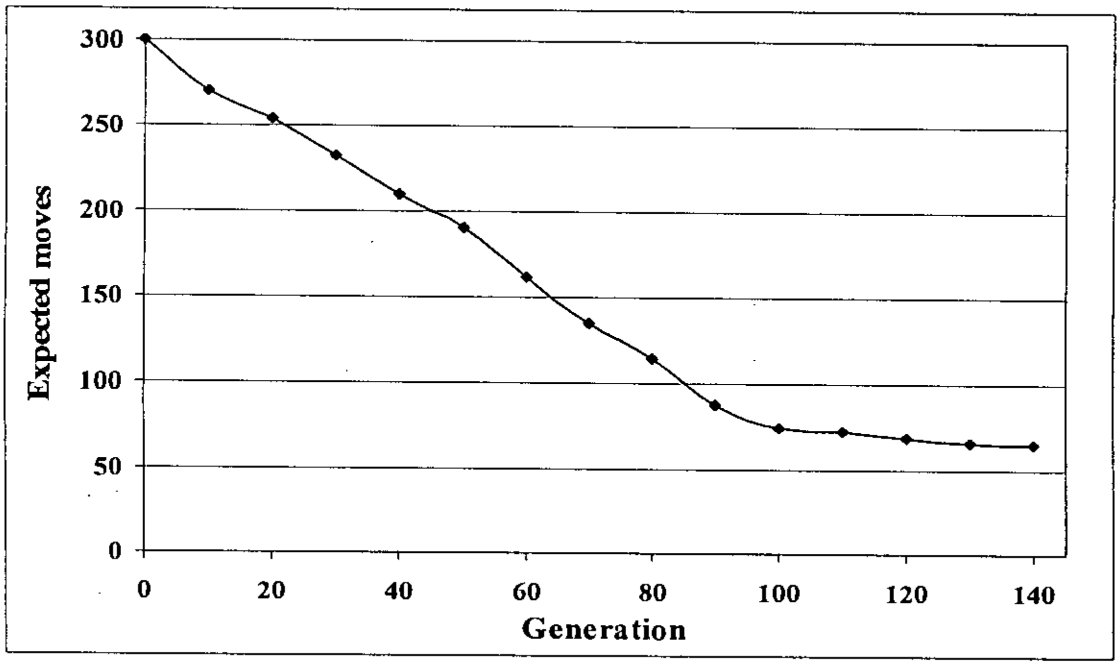


Figure 5.9 Generation vs expected number of moves for CCMAS

In figure 5.10, success rate is plotted against generations for CCMAS and reinforcement learning approach using the same setting as figure 5.8. It is found that CCMAS provides higher success rates than reinforcement learning.

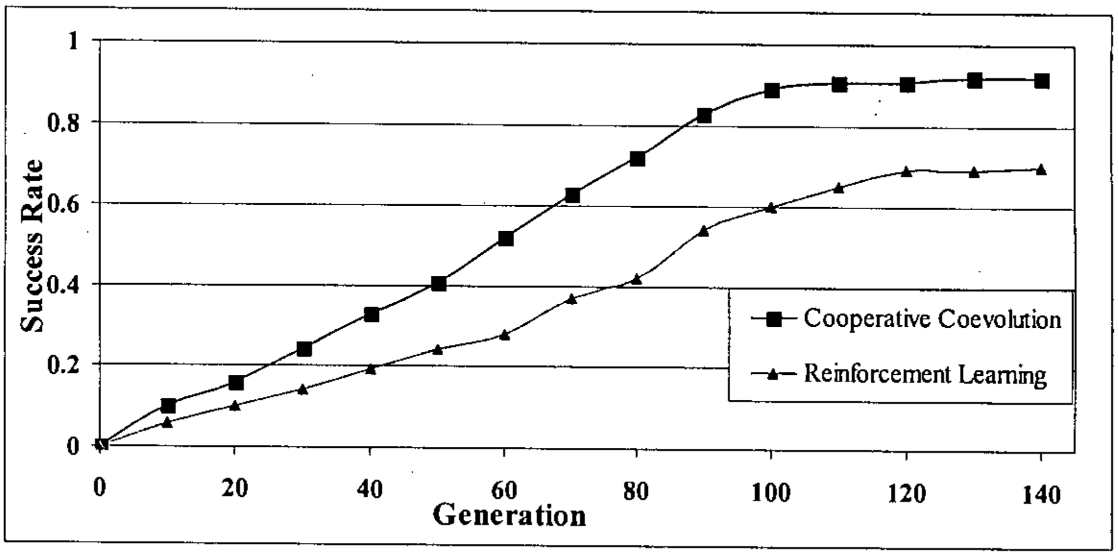


Fig 5.10 Generation vs success rate comparison of two strategies

In figure 5.11, Expected number of moves is plotted against generations for CCMAS and reinforcement learning approach using the same setting as of figure 5.8. Again, CCMAS requires less expected number of moves to capture the prey than reinforcement learning.

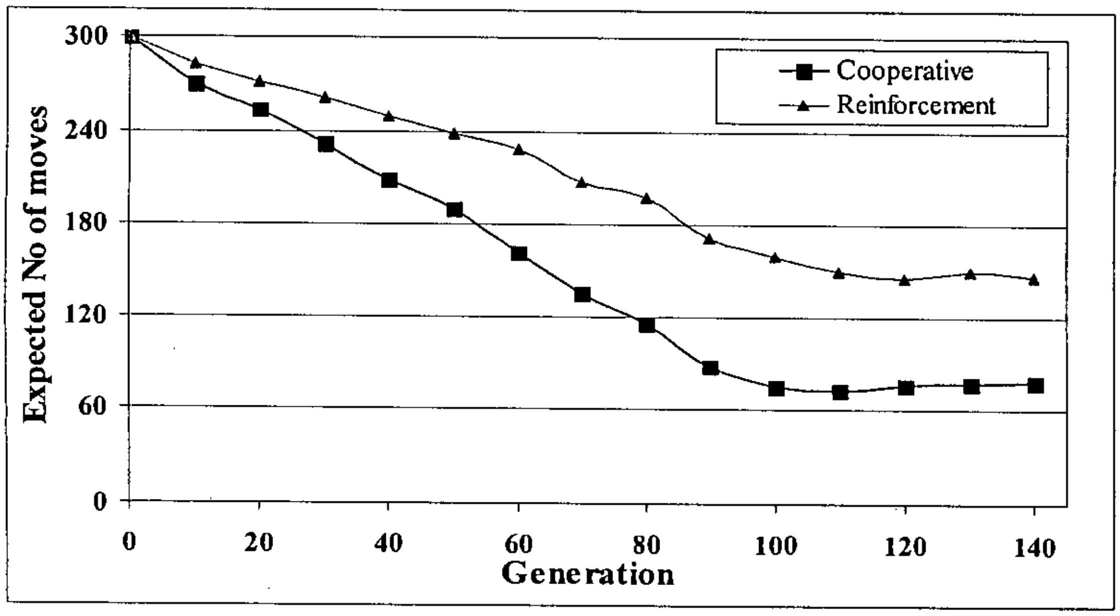


Figure 5.11 Generation vs expected moves comparison of two strategies

In figure 5.12, capture success rates are plotted for different sized environments for the two approaches. It is found that capture rate decreases with the increase of environment size as the prey finds more space to escape and CCAMS shows better success rate.

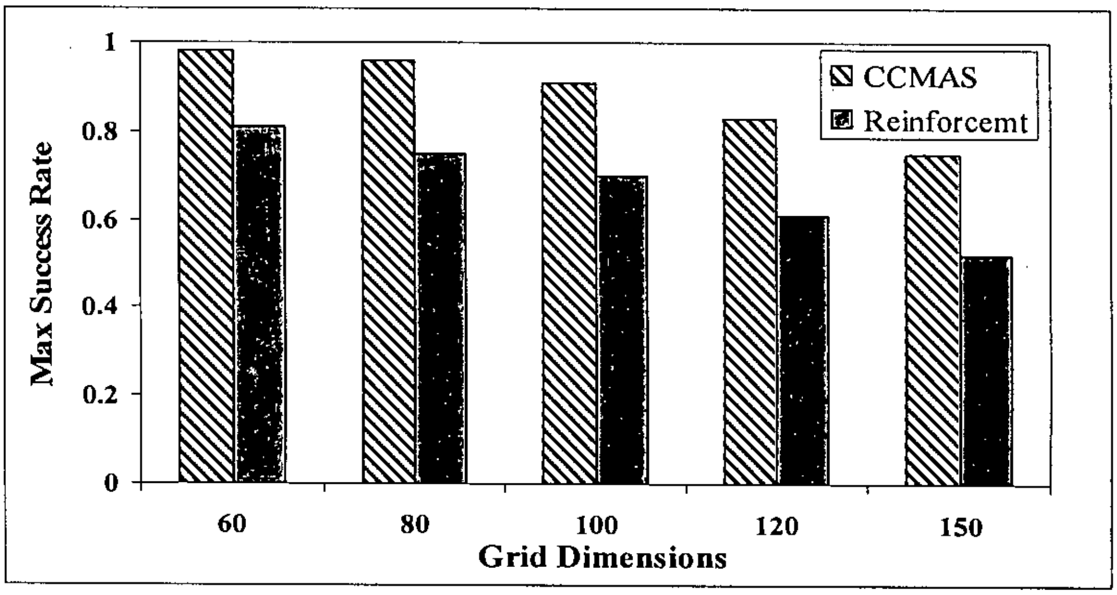


Figure 5.12 Capture rates for different sized environment

In figure 5.13, average capture times (in number of moves) are plotted for different sized environments for the two approaches. It is found that average capture time increases with the increase of environment size.

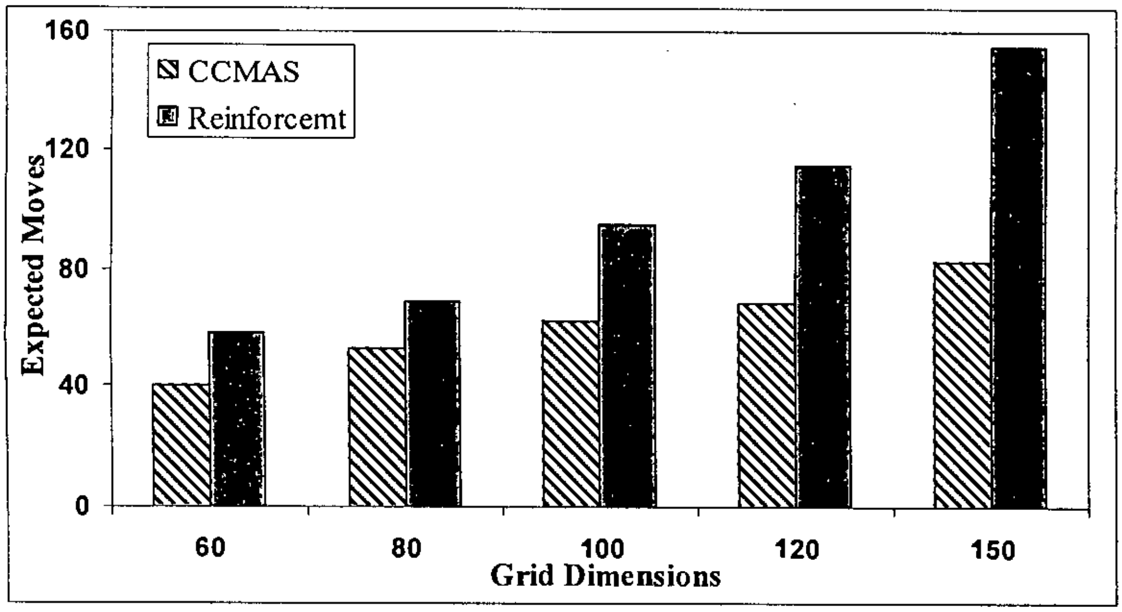


Figure 5.13 Average capture time for different sized environment

In figure 5.14, average success rates are plotted for different predator-prey speeds for the two approaches. It is found that capture rate increases with the increase of predator speed and decreases with the increase of prey speed. Again CCAMS shows better success rate.

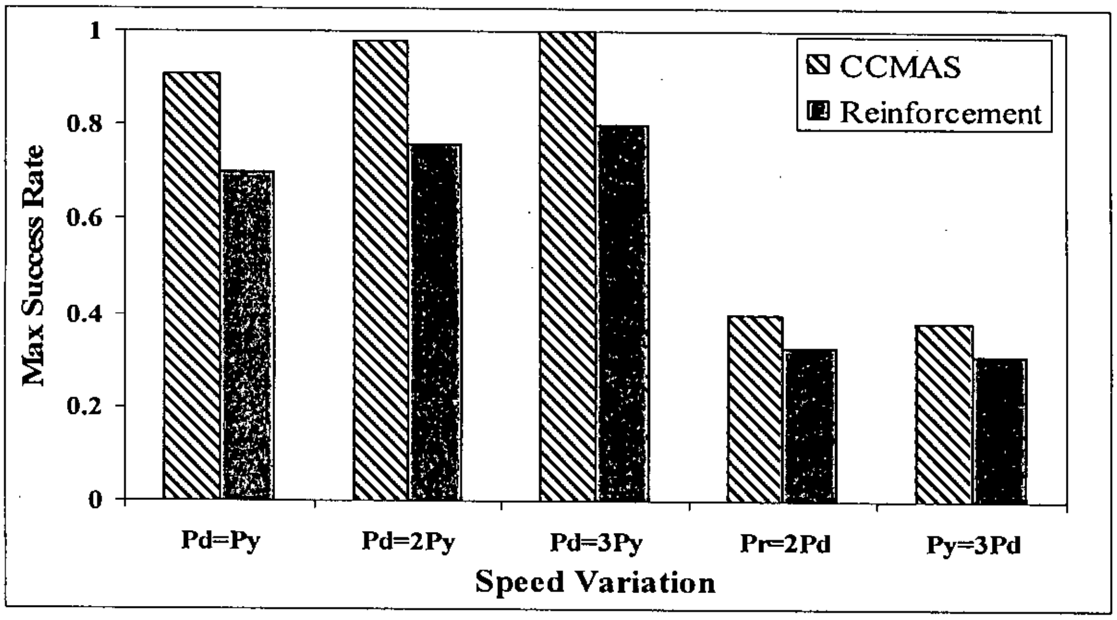


Figure 5.14 Capture rates for different predator-prey speeds

In figure 5.15, expected number of moves is plotted for different predator-prey speeds for the two approaches. It is found that expected moves decreases with the increase of predator speed and increases with the increase of prey speed.

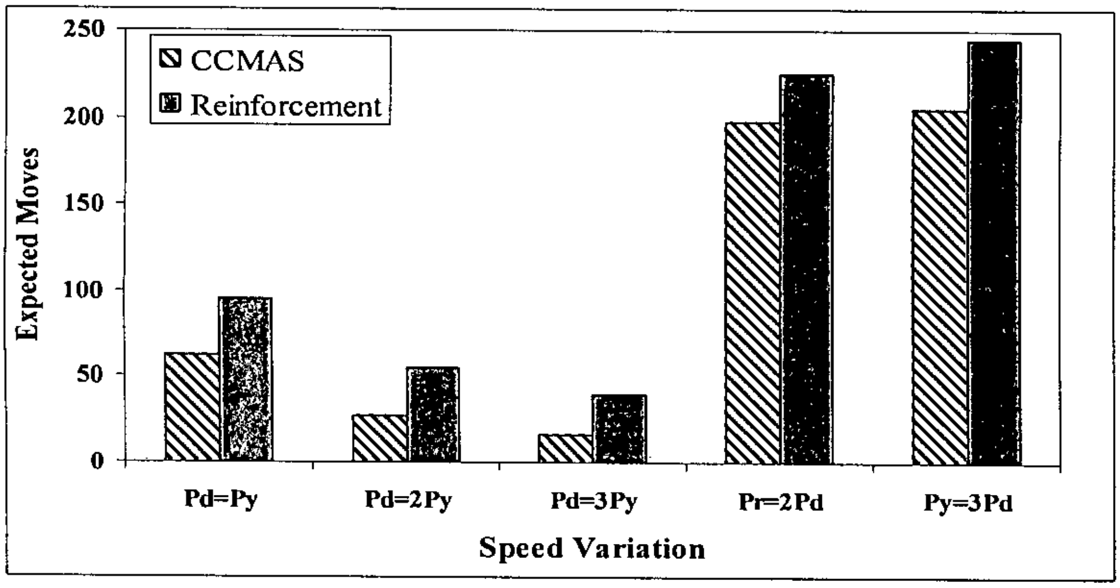


Figure 5.15 Expected number of moves for different predator-prey speeds

In figure 5.16, average success rates are plotted for different predator-prey vision power. It is found that capture rate increases with the increase of predator vision power and decreases with the increase of prey vision power.

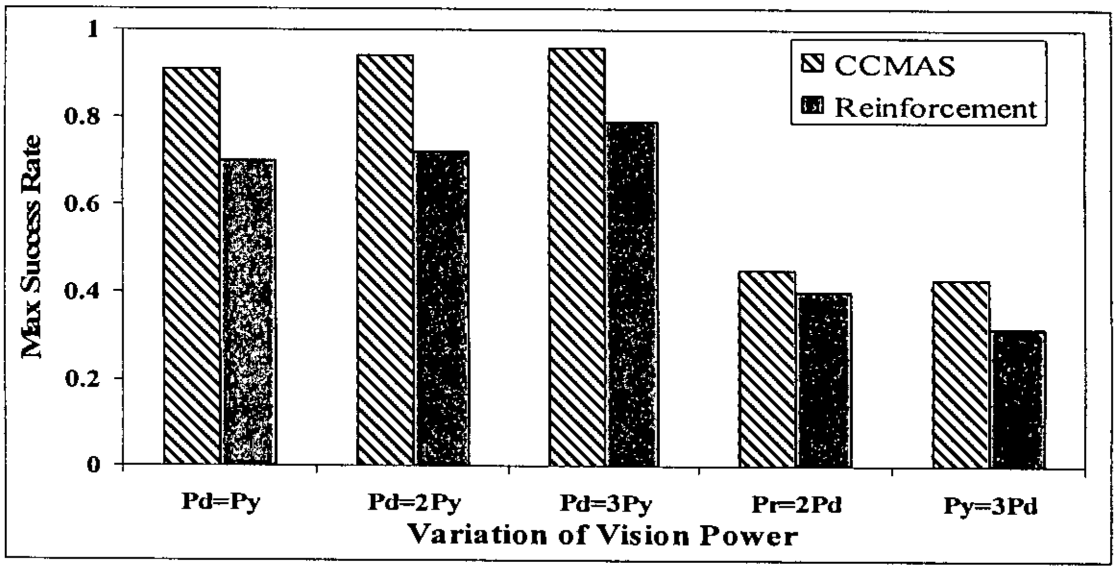


Figure 5.16 Capture rates for different predator-prey vision power

In figure 5.17, expected number of moves is plotted for different predator-prey vision power. It is found that expected moves decreases with the increase of predator eyesight and increases with the increase of prey eyesight.

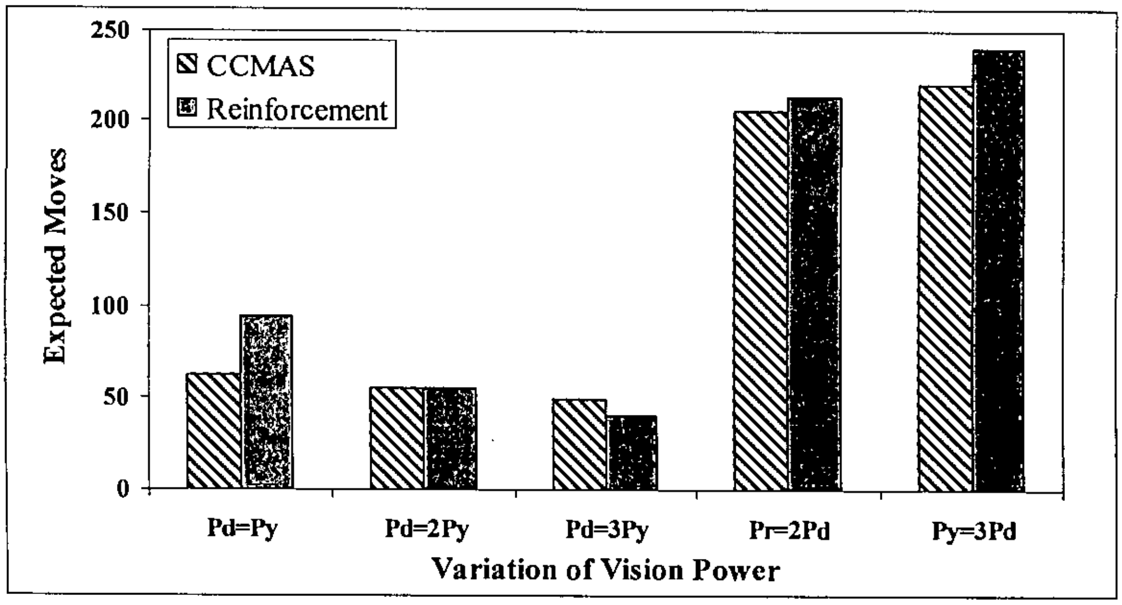


Figure 5.17 Expected number of moves for different predator-prey vision power

In figure 5.18, average success rates are plotted against generation for linear and random prey strategies. It is found that linear prey performs much better than random prey.

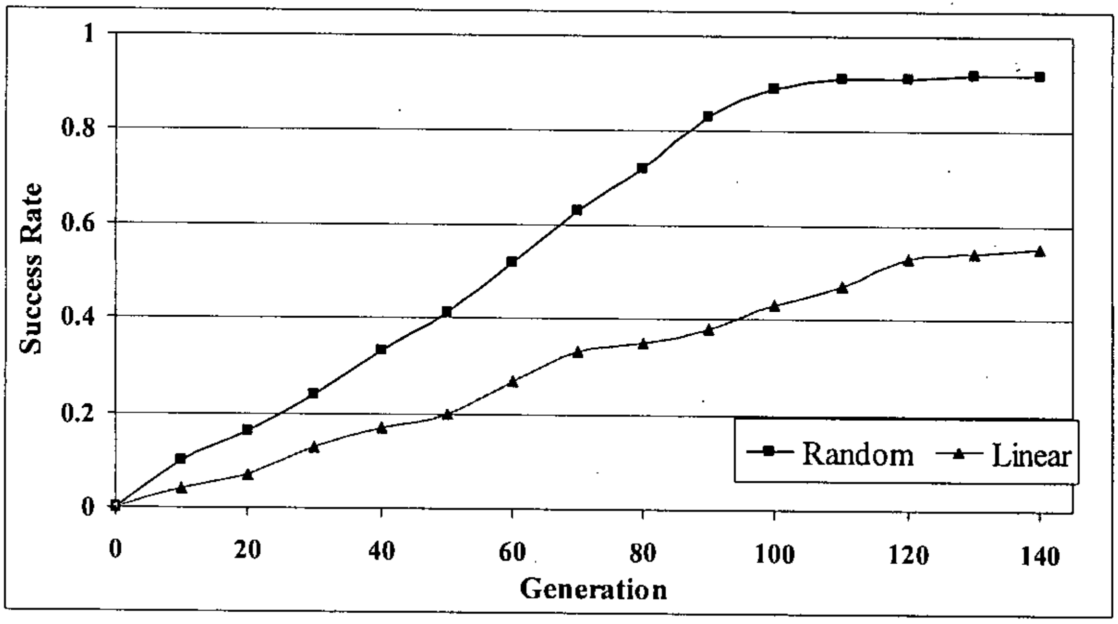


Figure 5.18 Generation vs success rates for linear prey and random prey

In figure 5.19, average number of moves is plotted against generation for linear and random prey strategies. It is found that linear prey requires higher number of moves to be captured than random prey.

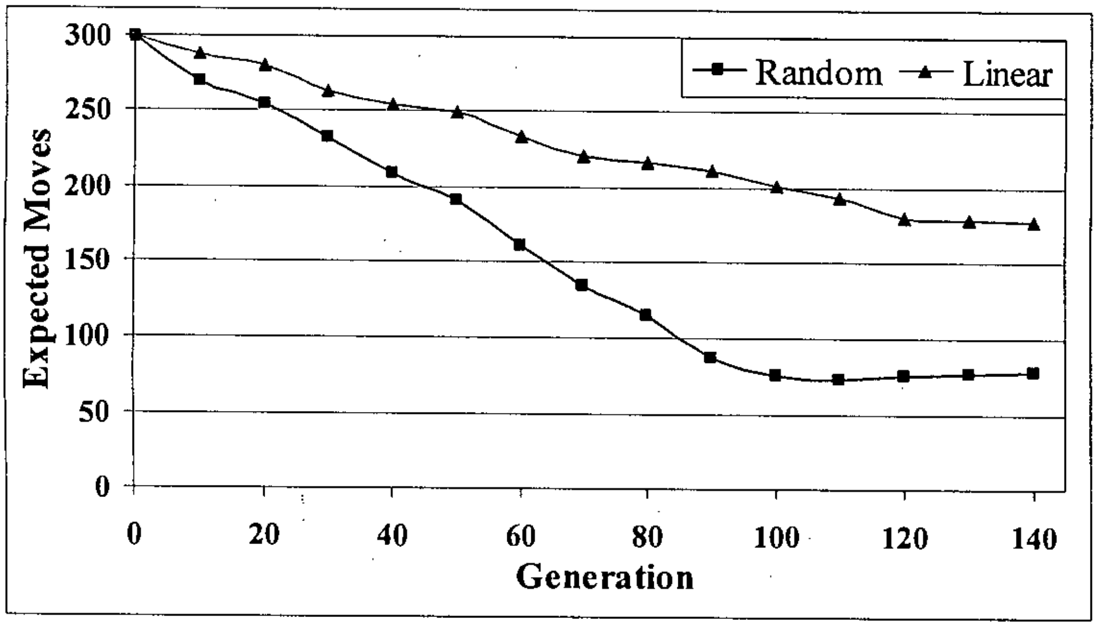


Figure 5.19 Generation vs expected moves for lincar prey and random prey

In table 5.1, for different maximum moves (allowed to capture the prey), the success rate, time to capture the prey successfully (that is, M_{avg}) and the expected number of moves (E_m) are presented. Here the environment is set 100 × 100, equal speed of the predators and the prey and also equal vision power. It is found that success rate increases with the increase of N (that is, maximum number of moves to capture the prey).

Table 5.1: Results varying maximum allowed move to capture the prey

Maximum no of moves allowed (N)	Success rate	Time to capture successfully (M_{avg})	Expected No of Moves (E_m)
100	0.75	47	85
150	0.9	53	78
200	0.96	58	72

In table 5.2, for different number of predators, the success rates, time to capture the prey successfully (that is, M_{avg}) and the expected number of moves (E_m) are presented. Here the environment is set 100 × 100, equal speed of the predators and the prey and also equal vision power. It is found that for three or four predator system, the success rate is quite high but for two-predator system since the environment is toroidal, the prey is able to escape more that 50% of the total program runs.

Table 5.2: Results varying number of predators

No of Predators	Success rate	Time to capture successfully (M_{avg})	Expected No of Moves (E_m)
2	0.43	67	200
3	0.82	63	106
4	0.92	54	74

In figure 5.20, success rates are plotted against generation for two predator strategies (with communication and without communication). It is found that communication predators perform much better than non-communication predators.

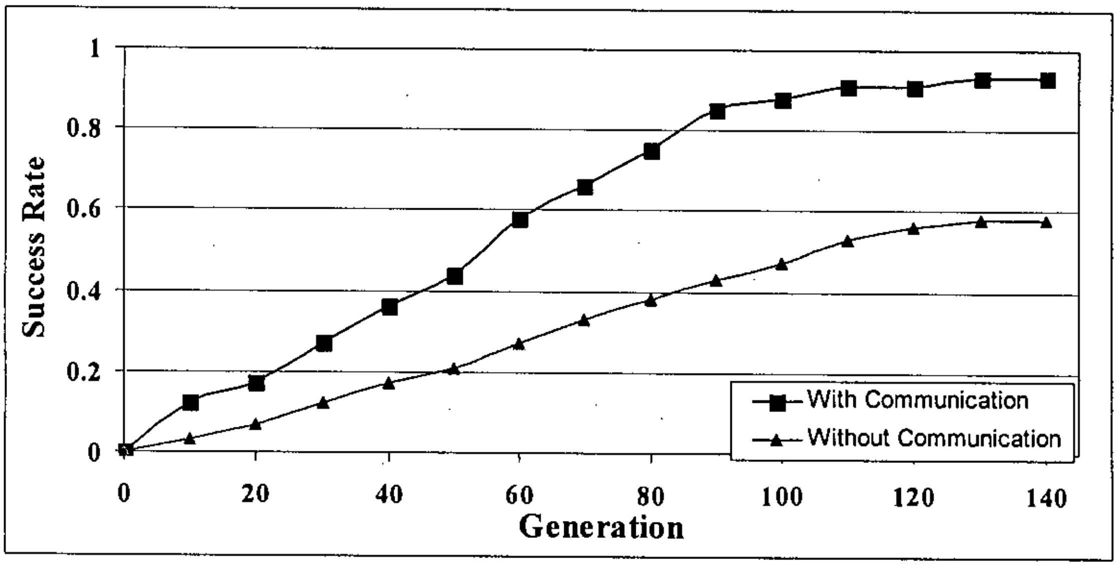


Figure 5.20 Generation vs success rates for communicating and non-communicating predators

In figure 5.21, Expected number of moves is plotted against generation for two predator strategies (with communication and without communication). It is found that non-communicating predators require much higher number of moves than communicating predators.

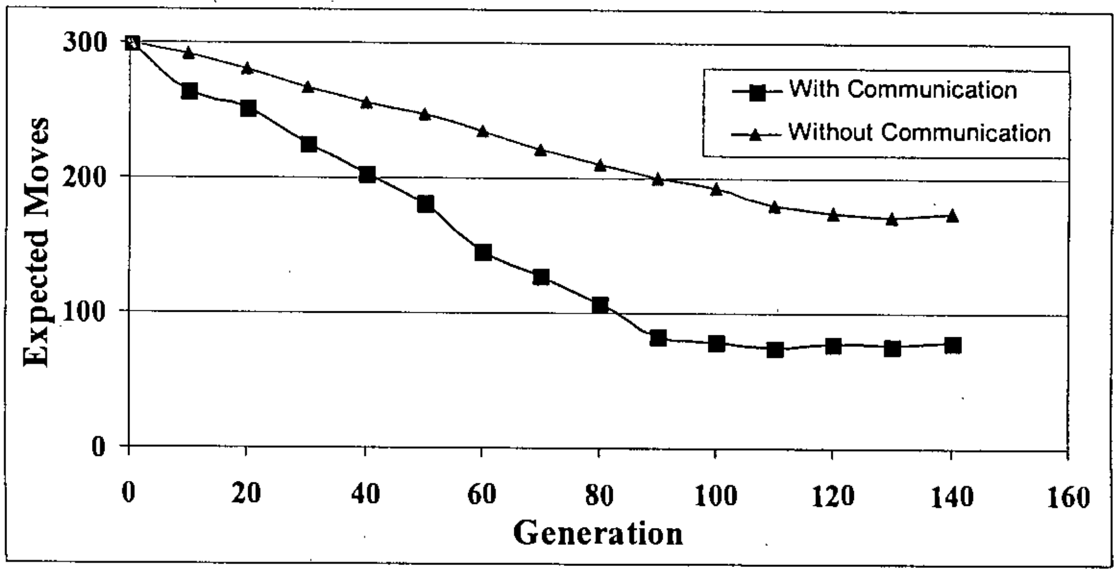


Figure 5.21 Generation vs expected moves for communicating and non-communicating predators

5.5 Analysis

Below is the summary of the results.

- Communication among the predators improves the capture rate.
- CCMAS method produces better results than reinforcement learning irrespective of different environment settings.
- Capture rate is higher than any other strategies.
- Expected number of moves is much less.
- Less number of generations required to capture successfully.
- Linear predators perform better than random moving predators.
- Mutation introduced to the predator direction of movement improves performance.

5.6 Comparison with Existing Works

It is quite difficult to make a direct comparison of the cooperative coevolutionary model for multi-agent system (CCMAS) in predator-prey domain with other existing works due to lack of such results. In some cases, the fitness functions defined are not identical. Moreover, our study uses an implementation which is quite different and more difficult from all the existing works (explained in section 4.3). Still we have tried to compare the results by setting similar environment and system parameters with the existing models.

Chapter 6
Conclusions and Future Works

6.1 Summary

This thesis work is on cooperative coevolution of multi-agent systems. We have developed the CCMAS model for evolving multi-agent systems which have added a new dimension in the research of cooperative coevolution. Our CCMAS model addresses major limitations of traditional evolutionary algorithms such as the issues of problem decomposition, credit assignment to the cooperating agents, maintaining interdependencies among the agents, keeping the population diversified etc by giving suitable solutions to these problems. In order to justify the efficacy and accuracy of the model, we have applied the model in *predator-prey problem* which is a very complex task that is yet to be solved.

103096

The CCMAS model has been found to be advantageous over other evolutionary method in solving prey capture task in the predator-prey domain. The success rates in capturing the prey by the cooperating predators are found to be almost 95% or more. The number of generations required to evolve the predators are also much less and the less number of moves are required to catch the prey.

In our implementation, two prey strategies are used: linear prey and random prey. Result shows that linear prey is more difficult to capture than random prey. Two predator strategies are also tested: predators without any communication and predators with communication. It is found that communicating predators perform much better than non-communicating predators. Our work also performs the sensitivity analysis of different system and environment parameters and on some of the characteristics of *predator-prey problem* likely to affect the performance of the coevolutionary model. Finally the resulting graphs comparing these analyses are presented.

6.2 Future Works

In our model, though the cost of communication has been incorporated, nothing has been mentioned about the means of communication. This can be further incorporated to the CCMAS model. For this type of communication among the cooperating agents, these agents will have to evolve a language that can represent the information to be exchanged among them.

Another point is that the CCMAS model was applied in simulation not on real time environment. While applying this model in real robots, there could be some deviations in performance. Moreover, some system and environment parameter may have to be varied to make it work better in real time environment.

In this work, the CCMAS model is tested on *predator prey domain.* This model can also be tested on other multi-agent domains, such as *Box Pushing* [44, 45], *Cooperative Navigation* [27], *Foraging* [46, 47], *Soccer Team* [48], *Cooperative Target Observation* [49] etc.

References

[1] Hillis, D. W., "Co-evolving parasites improve simulated evolution as an optimization procedure", Artificial Life II, SFI Studies in the Sciences of Complexity, Vol-10, pp 313–324, 1991.

[2] Angeline, P. J. and Pollack, J. B., "Competitive environments evolve better solutions for complex tasks", Proceedings of the Fifth International Conference on Genetic Algorithms, San Francisco, CA: Morgan Kaufmann, pp 264–270, 1993.

[3] Dawkins, R. and Krebs, J. R., "Arms races between and within species", Proceedings of the Royal Society of London, Series B, pp 489–511, 1979.

[4] Rosin, C. D., "Coevolutionary search among adversaries", Doctoral Dissertation, University of California, San Diego, CA, 1997.

[5] Valin, L. V., "A new evolutionary law", Evolution Theory, Vol-1, pp 1–30, 1973.

[6] Samuel, A. L., "Some studies in machine learning using the game of checkers", IBM Journal of Research and Development, Vol-3, No. 3, pp 210–229, 1959, Reprinted Vol-44, No. 1, pp 206–226, 2000.

[7] Juille, H. and Pollack, J.B., "Co-evolving intertwined spirals", Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press, pp 461–467, 1996.

[8] Holland, J. H., "Adaptation in natural and artificial systems", University of Michigan Press (reprinted in 1992 by MIT Press, Cambridge, MA), 1975.

[9] Rosin, C. D. and Belew, R. K., "Methods for competitive coevolution: Finding opponents worth beating", Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, pp 373–380, 1995.

[10] Sims, K., "Evolving 3D morphology and behavior by competition", Artificial Life, Vol-4, pp 28–39, 1994.

[11] Lindgren, K. and Johansson, J., "Coevolution of strategies in n-person prisoner's Dilemma", Evolutionary Dynamics: Exploring the Interplay of Selection, Neutrality, Accident and Function, MA: Addison-Wesley, 2001.

[12] Darwen, P. J., "Co-Evolutionary Learning by Automatic Modularization with Speciation", Doctoral Dissertation, School of Computer Science, University College, University of New South Wales, 1996.

[13] Floreano, D. and Nolfi, S., "God Save the Red Queen! Competition in Co-evolving Robotics", Proceedings of the Second International Conference on Genetic Programming , Morgan Kaufmann, San Mateo, USA, pp 398-406, 1997.

[14] Rosin, C. D. and Belew, R. K., "New methods for competitive evolution", Evolutionary Computation, Vol-5, No. 1, pp 1-29, 1997.

[15] Ficici, S. G. and Pollack, J. B., "Pareto optimality in coevolutionary learning", Sixth European Conference on Artificial Life, pp 316-325, 2001.

[16] Noble, J. and Watson, R. A., "Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection", Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, pp 493-500, 2001.

[17] Husbands, P. and Mill, F., "Simulated co-evolution as the mechanism for emergent planning and scheduling", Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, pp 264–270, 1991.

[18] Paredis, J., "The symbiotic evolution of solutions and their representations", Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, pp 359–365, 1995.

[19] Gomez, F. and Miikkulainen, R., "Incremental evolution of complex general behavior", Adaptive Behavior, Vol-5, pp 317–342, 1997.

[20] Potter, M. A. and Jong, K. A. D., "Cooperative coevolution: An architecture for evolving coadapted subcomponents", Evolutionary Computation, Vol-8, pp 1–29, 2000.

[21] Haynes, T. and Sen, S., "Co-adaptation in a team", International Journal of Computational Intelligence and Organizations, Vol-1, pp 135–142, 1997.

[22] Haynes, T. and Sen, S., "Evolving behavioral strategies in predator and prey", Adaptation and learning in multi-agent systems, Berlin, pp 113–126, 1996.

[23] Haynes, T. and Sen, S., "Crossover operators for evolving a team", Proceedings of Genetic Programming 1997: The Second Annual Conference, pp 162–167, 1997

[24] Werner, G. M. and Dyer, M. G., "Evolution of communication in artificial organisms", Artificial Life, Vol-2, pp 659-687, 1991.

[25] Batali, J., "Innate biases and critical periods: Combining evolution and learning in the acquisition of syntax", Artificial Life, Vol-4, pp 160–171, 1994.

[26] Wagner, K., "Cooperative strategies and the evolution of communication", Artificial Life, Vol-6, pp 149–179, 2000.

[27] Balch, T., "Behavioral Diversity in Learning Robot Teams", PhD thesis, College of Computing, Georgia Institute of Technology, 1998.

[28] Benda, M., Jagannathan, V. and Dodhiawalla, R., "On optimal cooperation of knowledge sources", (Technical Report BCS-G2010-28), Boeing AI Center, Boeing Computer Services, Bellevue, WA, 1985.

[29] Jim, K. and Giles, C. L., "Learning communication for multi-agent systems", 5th ACM International Conference on Autonomous Agents, pp 584-591, 2001.

[30] Jim, K. and Giles, C. L., "Talking helps: Evolving communicating agents for the predator-prey pursuit problem", Artificial Life, Vol-6, No. 3, pp 237–254, 2000.

[31] Hayes, T., Sen, S., Schoenefeld, D. and Wainwright, R., "Evolving a team", Working Notes for the AAAI Symposium on Genetic Programming, MIT, Cambridge, MA, pp 23–30, 1995.

[32] Korf, R. E., "A simple solution to pursuit games", Working Papers of the 11th International Workshop on Distributed Artificial Intelligence, pp 183–194, 1992.

[33] Tan, M., "Multi-agent reinforcement learning: independent vs. cooperative agents", Proceedings of the 10th ICML, pp 330–337, 1993.

[34] Nishimura, S. I. and Ikegami, T., "Emergence of collective strategies in a prey-predator game model", Artificial Life, Vol-3, No. 4, pp 243–260, 1997.

[35] Stephens, L. M. and Merx, M. B., "The effect of agent control strategy on the performance of a dai pursuit problem", Proceedings of the 10th International Workshop on DAI, 1990.

[36] Fogel, D. B., "Evolutionary computation: towards a new philosophy of machine intelligence", IEEE Press, 1995.

[37] Kursawe, F., "Evolution Strategies: Simple Models of Natural Processes", Revue Internationale De Systemique, France, pp 627-642, 1994.

[38] Ridley, M., "The Red Queen: Sex and the evolution of human nature", London: Viking Press, 1993.

[39] Reyes, C. A. P., and Sipper, M., "Fuzzy CoCo: a cooperative coevolutionary approach to fuzzy modeling", IEEE Transaction on Fuzzy Systems, Vol-9, No. 5, pp 727-737, 2001.

[40] Jong, K. A. D. and Spears, W., "On the state of Evolutionary Computation", Proceedings of the Fifth International Conference on Genetic Algorithms Mining, pp 618-623, 1993.

[41] Kauffman, S. A. and Johnsen, S., "Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches", Artificial Life II, SFI Studies in the Sciences of Complexity, Vol-10, pp 325–369, 1991.

[42] Alcazar, J. A., "A simple approach to the multi-predator multi-prey pursuit domain", Proceedings of Fifth International Conference on Complex Systems (ICCS), 2004.

[43] Steels, L., "Self-organizing vocabularies", Proceedings of Alife V, 1996

[44] Mahadevan, S. and Connell, J., "Automatic programming of behavior-based robots using reinforcement learning", National Conference on Artificial Intelligence, pp 768-773, 1991.

[45] Buffet, O., Dutech, A. and Charpillet, F., "Learning to weigh basic behaviors in scalable agents", Proceedings of the First International Joint Conference on Autonomous Agents and Multi-agent Systems, pp 1264-1265, 2002.

[46] Dahl, T. S., Mataric, M. J. and Sukhatme, G. S., "Adaptive spatio-temporal organization in groups of robots", Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vol-1, pp 1044-1049, 2002.

[47] Mataric, M. J., "Interaction and Intelligent Behavior", PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1994.

[48] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. and Osawa, E., "RoboCup: The robot world cup initiative", Proceedings of the First International Conference on Autonomous Agents, pp 340-347, 1997.

[49] Parker, L. and Touzet, C., "Multi-robot learning in a cooperative observation task", Robotic Systems, Vol-4, pp 391-401, 2000.