A Methodology to Evolve Cooperation in Pursuit Domain using Genetic Network Programming

Armin Tavakoli Naeini

A Thesis

in

The Department

of

Electrical and Computer Engineering (ECE)

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

April, 2014

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Armin Tavakoli Naeini

Entitled: "A Methodology to Evolve Cooperation in Pursuit Domain using Genetic Network Programming"

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. M. Z. Kabir

_____ Examiner, External
Dr. M. Y. Chen (MIE) To the Program

_____ Examiner
Dr. X. Zhang

_____ Supervisor
Dr. Chun Wang CIISE)

Approved by: _____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

_____20_____ _____
Dr. C. W. Trueman
Interim Dean, Faculty of Engineering
and Computer Science

Abstract

The design of strategies to devise teamwork and cooperation among agents is a central research issue in the field of multi-agent systems (MAS). The complexity of the cooperative strategy design can rise rapidly with increasing number of agents and their behavioral sophistication. The field of cooperative multi-agent learning promises solutions to such problems by attempting to discover agent behaviors as well as suggesting new approaches by applying machine learning techniques.

Due to the difficulty in specifying a priori for an effective algorithm for multiple interacting agents, and the inherent adaptability of artificially evolved agents, recently, the use of evolutionary computation as a machining learning technique and a design process has received much attention. In this thesis, we design a methodology using an evolutionary computation technique called Genetic Network Programming (GNP) to automatically evolve teamwork and cooperation among agents in the pursuit domain.

Simulation results show that our proposed methodology was effective in evolving teamwork and cooperation among agents. Compared with Genetic Programming approaches, its performance is significantly superior, its computation cost is less and the learning speed is faster. We also provide some analytical results of the proposed approach.

Keywords

Agent, Multi-Agent Systems, Cooperation, Teamwork, Multi-agent Learning, Evolutionary Computation, Genetic Network Programming, Pursuit Domain

Acknowledgements

I wish to thank everyone who helped me complete this study. Without their continued efforts and support, I would have not been able to bring my work to a successful.

First and foremost I offer my sincerest gratitude to my supervisor, Dr. Chun Wang, who has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my Master degree to his encouragement and effort and without him this thesis, would not have been completed or written.

A special thanks goes to the professors and staffs in ECE for providing a delightful academic environment for me to complete this study.

I dedicate this thesis to my beloved parents, for their endless love and support throughout my life; I thank you very much for your great care, love and prayers.

Table of Contents

List of Tables

List of Figures

Acronyms

Nomenclature

Chapter 1

Introduction and Motivation

1.1 Background and Motivation

An agent is a computational mechanism which exhibits a high degree of autonomy when performing actions in its environment based on information (sensors, feedback) received from the environment [1]. Figure 1, illustrates the architecture of a typical agent.

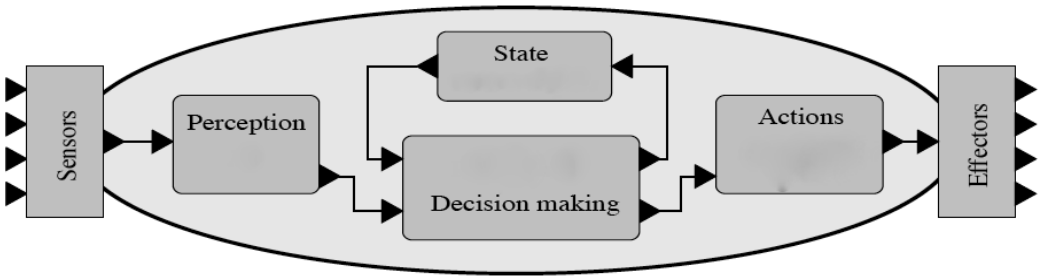

Figure 1. Architecture of a typical agent

Multi-gent systems (MAS) are systems in which a plural number of agents interact with each other and pursue a set of goals. A cooperative multi-agent system (as opposed to competitive multi-agent system) is a multi-agent system in which several agents attempt, through their interactions, to jointly solve tasks or to maximize utility, as illustrated in Figure 2.

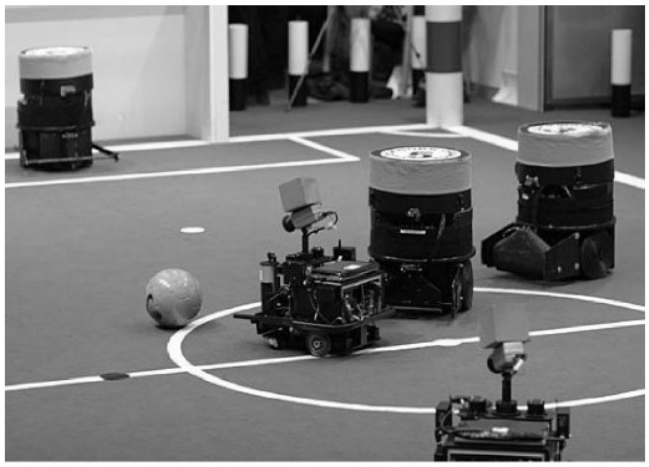

Figure 2. Team of soccer robots, an example of cooperative MAS

Design and development of cooperation and teamwork among the agents in a cooperative multi-agent system is a key research problem. Due to the extensive interactions among the agents, and inherent complexity of such systems, making solutions by hand is prohibitively difficult.

Nature has been quite successful at programming complicated multi-agent systems ranging from simple agents such as ants and bees to sophisticated cognitive agents-ourselves. The mechanism used to create these systems is Darwinian evolution. The global behavior of biological systems such as ant colonies is considered to be an emergent property of interactions between the different agents that make up system as a whole. Hence the agents' macro-level (group behavior) is called *emergent behavior*, and the cooperation achieved in this manner is called *emergent cooperation*.

Evolutionary Computation (EC) is a family of Machine Learning (ML) techniques in which abstract Darwinian models of evolution are applied to refine populations of candidate solutions (known as "individuals") to a given problem and evolve solutions automatically. Nowadays using EC techniques as a design process for control of agents and especially robots have received much attention in both real world as well as simulated problem domains. Such attention is due to the difficulty in specifying a priori an effective means of controlling multiple interacting agents, and also the inherent adaptability of artificially evolved controllers, thereby making them a problem-solving methodology that is potentially applicable in a wide range of problem domains.

Many problems in MAS, which require teamwork and cooperation, are solvable by multiple agents each of which uses essentially the same algorithm. Such agents are called *homogeneous* and are common candidates for the study of emergent behavior because only one simple set of rules for these agents can give rise to complex patterns of behavior during their interactions.

In this thesis, we design and develop a novel methodology using a relatively new evolutionary computation technique called Genetic Network Programming (GNP) [7] to evolve teamwork and cooperation among four homogenous agents named predators in the pursuit domain. The results are then compared to those of Genetic Programming (GP) in terms of performance and computational cost.

It is generally said that it is difficult to search for an optimum solution by using GP, since the search space of solutions becomes enormous due to its *bloat*. For this reason, the searching efficiency of GP is not such high in most cases. Consequently, GNP has recently been proposed to overcome this limitation in GP.

While GP uses tree structure to represent solutions, GNP uses a graph (network) structure. Graph representation in GNP improves solution representation and search ability and also prevents bloating, which is a common drawback in most GP-based multi-agent systems.

Bloating is the uncontrolled and unbounded growth of individuals (programs), usually independent of sufficiently justified improvements in their fitness. Bloat slows down the evolutionary search process, consumes memory, and can hamper effective breeding [9].

GNP like other EC techniques is a means of evolving solutions to difficult problems, where the answer is not obvious. GNP allows problems to be solved without explicitly programming the solution. This is done using a fitness function. The fitness function rates the quality of a possible solution. Good solutions are combined with other good solutions to hopefully create even better solutions. By continuing this process, GNP can evolve optimal or near optimal solutions. GNP has been recently tested with success on some domains [8, 10, 11].

1.2 Problem Definition and Challenges

The Pursuit problem is a well-studied test bed in MAS research (Figure 3). Benda et al [42] formulated the original problem definition as consisting of four *predators* that need to capture a single *prey*. The predators and the prey are situated on a two-dimensional world consisting of cells. This world is toroidal and the movements, which take the agents from one cell into another, are only possible in orthogonal directions. The goal of the predators is to move so as to *capture* the prey. To capture the prey, the predators must occupy all cells immediately adjacent to the prey. The prey moves randomly, and is slower than the predators. Only one agent may occupy a cell at a time, therefore *conflict*s occur if one or more agents try to occupy the same cell, thereby a *conflict resolution mechanism* is needed to resolve the conflicts.

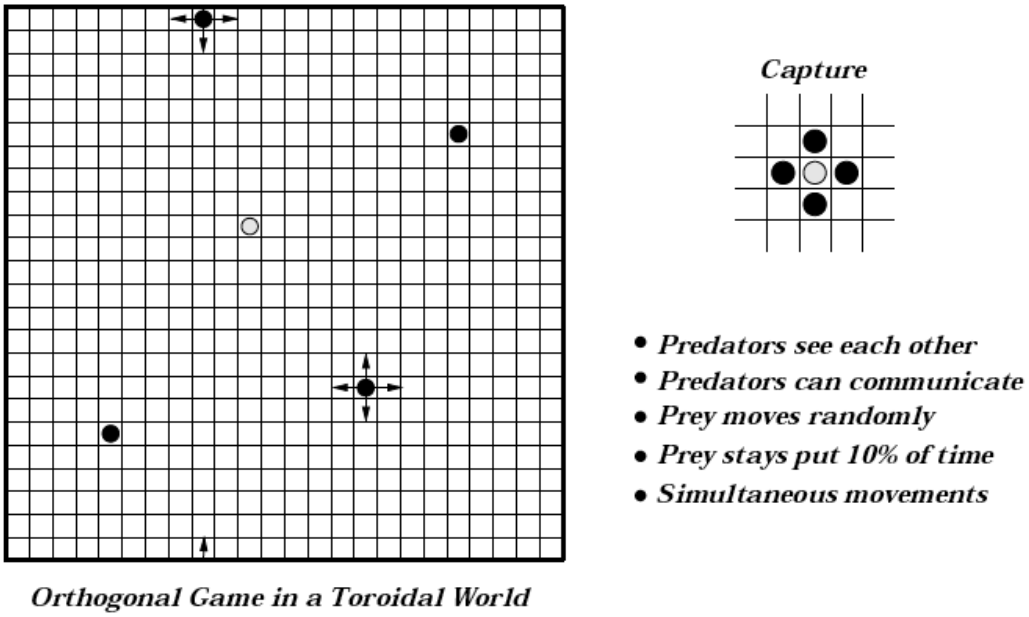

Figure 3. The pursuit domain

Our hypothesis is that GNP can be used to automatically program predators in the pursuit domain and evolve teamwork and cooperation among them so that they may capture the prey. Also given its inherent architecture its performance should be superior to GP.

To prove our hypothesis we attempt to answer the following questions:

1) What are the main issues in designing and developing a cooperation strategy for a multi-agent system in general and the pursuit domain in particular when using GNP?
2) Is GNP capable of providing a solution to the cooperation problem for the pursuit domain?
3) Are GNP results better than the GP results in terms of computational cost, capture rate and bloating?
4) Is GNP capable of overcoming the bloating problem highlighted by GP?
5) Is the proposed methodology robust, in other words, does it behave persistently under uncertain conditions in a dynamic domain?
6) What are the technical problems in scaling GNP to real world multi-agent applications?

Our key contributions are as follows:

1) Design, development and implementation of a successful GNP-based methodology capable of evolving homogenous cooperated team of agents in a difficult-to-solve MAS, named pursuit domain.
2) Showing that GNP-based system is superior to a GP-based system in terms of performance and computational costs.
3) Mathematically proving that our proposed GNP-based methodology overcomes the bloating problem in GP and has a constant complexity in terms of big-O notation whereas it is quadratic in a GP-based system.

1.3 Outline of the Thesis

The remaining thesis is structured as follows. Chapter 2 introduces cooperative multi-agent learning and reviews previous studies on how machine learning techniques are used to devise cooperation in multi-agent systems. It specifically surveys the literature related to applying evolutionary techniques in the pursuit domain. Chapter 3 describes the evolutionary algorithms in general and then it specifically introduces, describes and compares GP and GNP. Chapter 4 describes the pursuit problem and its parameters in detail. It also presents the Pursuit _Package_0.9 [44], which is used for the experimental simulations in this thesis. Chapter 5 demonstrates the design of our proposed methodology for the pursuit domain. In chapter 6, simulation results are shown and critically evaluated. It also compares and discusses the performance of our GNP-based methodology with GP. Finally chapter 7 concludes and further research directions stemming from our work are proposed.

Chapter 2

Literature Review

Design and development of cooperation strategies and interaction scheme among a set of agents using hand-coded, i.e. preprogrammed, approaches is prohibitively difficult and complex in multi-agent systems. This complexity is due to the agents' behavioral sophistications and the extensive amount of interactions among the agents. An idea that has become more and more interesting and therefore a research goal among many computer scientists is the integration of learning and adaptation capabilities in the initial design process of agents in order to allow them to evolve the intended behavior automatically, and in consequence at much cheaper costs. Hence, machine learning has proven to be a popular and flexible approach for solving cooperation problems in multi-agent systems recently. Cooperative multi-agent learning is a field of research, which studies the ways machine learning techniques can be used to develop teamwork and cooperation strategies among multiple agents in a multi-agent system.

This chapter describes background information and presents literature review related to MAS and cooperative multi-agent learning. *Emergent cooperation* which is a biologically inspired bottom-up cooperation problem-solving approach for multi-agent systems is described thoroughly in a separate section. Finally, a detailed review of literature which has used *emergent cooperation* in solving cooperation problem in the pursuit domain is presented.

2.1 Multi-Agent Systems

In recent years there has been increased interest in decentralized approaches for solving complex real-world problems. Such approaches fall into the area of *distributed systems,* where a number of entities work together to cooperatively solve problems. The combination of distributed systems and artificial intelligence (AI) is collectively known as *distributed artificial intelligence* (DAI). Traditionally, DAI is divided into two areas.

The first area, *distributed problem solving*, is usually concerned with the decomposition and distribution of a problem solving process among multiple slave nodes, and the collective construction of a solution to the problem using AI methods. The second class of approaches, *Multi-Agent Systems* (MAS), emphasizes the joint behaviors of agents with some degree of autonomy and the complexities arising from their interactions [2]. *Autonomy* conventionally refers to the degree to which an agent is able to make its own decisions about which actions to take next.

The term *multi-agent* is not well-defined in the community and different researchers have different definitions; We prefer Panait and Luke [12] definition of this concept which is relatively new and admittedly broad: A *multi-agent* environment is one in which there are more than one agent, where they *interact* with one another, and further, where there are *constraints* on that environment such that agents may not at any given time know everything about the world that other agents know (including the internal states of the other agents themselves).

Interaction and *constraints* are important points to the notion of multi-agent system problem definition. If the domain requires no interaction at all, then it may be decomposed into separate, fully independent tasks each solvable by a single agent and there is no need for a multi-agent system. Additionally if there are no constraints on the environment the distributed agents can act in sync with knowing exactly what situation the other agents are in and what behavior they will pursue and this "*omniscience*" permits the agents to act as if they were really mere appendages of a single master controller.

An example of a multi-agent system is an environment with multiple *foraging* robots. Foraging robots are mobile robots capable of searching for objects and when found, transporting them to one or more collection points. The problem becomes more complex when the robots can suggest good regions for search to each other and also help one another to forage. Figure 4, shows a typical cooperative foraging multi-agent system.

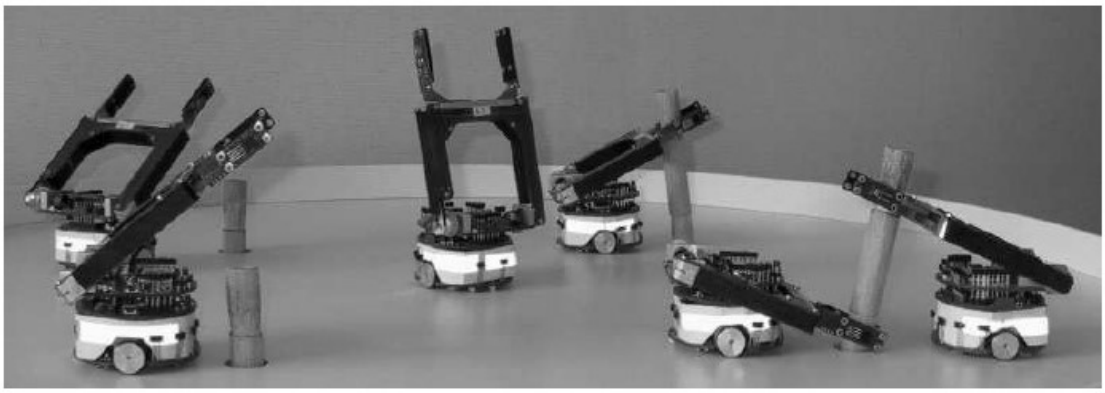

Figure 4. Multiple foraging robots engaged in stick pulling. An example of MAS

2.2 Machine Learning Methods

Machine learning techniques can be divided into three main categories: *supervised*, *unsupervised*, and *reward-based* learning. These categories are distinguished by the kind of feedback that critic provides to the learner. In unsupervised learning, no feedback is provided at all to the learner. In supervised learning, the critic provides the correct output. In reward-based learning, the critic provides a qualitative assessment of the learner's output, and is called the *reward*.

Supervised learning methods cannot be easily applied to the MAS problems because of the inherent complexity in the interactions of multiple agents, which makes it almost impossible for the critic to provide the agents with the *correct behavior* for a given situation and it is often infeasible to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act.

The reward-based learning literature can be divided into two main subsets: *reinforcement learning (RL)* methods, which estimate value functions and *evolutionary computation (EC)* methods, which directly learn behaviors without using value functions. RL methods are inspired by dynamic programming concepts and EC methods are inspired by Darwinian model of evolution.

The next subsections describe RL and EC techniques. A review of the literature related to applying these ML methods in solving MAS cooperation problems is presented in the section 2.3.

2.2.1 Reinforcement Learning

Reinforcement Learning is learning what to do, i.e. how to map situations to actions, so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning methods, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent reward. These two characteristics, which are trial-and-error search and delayed reward, are the two most important distinguishing features of reinforcement learning. RL methods are particularly useful in domains where reinforcement information can be provided after a sequence of actions performed in the environment. Reinforcement is provided to the learner as a positive or negative numerical value to express a *reward* or a *penalty*.

The two common RL methods are Q-Learning and Temporal-Difference (TD(λ)) Learning; the former learns the utility of performing actions in states, while the latter usually learns the utility of being in the states themselves.

Reinforcement learning methods are inspired by dynamic programming concepts and define formulas for updating the expected utilities and for using them to explore the state space. The update is often a weighted sum of the three elements: current value, the reinforcement obtained when performing an action, and the expected utility of the next state reached after the action is performed.

RL methods have theoretical proofs of convergence; unfortunately, such convergence assumptions do not hold for some real-world applications, including many multi-agent systems problems. So they are not applicable for a wide variety of real-world problems. For more information on reinforcement learning techniques, [20, 21, 22] are good starting points.

2.2.2 Evolutionary Computation

Evolutionary Computation (EC) (or *Evolutionary Algorithms* (EAs)) is a family of techniques in which abstract Darwinian model of evolution, i.e. "*survival of the fittest*", is applied to refine populations of candidate solutions known as "individuals" to a given

problem. An evolutionary algorithm begins with an initial population of randomly generated individuals. Then each member of this population is *evaluated* and a "fitness" is assigned to it. Fitness is a qualitative assessment of an individual and in the other words it shows how good an individual (candidate solution) is. EC then uses a fitness-oriented algorithm in order to "*select"*, "*reproduce"* and "*vary"* the fittest individuals to produce children, which will be added to the population, replacing older individuals. One evaluation, selection, and breeding cycle is known as a "*generation"*. By repeatedly generating subsequent solutions in the same manner better solutions can theoretically be generated progressively. By repeating this process, successive generations continue to refine the population until time is exhausted or a sufficiently fit individual is discovered.

Evolutionary computation methods include *Genetic Algorithms* (GA) and *Evolution Strategies* (ES), and *Genetic Programming* (GP), and *Genetic Network Programming* (GNP). There are many sources of additional information on EC; Good choices to start include [7, 8, 9].

2.3 Multi-Agent Learning

The application of machine learning techniques to solve problems involving multiple agents is called Multi-agent learning. Thus the field of multi-agent learning studies the ways machine learning techniques can be used to solve MAS problems.

Multi-agent learning has two specific features, which make it a field separate from ordinary machine learning and merit its study. First, because multi-agent learning deals with problem domains involving multiple agents, the search space involved can be unusually enormous; and due to the interaction of those agents, small changes in learned behaviors can often result in unpredictable changes in the resulting macro-level ("emergent") properties of the multi-agent group as a whole. Second, multi-agent learning *may* involve *multiple learners*, each learning and adapting in the context of others; and this may introduces game-theoretic issues to the learning process.

The topic of multi-agent learning can be divided into two broad categories*: Cooperative multi-agent learning* and *competitive multi-agent learning*. As Panait and Luke [12] suggest, we have a cooperative multi-agent learning system if the design of the problem

and the learning system is constructed so as to (hopefully) encourage cooperation among the agents. Hence just existence of a cooperation encouragement mechanism in design of MAS is sufficient for it to be cooperative MAS, even if agents fail to do so finally. And we have a competitive multi-agent learning setting if agents have non-aligned goals, and individual agents seek only to maximize their own gains.

The rest of this section specifically focuses on the cooperative multi-agent learning approaches, which can be divided into two major categories. The first one, which is termed *team learning*, applies a single learner to search for behaviors for the entire team of agents while the second category, which is called *concurrent learning*, uses multiple concurrent learning processes.

In team learning approaches since there is only one learner involved for the entire team, they are more similar to the traditional ML approaches, but they may have scalability problems as the number of agents is increased in the environment.

Concurrent learning approaches typically employ a learner for each team member, so there would be N different learners in a team of N agents. Thus the joint search space is reduced by projecting it into N separate search spaces. However, the presence of multiple concurrent learners makes the environment non-stationary, which means the dynamics of the environment might change due to some unknown or not directly perceivable causes and it is a noticeable violation of the assumptions behind most traditional machine learning techniques. For this reason game-theoretic issues are introduced to concurrent learning processes and they require new or significantly modified versions of ML methods.

2.3.1 Team Learning

Team learning is a cooperative multi-agent learning approach in which there is a single learner involved and this learner discovers the set of behaviors for the entire team of agents, rather than a single agent. This lacks the game-theoretic aspect of multiple learners, but still poses challenges because as agents interact with one another, the joint

behavior can be unexpected. This complexity arising from agents' interaction and their joint behavior is often called the *emergent complexity* of the multi-agent system.

Team learning is relatively easy approach to multi-agent learning because there is only one learner and it can use standard single-agent machine learning techniques and it sidesteps the difficulties arising from the co-adaptation of several learners in concurrent learning approaches. Another advantage of a single learner is that it is concerned with the performance of the entire team, and not with that of individual agents.

A major problem with team learning is the large state space for the learning process. For example, if agent A can be in any of 100 states and agent B can be in any of another 100 states, the team formed from the two agents can be in as many as 10000 states. This explosion in the state space size can be overwhelming for learning methods that explore the space of state utilities (such as reinforcement learning), but it may not so drastically affect techniques that explore the space of behaviors (such as evolutionary computation). For such reasons, evolutionary computation seems easier to scale up, and it is by far the most widely used team learning technique [12, 13]. A second disadvantage is the centralization of the learning algorithm since all resources need to be available in a single place where all computation is performed. This can be burdensome in domains where data is inherently distributed.

Team learning may be divided into three categories: *homogeneous* and *heterogeneous* and *hybrid* team learning. Homogeneous learners develop a single agent behavior, which is used by every agent on the team. Heterogeneous team learners can develop a unique special behavior for each agent, so it encourages agent specialization. Heterogeneous learners must cope with a larger search space, but has the possibility of better solutions through agent specialization. Hybrid team learning methods lay in the middle-ground between these two categories. Hybrid team learning divides the team into squads, with squadmates having the same behavior and different from the other squads.

Choosing among these approaches depends on whether *specialists* are needed in the team or not. Experiments conducted by Balch [46], Bongard [47] and Potter et al. [48] address exactly this issue.

Balch [46] suggests that domains where single agents can perform well, such as foraging, are particularly suited for homogeneous learning, while domains that require task specialization such as robotic soccer are more suitable for heterogeneous approaches.

Homogenous Team Learning

In homogeneous team learning, all agents are assigned identical behaviors, even though the agents may not be identical (for example, different agents have different computational capability and may take a different amount of time to complete the same task). Because all agents have the same behavior, the search space for the learning process is drastically reduced. The appropriateness of homogeneous learning depends on the problem. It is suitable for problems, which do not require agent specialization to achieve good performance. They are also suitable for problem domains, which have a too large search space. In these problem domains, particularly ones with very large numbers of agents ("swarms"), even though heterogeneity and specialization would ultimately yield the best results, the search space is simply too large to use heterogeneous learning.
In a straightforward example of successful homogeneous team learning, Haynes et al. [49], Haynes and Sen [50], Haynes et al. [45, 51] evolved behaviors for a predator-prey pursuit domain using genetic programming. When using fixed algorithms the prey behavior, the authors report results competitive with the best human-coded predator algorithms. In another well-known example, Quinn et al. [52] investigate the use of evolutionary computation techniques for a team formation problem.

Heterogeneous Team Learning

In heterogeneous team learning, the team is composed of agents with different behaviors, with a single learner trying to improve the team as a whole. This approach allows for more diversity in the behaviors of the agents in a team at the cost of increasing the search space [53]. The bulk of research in heterogeneous team learning has concerned itself with the requirement for the emergence of specialists. In a well-known example Andre and Teller [54] apply genetic programming to develop a team of soccer playing agents for the RoboCup simulator. The individuals encode eleven different behaviors (one for each player). In another renowned example Haynes and Sen [49, 50, 51] investigate the evolution of homogeneous and heterogeneous teams for the pursuit domain.

The authors present several crossover operators that may encourage the appearance of specialists within the teams. The results indicate that team heterogeneity can significantly help despite apparent homogeneity among the predators in the pursuit domain.

Hybrid team learning

In hybrid team learning, the set of agents is split into several squads, with each agent belonging to only one squad. All agents in a squad have the same behavior. One extreme, in which we have only one squad, is equivalent to homogeneous team learning, while the other extreme in which there is just one agent per squad is equivalent to heterogeneous team learning. Thus hybrid team learning permits the experimenter to achieve some of the advantages of each method. Luke [55],Luke et al. [56] focus on evolving soccer teams for the RoboCup competition. They compare the fully homogeneous results with a hybrid approach that divides the team into six squads of one or two agents each, and then evolves six behaviors, one per squad. The authors report that homogeneous teams performed better than the hybrid approach, but mention that the latter exhibited initial offensive defensive squad specialization and suggest that hybrid teams outperform the homogeneous ones. Figure 5, shows Luke [55] results for RoboCup competition, in which after a number of generations the initial population of randomly moving soccer agents began to develop elementary defensive ability.

Figure 5. RoboCup Soccer. After a number of generations some of the initially random moving players, learned to hang back and protect the goal, while others chase the ball

2.3.2 Concurrent Learning

Concurrent learning, in which there exists multiple learning processes instead of a single learner and each one attempts to improve a part of the team is considered the most common alternative to team learning in cooperative multi-agent systems. Typically each agent has its own exclusive learning process to modify its behavior. But of course there could be degrees of granularity in concurrent learning approach. For example the team may be divided into *squads*, each with its own learner.

Concurrent learning and team learning each have their defenders and critics. Bull and Fogarty [57] and Iba [17] present experiments where concurrent learning preforms better than both homogeneous and heterogeneous team learning, while Miconi [58] reports that team learning is superior in certain applications. Now the important question is then when each method would be preferred over the other? Jansen and Wiegand [59] argue when some decomposition is possible and helpful, concurrent learning may be preferable, and also in those domains for which it is convenient and beneficial to focus on each sub-problem to some degree independently of the others. The reason for this is that concurrent learning maps the large joint team search space onto separate, smaller individual search spaces. If the problem can be decomposed such that individual agent behaviors are relatively disjoint, then this can result in a significant reduction in search space and therefore in computational complexity. A second, related advantage is that when the learning process is broken into smaller processes, it allows more flexibility in the use of computational resources for learning because the smaller learning processes may, at least to some extent, be learned independently of one another.

The main challenge for concurrent learning is that standard machine learning techniques may not be applicable since each learner is adapting its behaviors in the context of other co-adapting learners over which it has no control. In single-agent scenarios where standard machine learning techniques can be used, a learner explores its environment, and while doing so, improves its own behavior. While when there are multiple learners, as the agents learn, things in the environment may change and agents need to modify their behaviors, which in turn can make obsolete the assumptions on which other agents use during their learning process and damage their learned behaviors [58, 59].

Concurrent learning literature breaks down along more different lines than team learning literature. Since each agent is free to learn separately, heterogeneity versus homogeneity has been considered an emergent aspect rather than a design decision in concurrent learning.

2.4 Evolutionary Methodologies in Multi-Agent Systems

This section aims to provide an overview of important research contributions that investigate utilization of concepts such as emergence, evolution and self-organization as a means of attaining cooperation in complex systems followed by a survey of emergent cooperation using evolutionary methodologies especially in the pursuit domain.

In nature there are lots of examples of systems whose complex stable behaviors emerge from simple local interactions during their evolution. For example the global complex behavior of biological systems such as ant colonies, bird flocking, fish shoaling and herd behavior of land animals are considered to be an emergent property of the simple interactions between the different individuals that make up the whole system.

In anthropology, researches are based on the belief that individual intelligent behaviors of people must be observed and analyzed within their social and therefore cultural environment and cannot be understood in isolation. Societies of individuals vary in size and complexity but have a common property. They provide and maintain a shared *culture*. Wikipedia defines culture as: “The distinct ways that people who live differently, classified and represented their experiences and acted creatively”. The complexity of a culture results from the local interactions among the individuals.

n contrast to traditional AI, which addresses intelligence as an isolated phenomenon, recently there has been an increased research interest on intelligence as a *social* phenomenon. The concept of emergent behavior in MAS is inspired by the biological systems in nature and culture concept in anthropology. Emergent behaviors are defined as purposive group (macro-level) behaviors, which are results of simple local (micro-level) interactions among individuals. In other word emergent behavior is the desirable or not desirable behavior of a system that is not explicitly described by the behavior of the

components of the system, and is therefore unexpected to a designer. Emergent behaviors are characterized by following properties: 1) it is manifested by global patterns in behaviors which are not explicitly programmed but result from local interaction among a system's components. 2) It is considered interesting based on some observer-established metric.

Early research in decentralized systems suggested [59] that complexity at a group level might be attainable with very simple individual agents, with no need for central control. For instance, Grey Walter and his colleagues [60] studied simple robots equipped with light and touch sensors and very simple behaviors. When placed together, these robots exhibited complex social behavior in response to each other's movements.

2.4.1 Emergence of Cooperation in Multi-Agent Systems

Artificial Life studies the logic of living systems in artificial environments in order to gain a deeper understanding of the complex information processing that defines such systems [60]. Specifically agent-based systems, which are used to study the emergent properties of societies of agents, are also included in the umbrella term of artificial life. Despite traditional AI, which relies on top-down modularity, artificial life typically adopts a bottom-up approach to model emergent social phenomena and various forms of collective behavior observed in biological systems.

Emergent cooperation is a key topic in artificial life research. Therefore emergent cooperation is considered a bottom-up problem-solving methodology, which is potentially applicable to a wide range of problem domains. It is very interesting for researchers in this field because it appears to provide something from nothing [60].

Cooperation among the agents in a multi-agent system can be recognized as a desirable emergent behavior. Desirable emergent behavior has been observed in many biological systems, though reproducing the conditions leading to the emergence of such behaviors in artificial systems has proved to be difficult as there is potential for the emergence of undesirable behaviors. It is therefore essential to be able to understand the mechanisms that motivate emergent cooperative behavior in these systems.

To date, research, which qualitatively measures and evaluates mechanisms underlying and motivating emergent cooperative behavior in real world and artificial systems remains largely in stage of research infancy.

A derivative of emergent cooperation includes the artificial systems typically designed using an *evolutionary computation (EC)* methodology such that a global organized behavior emerges from interaction of the systems components [60, 61, 62]. It has been argued by many researchers [61, 62, 63] that the use of biologically inspired principles such as evolution and emergence in the purposeful design of complex artificial systems is needed in order to replace ineffective preprogrammed and centralized design methodologies.

2.4.2 Evolutionary Methodologies in the Pursuit Domain

The use of biologically inspired design principles for investigating emergent cooperative behavior remains a relatively unexplored area of research in the pursuit domain. The original version of the pursuit-evasion problem introduced by Benda *et al* [42] and the goal of this research was to illustrate emergent cooperative behavior from the interactions of predators following simple pursuit strategies.

Since then, different approaches [43, 50, 51] have been used to study cooperative pursuit strategies in the pursuit domain. Research investigating cooperative pursuit strategies, typically studies cooperative behavior in the context of *a game theory* model [64, 65, 66] in which a gird world pursuit domain is casted in game-theoretic terms; essentially as a strategy game consisting of matrices of payoffs for each predator versus a single prey agent based on their joint actions. But, a few researchers [50, 51, 67], have investigated emergent behavior in the form of cooperative behavior that *emerges* within a group of predators as a need to collectively capture a prey.

For instance, throughout a series of papers, Haynes and Sen [49, 50, 51] compared different genetic programming approaches for the evolution of cooperative pursuit strategies. They also proposed [50] a new approach for the development of cooperative strategies, which was derived from genetic programming and tested it within a pursuit-evasion game scenario. The authors argued that the approach differed from existing

approaches because in that research, pursuit strategies were incrementally constructed via repeatedly evolving and testing them for increasingly difficult pursuit tasks. Additionally the authors claimed an important feature for their GP-based approach that is their approach does not rely on domain specific knowledge and only relied upon the performance of emergent solutions and therefore it can be used in other domain easily.

In their experimental setup they used a grid world where initially the prey was placed in the center and four predators in random positions. A predator could see the prey, though not other predators, and there was no explicit form of communication between the predators. For all experiments, a genetic programming approach called *Strongly Typed Genetic Programming* (STGP) devised by Montana [68] was applied to the task of evolving a program that represented a behavior that is a pursuit strategy. The pursuit behavior (strategy) was shared by all the predators in the case of homogenous teams, and was unique to each predator in the case of heterogeneous teams. In order to generate generalized solutions that were not dependent upon initial agent positions, each pursuit strategy in the population of strategies was evaluated by testing it in k randomly generated pursuit-evasion scenarios. The program with the highest percentage of capture was taken as the fittest. The STGP technique first randomly generated a population of N programs, and then assigned fitness to each after executing and evaluating them in a pursuit-evasion scenario. A subset of the N programs was then selected for generating a new population of programs by combining the selected programs and swapping random sub-parts of the programs. One hypothesis of this research was that evolution of these structures, incrementally evaluated and updated, would produce effective cooperative pursuit strategies for heterogeneous as well as homogenous team of predators. Homogenous teams consisted of k predators that all shared the same behavioral pursuit strategy (programs), and the evolutionary process would maintain a population of these behavioral strategies.

Heterogeneous teams also consisted of k predators, though each predator utilized a different behavioral strategy. The evolutionary process maintained a population of team-level strategies, where each team-level strategy consisted of some combination of the k behavioral strategies that represented all predators in the team.

Haynes and Sen [50, 51] introduced a *cooperative co-evolutionary* process into their experiments, which was designed to facilitate the development of more complex forms of cooperative pursuit strategies in teams of heterogeneous predators.

Coevolution in evolutionary algorithms refers to maintaining and evolving individuals for different roles in a common task, either in a single population or in multiple populations. It is divided into two categories cooperative coevolution and competitive coevolution. In *competitive coevolution* these roles are adversarial in that one agent's loss is another one's gain. However in cooperative coevolution, the agents share the rewards and penalties of successes and failures. The kinds of problems that can best utilize cooperative coevolution are those in which the solution 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.

Haynes and Sen [50] hypothesis was that k different behavioral strategies for controlling the actions of k different predators could be combined, through a cooperative coevolution process, to form a cooperative strategy to achieve some predefined global goal. The authors' supposition was that a cooperative coevolution, as opposed to competitive coevolution approach [51] would be more effective in the derivation of complex cooperative pursuit strategies. They utilized the STGP methodology in order to evolve behavioral strategies, which enabled a heterogonous team of predators to cooperatively achieve a common goal. Each predator team consisted of k programs, where each program explicitly represented an individual predator, or more precisely, a behavioral aspect of the cooperative team strategy that emerged when the predators interacted. Thus entire teams of predators were evolved, as opposed to individual predators, so that a particular combination of the k programs comprising the team would determine the behavioral strategy of a particular team. Each predator always participated in the same team and fitness was assigned to the team as a whole.

Haynes and Sen [50] also implemented a series of experiments that evaluated a set of new genetic programming crossover mechanisms for evolving cooperative strategies amongst a heterogeneous team of predators. Results indicated that only one of the new

crossover mechanisms, named *team-uniform* by the authors, evolved a team faster than the traditional crossover mechanisms. The team uniform crossover mechanism was found to accelerate the evolutionary process, and also facilitate emergent cooperative pursuit strategies with a higher average fitness within heterogeneous teams of predators. In several additional experiments, the role of communication between the predators was also studied and found to be unnecessary and even damaging. Specifically, in the experiments that did not use communication each non-communicating subpopulation converged towards the optimization of a specific function in the team. This resulted in the derivation of cooperative pursuit strategies facilitated by the emergence of predators with specialized and complementary pursuit behaviors. Specifically, certain predators, termed *chasers* by the authors, only chased the prey, while other predators, termed *blockers* by the authors, only attempted to block the path of the prey.

The authors compared the evolution of cooperative strategies using homogenous and heterogeneous teams in experiments testing two types of prey agents, those that moved randomly and those that attempted to maintain a maximum distance from the predators. Results illustrated that emergent cooperative pursuit strategies outperformed all but one of four preprogrammed heuristic pursuit strategies, which used a greedy search algorithm [51], and that the emergent cooperative strategies of heterogonous teams outperformed homogenous teams. The authors concluded that their genetic programming approach was an effective way for deriving cooperative behavior, given that it required no explicit communication and minimal domain knowledge.

The key criticism of these series of research is the open questions concerning emergent specialized behavior and how cooperative behavior emerged. For example, it remains unclear which part of the genetic programming tree structure describes a predators cooperative pursuit behavior in the case of a heterogeneous approach, or a team's behavior in the case of a homogenous approach.

Although, emergent team-level cooperation in homogenous teams, and then emergent specialization in the formation of cooperative pursuit strategies with heterogeneous teams were interesting results, the emergence of such behaviors can largely be attributed to the genetic programming implementation and the simple grid world environment utilized.

Also, the application of the genetic programming methodology to other problem domains was not reported, so it remains uncertain if the cooperative behaviors would emerge beyond the grid world implementation.

Yong and Miikkulainen [67] conducted a research similar to the Haynes and Sen [50] research, investigated the role of behavioral specialization in the evolution of cooperative pursuit strategies in a pursuit-evasion scenario. This research compared two artificial evolution approaches for the incremental evolution of a neural network architecture, where this architecture controlled the behavior of predators. The first approach was a centralized controller, where a single neural network controlled all predators, and the second method was a distributed approach where a separate neural network controlled each predator in the team. For both of these approaches, an incremental approach to artificial evolution was used, such that evolved neural networks were first tested upon a relatively simple pursuit-evasion task and then upon increasingly complex tasks. The incremental evolutionary process proceeded through five stages, where in the simplest stage the prey was stationary, and in each subsequent stage the prey moved progressively faster, until in the final stage it moved equally as fast as the predators. The authors argued that the advantage of this incremental evolutionary approach was that it prevented the artificial evolution algorithm from converging to a solution in a sub-optimal region of the solution space. These approaches for artificial evolution were based on an architecture termed Enforced Sub-Populations (ESP) [69] and it is illustrated in Figure 6.

The enforced sub-populations approach used multiple populations of neurons and at the turn of every generation a single individual, in this case a neuron, was selected from each population of neurons in order to construct the neural network for controlling an individual predator or a predator team. The enforced subpopulations approach to artificial evolution was used to encourage the emergence of specialized behavioral roles in cooperative pursuit behaviors, such as the chaser and blocker behaviors evident in the experiments of Haynes and Sen [50].

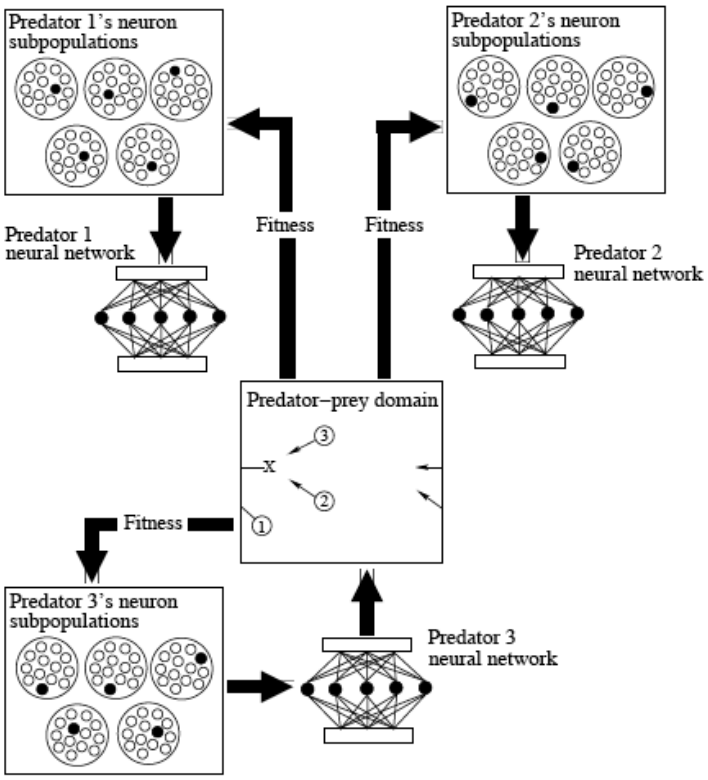

Figure 6. Multi-agent ESP architecture. Each predator is controlled by its own neural network, formed from its own subpopulations of neurons. The three neural networks are formed and evaluated in the domain at the same time as a team and the fitness for the team is passed back to all participating neurons

Yong and Miikkulainen [67] experimental setup was similar to that of Haynes and Sen [50], in that it used a grid-world with obstacles, three predators, and a single prey, where each agent was able to occupy a single grid space and was able to move in one of four directions at each simulation time step. The goal of any given pursuit-evasion scenario was for two or more predators to occupy the grid squares immediately surrounding the prey's position. The fitness of a predator team was calculated according to how close they were to the prey at the end of a given pursuit-evasion scenario. The authors' justification for using this fitness function was that the starting positions of the predators should not influence the team's fitness, and hence the time taken for predators to capture the prey was not taken into account. Certain experiments also incorporated communication into the behaviors of individual predators, where communication was defined as the capability of predators to see each other. Thus, neural networks controlling

either individual predators, or a whole predator team, took into account the coordinates of all predators in the team, in the derivation of cooperative pursuit strategies.

Comparative sets of experiment were executed, that is, those using both the centralized and decentralized approaches to neural network control, and for each of these experiments testing predator teams with and without communication were tested. Results clarified that the decentralized approach to evolution without communication derived predator controllers with specific functional roles, such as chasers and blockers, where each role contributed to the formation of a cooperative pursuit strategy. Thus, given that each predator performed its specific behavioral role, the team was able to effectively capture the prey even though there was no explicit form of communication to enable this cooperative behavior. Experiments using the decentralized approach for evolution of controllers with communication capability, produced teams with more flexible behaviors, although several different team-level behaviors emerged, where each lacked the composite forms of behavioral specialization evident in previous experiments, and as a result these team level behaviors performed worse as pursuit strategies. Specifically, evolution without communication placed strong evolutionary pressure on each predator to perform a particular role, though evolution with communication utilized variations and combinations of two or more emergent pursuit strategies, so it was not necessary for predators to adopt specific roles in order for a pursuit strategy to be successful. Experiments which test the centralized approaches, with and without communication capability, also resulted in the emergence of cooperative pursuit strategies, though these strategies performed poorly in comparison with the decentralized approaches.

The conclusion was that the distributed approach to the enforced subpopulations methodology for the incremental evolution of neural controllers, proved superior in terms of the time taken to evolve good pursuit strategies. Also, the distributed approach, without communication, allowed for the emergence of specialized behavioral roles, such that each sub-population was optimized for a specific function by the evolutionary process. The authors stated that adaptive niching in the evolutionary process facilitated the emergence of specialized behavioral roles. That is, as one sub-population started to converge to a particular behavior, other sub-populations that behaved in a complementary manner were rewarded and started to converge to other behavioral roles. In terms of the

domain implementation, having all predators develop behaviors that converged to complementary behavioral functions, contributed to the formation of effective cooperative pursuit strategies, and yielded a higher fitness for the predator team as a whole. The authors argued that the distributed enforced subpopulations approach was applicable to any problem domain that can be decomposed into a sequence of tasks of increasing complexity. Unfortunately this approach was not tested beyond the pursuit grid world environment using different configurations of predator starting positions, and obstacles in the environment.

The research of Denzinger and Fuchs [71] investigated the learning of cooperative pursuit behavior, which was achieved via the evolution of a set of appropriate prototypical situation-action pairs. The simulation environment made use of pursuit-evasion scenarios similar to those described by Haynes and Sen et al. [50] and Yong and Miikkulainen [67]. The environment was a grid-world containing three predators and one prey where the task of any given pursuit-evasion scenario was for at least two predators to position themselves in grids squares adjacent to the prey. Predator agent architecture was specified based on the classification of situations with the *nearest neighbor rule* and a learning mechanism that attempted to generate a set of prototypical situation action pairs. Predator behavior was derived from this of situation-action pairs in that, when a predator was confronted with a new situation it determined the situation-action pair that was most similar to the given situation in accordance with the nearest neighbor rule.

The predator then applied the action associated with the selected pair. The authors argued that this predator agent architecture provided a suitable basis for learning cooperative behavior given its flexibility. That is, a predator behavior could be readily changed via modifying, adding, or removing situation-action pairs. The learning of cooperative behavior was defined as searching for an appropriate set of situation-action pairs using a genetic algorithm. The genetic algorithm started with an unfit set of pairs, that is, those defining a set of poor pursuit behaviors (strategies) for a given set situations.

The fitness function defined a comparison procedure for sets of situation-action pairs so that the fitter set could be determined. The authors used several variants of the pursuit domain where each variant required differing degrees of cooperative behavior amongst

predators. The variants upon the game include changing the number of predators, the boundaries of the grid world, as well as the communication and observation capabilities of the predators. The goal of the experiments was to demonstrate the approach to be considerably versatile in that it allowed a designer of multi-agent systems to specify requirements in terms of representation of situations and possible actions, and then for a satisfactory solution to be evolved automatically.

The agent architecture was able to evolve effective cooperative pursuit strategies for many variants of the pursuit-evasion game, although for problems requiring more complex representations of situations, agent architecture able to operate in environments other than grid-worlds would be required. The use of a grid world allowed for the selection of distinct sets of situation-action values where a finite set of actions and resultant outcomes could be defined.

Nishimura and Takashi [71], in a variation on the pursuit domain, studied the emergence of cooperative behavior in the form of different types of flocking strategies using a more traditional style predator-prey system that contained large numbers of predators and prey. In this predator-prey system both predators and prey inhabited a simulated two-dimensional grid-world environment, and interacted through a series of pursuit-evasion game scenarios. The game scenarios used a score based system, and were implemented in the context of an artificial evolution algorithm. The rules of this particular pursuit-evasion game were such that when a predator moved to an adjacent grid-square behind a prey, the predator was awarded p points, whereas the prey lost p points, and when a predator moved to a grid square adjacent to a prey, and both were facing each other, both species lost p points. After receiving a score, individual predators and prey were categorized as either winners or losers. At the end of each generation the species with the higher score was able to reproduce more, and the species with the lower score reproduced less and was consequently diminished. Individual predators and prey in the system were characterized by a set of parameters that controlled their social interaction dynamics and behavior over the course of the evolutionary process. That is, behavioral interaction between predators and prey were formalized as a set of dynamical equations, where adjusting parameters of these equations helped to yield different individual and collective behaviors over the course of many generations. Since it was difficult for the authors to

know which parameter values were relevant to the formation of which types of cooperative behaviors, changes in dynamical equation parameters and state variables were taken into account by the evolutionary process. The offspring of "winner" individuals inherited the behavior of their parents in form of slightly modified algorithms or parameters. Mutation in the evolutionary process was simulated via the addition of a small level of Gaussian noise to random sets of parameters.

The authors executed several sets of experiments testing various pursuit evasion game scenarios and different parameter settings. For example, experiments were executed testing predators with adaptive behavior versus prey with fixed behavior, as well as predators with fixed behavior versus prey with adaptive behavior. From the first set of experiments, the authors observed that both predators and prey tended to group in loose probabilistic formations, and that there were certain random swarming dynamics that allowed both predators and prey to maximize their life expectancies. In the second set of experiments, where only the prey were adaptive, cooperative behaviors such as spatially disordered groupings, such as one-way marching, random swarming, lattice formation, or rotating clusters emerged. In the third set of experiments that introduced adaptive behaviors for predators and prey, similar though more complex forms of these collective behaviors emerged. From their experiments, the authors learned that the predators and prey were able to co-exist for the longest period of time when individuals of both species inherited a parameter responsible for the derivation of a cooperative behavior known as random swarming. In particular the random swarming formation in groups of predators, prevented predators from being able to synchronize their headings with groups of prey, and thus follow the same prey for extended periods of time. That is, over the course of the evolutionary process, the emergent random swarming formations decreased the chance of predators capturing prey, thereby minimizing the chance of extinction of both predators and prey. Given that the prey then had a lower probability of becoming extinct, they were more readily available to predators as a food source. Thus the predators also benefited from a reduced probability of extinction.

In conclusion to their research, the authors related their results to natural predator-prey systems, by stating that similar results have been found in theoretical biological studies conducted on shoaling fish [70, 71]. These studies also reported cooperative group

behaviors that were loose probabilistic formations. Though, even in such biological studies the relevance of, and mechanisms leading to, emergent cooperative group behaviors, has yet to be established. In terms of linking their own experimental results to results of studies in biological predator-prey systems, the authors suggested that uncertainty in predator-prey dynamics observed in their own experiments could encourage mutually beneficial coexistence via phenomena such as "symbiosis", as evident in certain biological predator-prey systems. That is, in natural systems spatially induced dynamic randomness is important for symbiosis, and in their own experiments the authors demonstrated emergent cooperative behaviors as an unstable dynamic without any explicit organization or structure. Although, interesting cooperative behaviors and a stable state in the system were attained by use of a finite set of equations running within an artificial evolution process, the key criticism of this research is that the evolved behaviors were limited by the grid-world environment and were somewhat devised by adjusting equation parameters prior to the execution of each evolutionary process.

2.5 Summary

In conclusion, this chapter describes background information and literature review related to cooperative multi-agent learning and emergent cooperation. The most relevant research examples were selected and reviewed. These examples use biologically inspired design principles as a means of motivating multiple agents to collectively solve a pre-defined problem that could not otherwise be solved by an individual agent. The related research examples in pursuit domain were identified and selected based on results where emergent cooperative behavior had been achieved using biologically inspired design methodologies which made use of concepts such as self-organization, learning, and evolution.

It is evident from the literature that the use of various forms of simulated artificial systems is considered by many researchers to be an effective approach in investigating emergent cooperation. Such simulations provide a means for studying conditions under which cooperation emerges, and the effects of parametric changes can be seen in a relatively short amount of time.

Chapter 3
Evolutionary Computation

Some of the most sophisticated existing programs are not human coded, they evolved naturally. They can be found inside the cells of living creatures in the form of DNA. These programs were *coded* by the process of natural selection as described by Charles Darwin and are responsible for the highly optimized systems that we see around us. It is not surprising therefore that the AI community has been active in trying to make use of the power of natural evolution to build both hardware and software. Evolutionary Computation (EC) is emerging as a new engineering computational paradigm. It is a search technique, which uses concepts and mechanisms of Darwinian evolution and natural selection to solve problems in many fields of engineering and science. The purpose of this chapter is to introduce EC in general and genetic network programming (GNP) in particular.

3.1 Evolutionary Algorithms

Evolutionary algorithms are a family of population based search algorithms that simulate the evolution of individual structures by interrelated processes of selection, reproduction, and variation. There is a variety of EAs that have been proposed and studied. They all share a common set of underlying assumptions but differ in the breeding strategy and representation on which EAs operate.

Strong resemblance to biological processes as well as their initial applications for modeling complex adaptive systems influenced the terminology used by EA researchers. It borrows a lot from genetics, evolutionary theory and cellular biology. Thus, a candidate solution to a problem is called an *individual* while an entire set of current solutions is called a *population*. The actual representation (encoding) of an individual is called its *genome* or *chromosome*. Each genome consists of a sequence of genes, i.e. attributes that describe an individual. When individual solutions are modified to produce new candidate

solutions they are said to be breeding and the new candidate solution is called an *offspring* or a *child*. During the evaluation of a candidate solution, it receives a grade (value) called fitness, which indicates the quality of the solution in the context of a given problem. When the current population is replaced by offspring, the new population is called a new *generation*. Finally, the entire process of searching for an optimal solution is called *evolution*.

Historically, four major EAs have been developed: evolution strategies (ES), evolutionary programming (EP), genetic algorithms (GAs), and genetic programming (GP). These algorithms have been mostly used to evolve solutions to parameterized problem domains. GA has the genome of string structure, while the genome in GP is the tree structure. Therefore GP can be applied successfully to many real problems where complicated programs are to be constructed to solve the problem. On the other hand, the fourth major EA developed recently, genetic network programming (GNP) [7], has been used to evolve computer programs to solve computational tasks and has the genome of graph (network) structure.

From the engineering point of view, EC can be understood as a search and optimization process in which a population of solutions undergoes a process of gradual changes. This process depends on the fitness (a formal measure of perceived performance) of the individual solutions as defined by the environment (objective function).

A typical evolutionary algorithm consists of the following steps:

1. Initialize the population
2. Evaluate all members of the population
3. While the termination condition is not satisfied

{

3.1 Select individual(s) in the population to be parent(s)

3.2 Create new individuals by applying the variation operators to the parent(s)

3.3 Evaluate new individuals

3.4 Replace some/all of the individuals in the current population with the new individuals

}

This Algorithm works as follows. Before an actual evolutionary process begins, an initial population of individuals (solutions) is created. Traditionally, the initial population is created *randomly* but several other initialization techniques have also been used (e.g. starting from a set of previously known or arbitrarily assumed solutions). Next, each individual in the initial population is evaluated and assigned a *fitness* value. Using the fitness values, the selection mechanism chooses a subset of the current population as parents to create new individuals. When the selection mechanism uses bias toward individuals with better fitness, the created offspring will, more likely, have higher fitness. Once the set of parents has been selected, the new individuals are created by copying them and applying variation operators.

There are several commonly used selection strategies within EC community. *Fitness proportionate selection* (roulette wheel selection) normalizes the fitness values of all individuals in the population and assigns these normalized values as probabilities that their respective individuals will be selected. *Ranked selection* works by first ranking all individuals in the population by their fitness, and use these ranks, rather than actual fitness values, to determine selection probabilities of the individuals. The most popular selection strategy is the *tournament selection* which is used in this thesis. In this strategy, a pool of k individuals is picked at random from the population. Each of the individuals in the pool is selected independently and it might be the case that the same individual will be selected multiple times. Next, an individual from the pool with highest fitness value is selected to form the new population. This procedure is repeated as many times as necessary to create either an entirely new population or a subset of it. Selection pressure is easily adjusted by changing the tournament size. If the tournament size (k) is larger, weak individuals have a smaller chance to be selected.

The two most popular variation operators are *mutation* and *crossover*. Mutation acts on a single individual and works by applying some variation to one or more genes in the individual chromosome. Crossover, on the other hand, operates on multiple individuals (usually two) and combines parts of these individuals to create new ones. The newly created individuals are evaluated and assigned fitness values. After variation process, either all or only a subset of the current population is replaced by these new individuals.

The steps 3.1 through 3.4 of the EA are repeated until an assumed stopping criterion is met. This criterion is usually defined as reaching a predefined number of generations or obtaining an individual with satisfactory fitness. This model of GA is called *generational* (standard) genetic algorithm.

3.2 Genetic Programming (GP)

Genetic Programming is technique for automatically programming computers. It searches the space of possible computer programs to find suitable programs to solve the desired problem. Koza [5] suggested that a tree structure should be used as the program representation in a genome to overcome the problems of GA. Koza [5], however, was the first to recognize the importance of the method and demonstrate its feasibility for automatic programming in general. In his 1989 paper, he provided evidence in the form of several problems from five different areas.

The tree type genome of typical GP is shown in Figure 7. GP consists of one root node and a number of non-terminal nodes and terminal nodes, where non-terminal nodes are used as functions such as arithmetic function, Boolean function, and conditional statements and so on. Terminal nodes contain the inputs to the GP program which are used for a particular processing which depends on the problems concerned. GP can be used to generate the behavior sequences of dynamic agents in which agent action sequences can be obtained by processing the nodes of the tree starting from the root node.

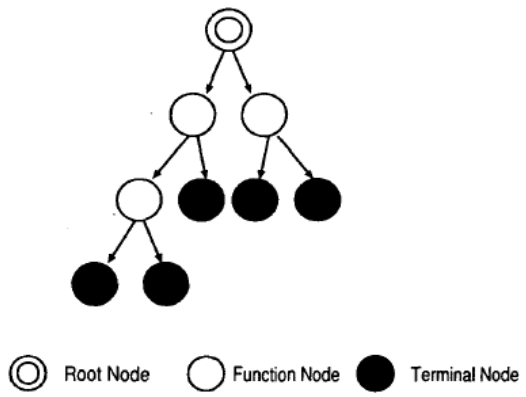

Figure 7. Typical tree structure of a GP program.

There two genetic operators, which are used in GP. These operators are *crossover* and *mutation*. Figure 8, shows GP mutation which is to select a point in the tree randomly and replacing the existing sub-tree at that point with a new randomly generated sub-tree.

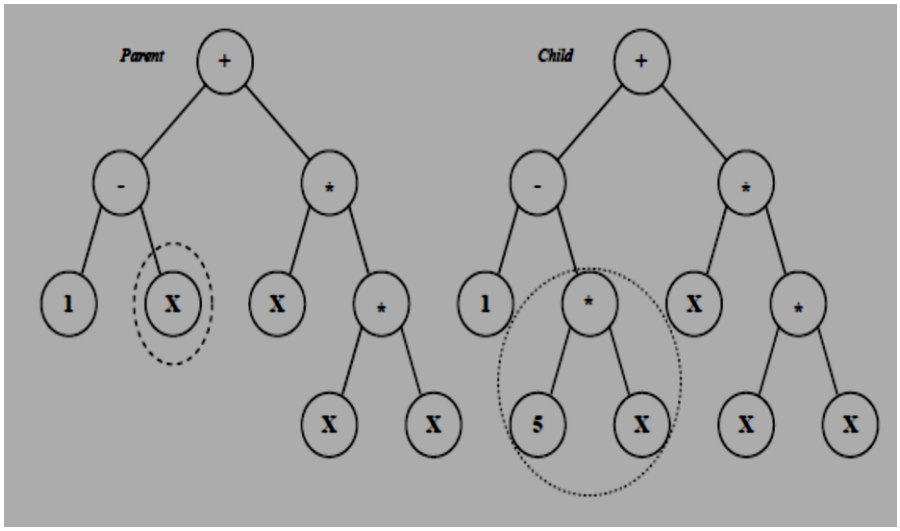

Figure 8. GP mutation

Figure 9, shows GP crossover which is to swap the selected subtrees between the two parents. GP evolves a population of programs in parallel. The driving force of this simulated evolution is some form of fitness-based selection. Fitness-based selection determines which programs are selected for further improvements

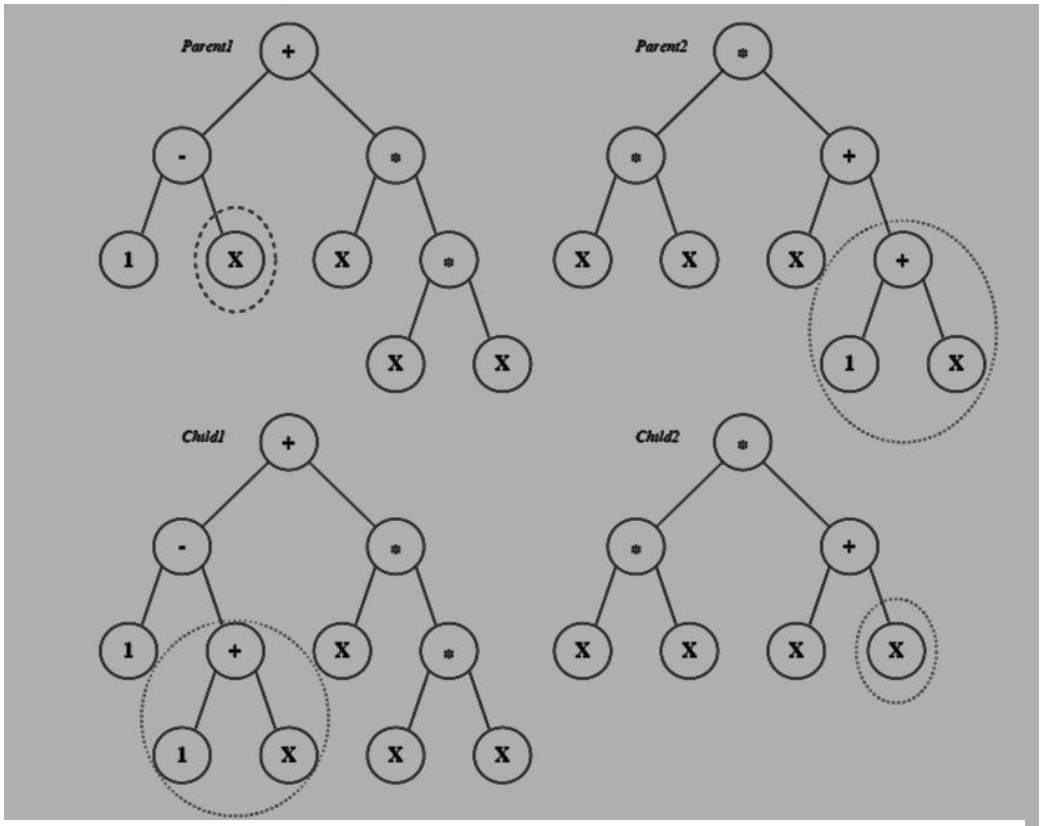

Figure 9. GP crossover

3.3 Genetic Network Programming (GNP)

In this section, Genetic Network Programming (GNP) is explained briefly. GNP is an extension of GP in terms of gene structures. The original idea is based on the more general representation ability of directed graphs than that of trees. Compared to GP, GNP can find solutions for problems without bloat because of fixed number of nodes in GNP. Bloat is a problem, which is highlighted by GP, and it is the uncontrolled growth of the average size of an individual in the population, and it will be explained in detail in section 3.5. In addition, GNP has built-in memory functions which implicitly and preserve agent's actions as a chain of events and makes it a better approach to automatically program agents.

3.3.1 Basic Structure of GNP

The basic structure of GNP is shown in Figure 10. An individual of GNP has a directed graph structure which is consisted of three kinds of nodes: "Initial Boot node", "Judgment nodes" and "Processing Nodes". In Figure 10, the initial boot node, the judgment node and the processing node are denoted by square, diamond and circle respectively. The initial boot node is the node which is executed only when GNP starts. Initial boot node is an origin for node transitions and has no functionality. The Processing node is the smallest unit describing agent's actions to environment whereas the judgment node is the smallest unit describing how to judge the environment the agent senses. Each judgment node has several judgment results, which corresponds to the number of its outgoing branches. Each processing node has only one outgoing branch. Judgment nodes and processing nodes have their "Function Label" in a library which is prepared by the system designer in advance. If we assume there are P types of processing nodes, their function label varies from 1 to P and when there are J types of judgment nodes, their function label varies from 1 to J.

3.3.2 Chromosome Representation

As shown in Figure 10, GNP can be illustrated by its "Phenotype" and "Genotype". Phenotype shows the directed graph structure and genotype provides the chromosome of a GNP individual which is a set of the gene of nodes. In Figure 10, NID_i shows the gene of node i in which all variables are integers. NT_i describes the node type, $NT_i = 0$ when the node i is initial boot node, $NT_i = 1$ when the node i is a judgment node and $NT_i = 2$ denotes a processing node. NF_i describes the function label (Judgment node: {1,2,…,J}, Processing node: {1,2,...,P}). C_{ik} indicates the node number to which node i's k-th branch is connected. d_i means the delay time needed for processing and judgment at node i and d_{ij} means the delay time for moving from node i to j. These delay times have been introduced to model GNP like human brain, which needs the time for thinking. When the accumulated time delay exceeds the "Time Delay Threshold Value" which is a determined in advance, GNP fails in operating in the desired time [8].

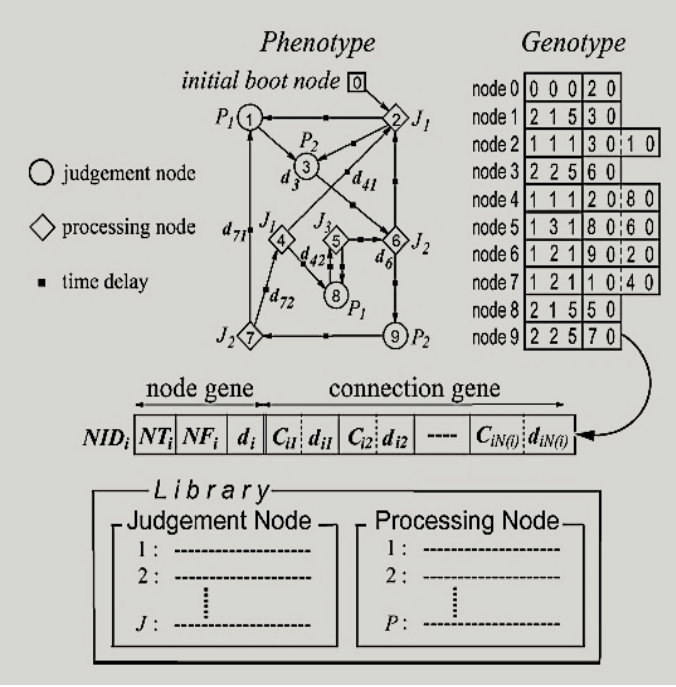

Figure 10. Basic structure of GNP

3.3.3 Genetic Operators in GNP

Selection, mutation and crossover are carried out as genetic operators in GNP.

1) *Selection*: Selection is the operation of selecting an individual which serves as parents of the next generation, according to the degree of fitness. The fitness shows the quantity of how each individual adapt itself to environment.

In GNP, the commonly used selection operators in evolutionary computations can be used and their algorithms are the same as conventional evolutionary computations. In this research, "Tournament Selection" and "Elite Preservation" are used.

2) *Mutation*: Mutation operates on only one network and new offspring is generated as follows.

- A network is selected using selection method.
- Some branches are selected randomly for mutation with the probability P_m.
- The selected branches are changed randomly and new offspring is generated.

Figure 11, shows an example of mutation in GNP.

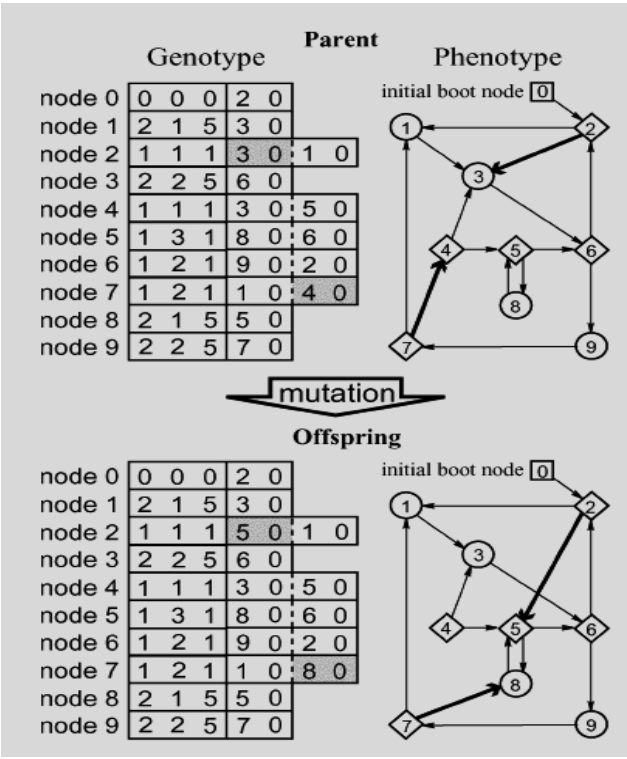

Figure 11. GNP Mutation

3) *Crossover:* Crossover is executed between two networks called parents and generates two offspring as follows.

- Two parent individuals are selected using selection method.
- Corresponding nodes with the same node number are selected as crossover nodes with probability *Pc*.
- Two parents exchange the selected corresponding nodes having the same node number and two new offspring are generated.

This type of crossover is called *uniform crossover*. Figure 12, shows an example of uniform crossover in GNP.

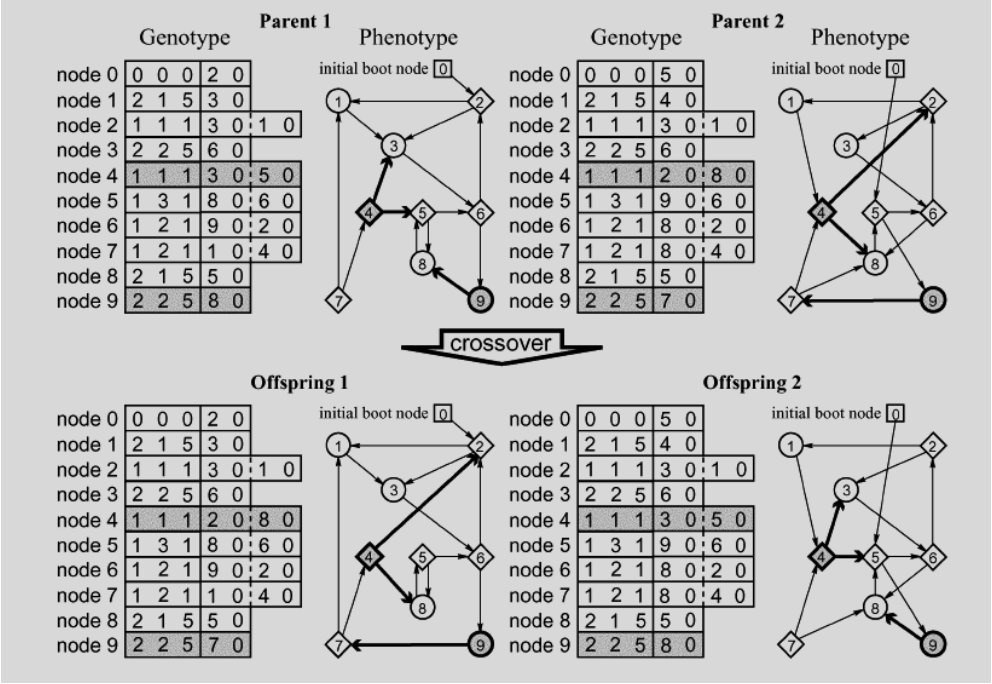

Figure 12. GNP crossover

3.3.4 Algorithm of GNP

The algorithm for GNP evolution process is explained in following and it is illustrated in Figure 13.

1) **[Initialization]**: Initialize the population with N_p randomly generated individuals.

2) **[Fitness evaluation]**: Calculate the fitness value of each individual and find an elitist.

3) **[Genetic operations]**:

3-1) Selection: Select parents for crossover by the tournament selection.

3-2) Crossover: Apply the crossover operator to the selected parents.

3-3) Mutation: Apply the mutation operator to the connection gene.

3-4) Elite strategy: Preserve the elitist individual found in Step 2 or Step 5.

4) **[Replacement]**: Replace the newly generated population with the previous one.

5) **[Fitness evaluation]**: Calculate the fitness value of each individual and find an elitist individual.

6) **[Termination condition]**: Terminate the algorithm if the specified condition is satisfied. Otherwise return to step 3.

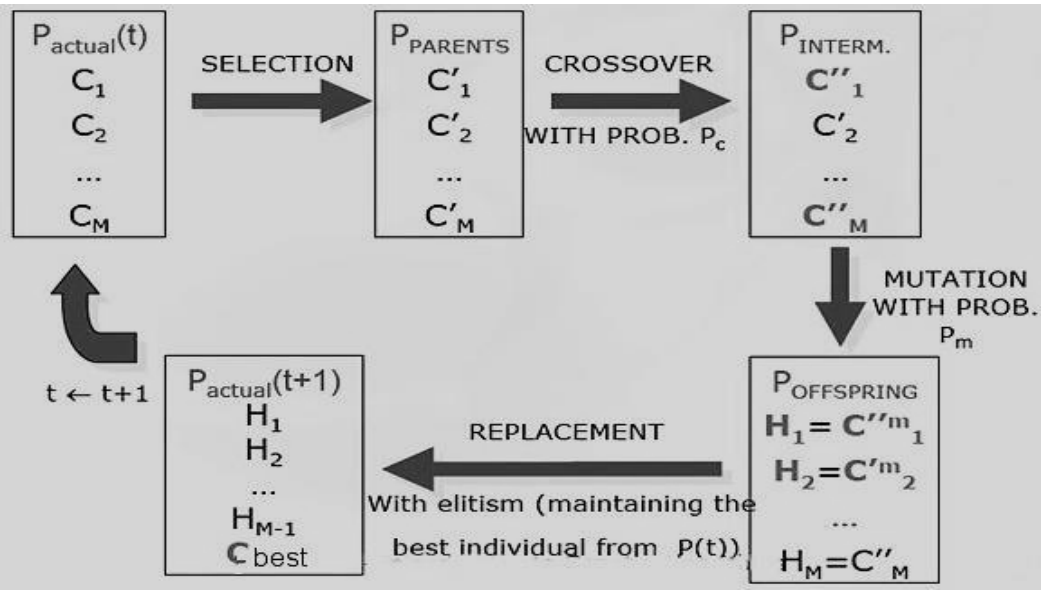

Figure 13. GNP algorithm

3.4 Evolutionary Computation and Engineering Design

Recently many studies have been made on automatic design of the complex systems by using EA techniques. Much of the work on engineering applications has taken place using EA algorithms for design *optimization*.

The three main issues in applying EAs to an engineering design problem are:

1. Selecting an appropriate representation for engineering designs.

2. Defining efficient genetic operators.

3. Providing an adequate evaluation function for estimating the *fitness* of generated solutions (points in the search space).

An appropriate representation of an engineering system is one of the most crucial elements of evolutionary design. The process of creating an efficient and adequate representation of an engineering system for evolutionary design is complicated and involves elements of both science and art.

One has to take into account not only important aspects of understanding traditional modeling of an engineering system, but also relevant computational issues that include search efficiency, scalability, and mapping between a search space (genotypic space) and a space of actual designs (phenotypic space).

Appropriate choice and implementation of genetic operators, i.e. mutation and crossover operators, and careful tuning of their rates is an important issue as it can have a big impact on the success of EAs and has therefore been a subject of both theoretical [38] as well as experimental investigations [33, 34, 35]. Genetic operators are primary sources of *exploration* in EAs. On the other hand, selection mechanisms provide EAs with *exploitative* power. Thus, by properly defining and controlling the variation mechanisms (genetic operators), one can achieve a higher level goal of finding ‘‘an effective balance between further exploration of unexplored regions of the search space and exploiting the regions already explored’’ [42].

Another important issue in successful application of EAs is how to choose an adequate fitness evaluation function for a problem domain. Evaluation functions provide EAs with feedback about the fitness of each individual in the population. EAs use this feedback to bias the search process in order to improve the population’s average fitness. Naturally, the details of a particular fitness function are problem specific.

3.5 Evolving Intelligent Agent using GP and GNP

A large number of studies have been made on automatic design of behavior sequences for agents, such as the design of the behavior sequence to carry out some tasks in the virtual world, the experiments of creating artificial life aiming to realize the behaviors of ants or fishes, the planning for real mobile robots which have a simple object in the real world, and so on [34, 35]. Many models to express such behavior sequences for agents have been proposed, and they have used evolutionary optimization techniques such as Genetic Algorithm (GA), Evolution Strategy (ES), Evolutionary Programming (EP) and Genetic Programming (GP) to acquire their desired structures. As a rule, the planning of behavior sequences for agents takes enormous searching time, nevertheless the acquired plan have poor ability in terms of adjustability in dynamic environments. But, the real world is in a dynamic environment of high dimensions. So, it is highly expected to develop a method of planning that can achieve given missions efficiently regardless of their surrounding environments. A great deal of efforts has been made on the planning of behavior sequences especially by using GP and EP. The characteristics of these methods are trying to generate behavior sequences making much of 'what should the agent do now' using all surrounding circumstances. On the contrary, in GNP, the planning of the behavior is regarded as a sequential stream and makes much of 'how have the agents behaved or what have the agents judged up to this time?' using only some information which seems to be needed at the time. Therefore, it seems that the proposed method is likely to more resemble the functions of the brain of real creatures than other methods.

3.6 Comparison between GP and GNP

GP's enormous freedom of representation is a mixed blessing. With such a huge search space, an algorithm might have to search for a long time. Therefore, it is generally said that GP is sometimes difficult to search for an optimal solution, because the searching space for solutions become enormous, due to a phenomenon called *bloat* which is the main reason that searching efficiency of GP is not very high in most of cases.

Bloat is a problem, which is highlighted by GP, and it is the uncontrolled growth of the average size of an individual in the population. In other words, after some generations (typically below one hundred), the search for better programs halts as the programs

(trees) become too large and searching space becomes enormous. The drawbacks of bloating are as follows:

- Bloated trees occupy a large amount of memory.
- Bloated trees are computationally intensive and slow down the system.
- Bloated trees are often more difficult to modify in meaningful ways.

The most common approach on how to deal with bloat in tree-based genetic programming individuals is to limit their maximal allowed depth. Also, a common alternative to limit the depth is to punish individuals in some way based on excess size. On the other hand, genome structure in GNP is a network which enables GNP to construct problem-oriented compact genome. Given the network structure in GNP too many transition functions need not be installed due to the fact than an appropriate number of problem-oriented judgment nodes and processing nodes are set in the network. Therefore, the bloat phenomenon is completely eliminated in the GNP approach.

3.7 Summary

This chapter introduced evolutionary algorithms and described how they work. First, GP was introduced as the most popular example in evolutionary algorithms. Then, GNP was presented as a newly developed evolutionary algorithm based on GP and its advantages over GP such as search efficiency and searching without bloat. It was shown that we can see GP and GNP from two different perspectives. From a pure machine learning perspective they are general techniques that allow a machine to learn how to solve a given task from training examples (the test cases) and from an engineering design perspective they can be used for automatic design of behavior sequences for agents in a multi-agent environment.

Chapter 4
Pursuit Problem

The pursuit domain was introduced by Benda et al. [42]. Over the years, researchers have studied approaches on several variations of its original formulation. This domain is an appropriate environment to illustrate MAS since it has been studied using a wide variety of approaches and it has different instantiations that can be used to illustrate different multi-agent scenarios. Since it involves agents moving around in a world, it is particularly appropriate as an abstraction of robotic MAS. The pursuit domain is not presented as a real-world domain, but rather as a toy domain that helps concretize complex concepts in real MAS.

4.1 Introduction

The pursuit problem (also called the Predator/Prey problem) is a well-studied benchmark for Distributed Artificial Intelligence (DAI) research. Benda et al [42] formulated the original problem definition as consisting of four *predators* that need to capture a single *prey*. The predators and the prey are situated on a two dimensional world consisting of cells (Figure 14). Movement in this world, is possible in orthogonal directions only, and takes the agents from one cell into another (Figure 15a). The goal for the predators is to move so as to *capture* the prey (Figure 15c). To capture the prey, the predators must occupy all cells immediately adjacent to the prey (which we call *capture positions* Figure 15b). Many variations to the original definition have been devised by changing key domain parameters, these parameters are described in below and listed in Table 1.

Capture condition: In its original setting, a prey is captured when all its adjacent cells are occupied by predators. Other possible capture criteria are surrounding the prey with two predators or occupying the same cell as a prey.

Visible range: This denotes the number of cells from which a predator receives sensory information. Preys and predators that are outside this range are not visible.

Communication: An important variation is whether the predators are allowed to communicate with each other and in this way be able to inform other predators of their strategies or sensory findings.

Legal moves: Originally, an agent was only allowed to move to adjacent cells. A possible variation is to also include diagonal movements.

Grid size: The size of the world can be changed to different sizes. Furthermore, the world can be made planar (with borders on all edges) or toroidal. In the latter case the agents can directly move from one side of the grid to the other side.

Simultaneous or sequential movement: This variation indicates whether the agents move at the same time or one after the other.

Prey movement: In most variations the prey moves randomly. Other variations would allow the prey to be more sophisticated and actively try to escape capturing

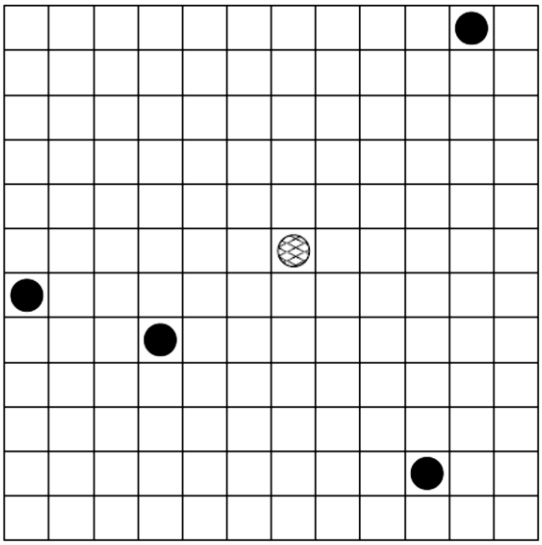

Figure 14. Pursuit domain consists of a two dimensional (usually toroidal) world in which four predators must try to capture the prey by surrounding it.

Table 1. Pursuit Domain Parameters

Parameters	Examples
Grid size	finite, infinite
Grid shape	square, hexagonal, continuous, toroidal, edged
Capture definition	prey surrounded, predator on same square as prey
Legal moves	orthogonal only, diagonals permitted
Movements	agents take turns to move, agents move simultaneously
Sensors	what can be sensed, by whom and from how far away
Predator communication	permitted, disallowed, range, implicit, explicit
Prey movements	random, stationary, deterministic
Predator/Prey number	1/4, 2/6

In the pursuit domain usually the prey moves randomly, and is slower than the predators (usually implemented by ensuring the prey is stationary some percentage of the time). In most of the implementations, only one agent may occupy a cell at a time, and therefore conflicts can occur if one or more agents try to occupy the same cell (in Figure 15d agents 0 and 1 are in conflict).

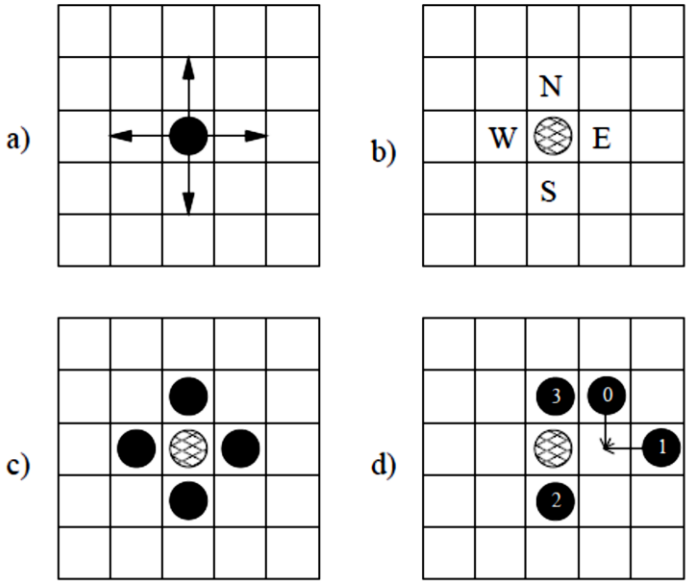

Figure 15. Pursuit domain rules; a) movement is restricted to orthogonal directions, b) capture positions are cells immediately adjacent to the prey, c) the prey is captured when all capture positions are occupied by predators, d) conflicts occur when one or more agents try to occupy the same cell.

The pursuit domain with different grid shapes and capture definitions are illustrated in Figure 16.

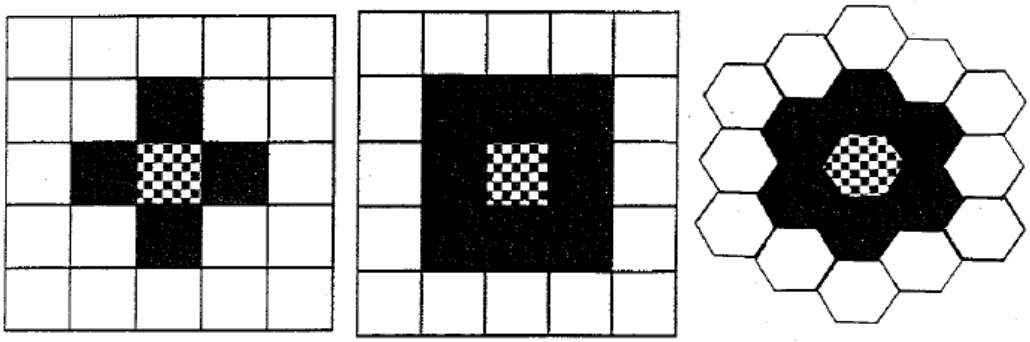

Figure 16. Pursuit domain with different grid shapes and captures definitions

Several researches have been conducted in the pursuit domain with different methodologies. But the one which is related to our research is the one which was performed by Haynes et al [50, 51]. They used GP to evolve homogeneous predator agents for the pursuit domain. The agents in their version of the pursuit game move concurrently. Agents that try to move to the same cell are bumped back to their original position. A predator may push another predator out of the cell which it occupies if it decides not to move. However this push operator is not available in the function set presented to the GP system. This form of conflict resolution therefore is executed by the environment rather than the agents. They use a 30 x 30 toroidal, orthogonal game board, in which the agents can move orthogonally or stay still, and randomly generated training test cases consisting of the prey in the center and the predators placed randomly. The prey moves at 90% of the speed of the predators. They used a STGP (Strongly Typed Genetic Programming) system to allow functions and terminals of different types to be combined in one tree, which they argue reduces the search space. They demonstrated that the latter is superior to standard GP applied to this domain through comparison experiments.

Their trees return the direction to move, which can be any of North, East, South, West or Here (random instances of which are generated as constants in the terminal set). Here indicates that the agent is to stay put. They provide a function called *CellOf* which takes two parameters, the first being an agent, and the second a compass direction (tack in their

terminology), and return the cell that is positioned in the specified tack relative to the cell occupied by the specified agent. The terminal set includes a reference to the prey and current predator. Other functions that they use include *IfThenElse*, and *MD*, where *MD* returns the Manhattan Distance (MD) between the two cells that are provided as arguments. The fitness function that they employ awards agents for getting close to the prey, with further bonuses for occupying capture positions, and capturing the prey. They used 100 time steps in their training.

4.2 Simulation environment

In order to carry out the experiments in this thesis, a well-known software package named Pursuit_package-0.9 [44] is used which is an open source package and it is available online. This package which is developed in 2004 by Jelle Kok [44], simulates the pursuit domain and runs under Linux. Figure 17, shows a snapshot of the pursuit domain implemented by this package in which the size of the environment is 10*10. Traditionally, the predators are blue and the prey is red.

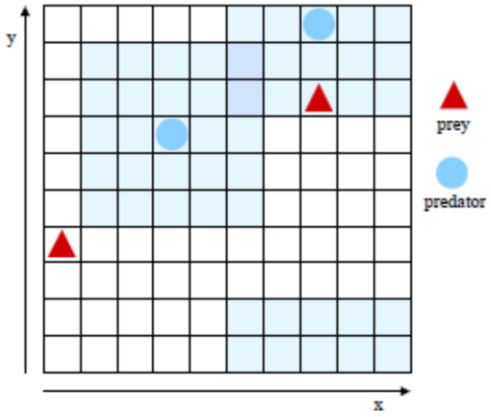

Figure 17. Pursuit package environment

The pursuit package consists of the following components:

1. **Pursuit server:** This is the core program of the package. It models the complete environment and handles the connections with the client programs.
2. **Monitor:** The monitor can be used to display the current world state of the pursuit server. After a monitor is started, it creates a connection with the server and then

starts to receive information about the current world state. The monitor visualizes this information.

3. **Agents**: The agents (both predators and prey) also create a connection with the server and then start to receive sensory information. Furthermore, they can sent actuator commands to the server, which causes the world to be updated accordingly.
4. **Logplayer:** If the related option is turned on, the pursuit server logs all the subsequent world states to a file. The logplayer can be used to replay the contents of this file. A monitor is needed to visualize the current contents of the logplayer.

The agents (either predators or preys) can be started by connecting them to the server process. A skeleton implementation is shown for both a predator and prey agent in Table 2. It takes care of all communication with the server and defines several callbacks functions in which the behavior of the agent can be defined.

Table 2. The client structure showing the response to different server messages

```
{Client structure}
continue = true
while continue == true do
        receive message from server
        if message == (quit) then
                continue = false
else if message starts with (see then
        processVisualInformation()
if determineCommunicationCommand() is not the empty string then
        send communication message
end if
else if message starts with (hear then
        processCommunicationInformation()
else if message starts with (send action then
        determineMovementCommand()
        send movement command to the server
end if
end while
```

In this thesis each agent has its own independent process and there is one *identical* agent per predator. So they have identical capabilities and decision procedures. Agents have (the same amount of) limited information about other agents' internal states. These kinds of agents are called homogeneous. The pursuit domain with homogeneous agents is illustrated in Figure 18.

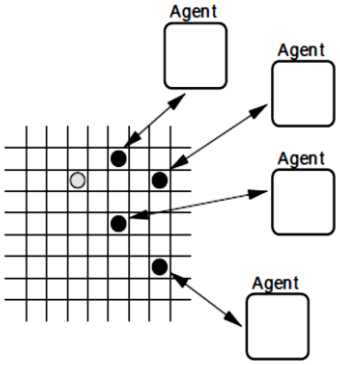

Figure 18. The pursuit domain with homogeneous agents

On the other hand, there is heterogeneous pursuit domain. As in the previous scenario, the predators are controlled by separate agents. But they are no longer necessarily identical agents: their goals, actions and domain knowledge may differ. The pursuit domain with heterogeneous agents is illustrated in Figure 19.

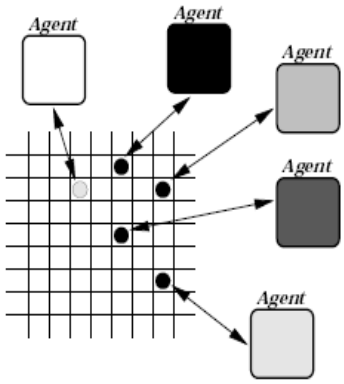

Figure 19. The pursuit domain with heterogeneous agents

4.3 Conflict and Conflict Resolution

In the pursuit domain conflicts (collision) arise when one or more agents try to occupy the same cell. If such conflicts are not resolved, then they can result in *deadlock*. Just as in concurrent and distributed systems, there are two ways of handling deadlocks in the pursuit domain; *deadlock avoidance*, and *deadlock detection and resolution*. Deadlock avoidance is the simplest and most effective approach that we can employ. Here the predators will need to coordinate their movement to eliminate or minimize conflicts. Deadlock detection and resolution is far more complex. Agents in conflict must first be able to detect the conflict and then coordinate their behavior so as to resolve it. We note that in much of the research above, conflict resolution has been avoided by providing environmental mechanisms for resolving conflicts or avoiding them. Examples include the ordered execution of moves used by Stephens [72] and Korf [43], and the implicit push operator used by Haynes et al [51].

In our experiments in this thesis we used the same conflict resolution mechanism as Hayens et al [51] used. All agents choose their action simultaneously, the world is accordingly updated by using some conflict resolution, and the agents choose their actions again based on the updated world state. Conflict resolution means, we do not allow to agents to co-occupy a position. If two agents try to move to the same location (square) simultaneously, they are bumped back to their prior positions. One predator, however, can push another predator (but not the prey) if the latter decided not to move. These three conflict conditions are illustrated in Figure 20.

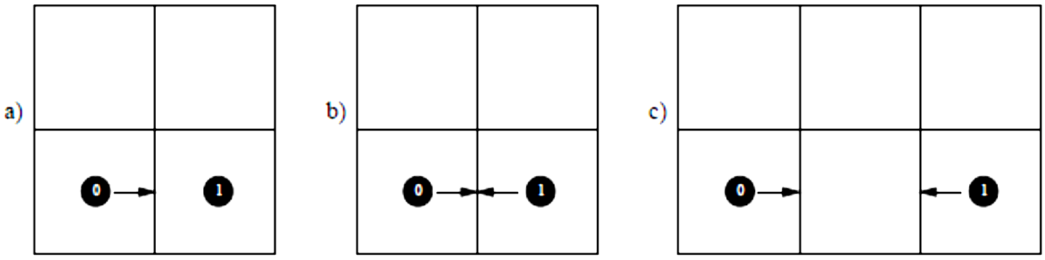

Figure 20. Conflict conditions: a) Agent 0 tries to move to a cell occupied by agent 1, b) Agents 0 and 1 try to exchange positions, c) Agents 0 and 1 try to move to the same cell.

4.4 Summary

In this chapter first we described the pursuit (also called predator/prey) problem in detail. Then the Pursuit_Package [44], used for simulating the multi-agent experiments in this thesis was presented. This software is an implementation of the pursuit domain. It is has a client-server architecture and is run under the Linux operating system. Many aspects of the package are configurable, making it possible to test different variations of the pursuit problem. Finally the concepts of conflict (collision) and conflict resolution in pursuit domain are described and illustrated.

Chapter 5
The Proposed Methodology for Pursuit Problem

The pursuit problem described in the previous chapter is a well-known domain in the multi-agent systems (MAS) community and the cooperation of predator agents is highly required to achieve their goal. Moreover, it is known as an easy-to-describe but difficult-to-solve problem and since many researches have conducted their experiments in this domain, it is an appropriate benchmark to evaluate and compare different solutions.

In this chapter, we design and develop a methodology using a relatively new evolutionary computation technique called Genetic Network Programming (GNP) to automatically evolve teamwork and cooperation among predator agents in the pursuit domain in order to capture the prey agent as soon as possible. The results of our approach are presented in the next chapter and are compared to those of Genetic Programming (GP), the most popular evolutionary approach.

The use of evolutionary computation to exploit emergent cooperation is a relatively unexplored area of research in the pursuit domain and GNP has been only tested in a few specific domains; its performance and success need to be examined in other various domains. Therefore, in this thesis we have tried to apply GNP to evolve cooperation and teamwork among agents in order to address this research gap.

5.1 Design Phases

This section demonstrates the design of our proposed GNP-based system to solve the pursuit problem. It is worth mentioning that there is one leaner involved and it discovers the behaviour for all predator agents, thus our approach is categorized as team learning. Furthermore all the predators have identical GNP programs (graphs), therefore they are homogenous and it concludes that we are designing a homogenous team learning methodology. There are four major phases required to design a GNP based methodology for a specific domain:

1) Designing a suitable fitness function,
2) Assigning appropriate values to the running parameters,
3) Choosing a suitable termination condition
4) Designing appropriate judgment and processing nodes for the GNP agent architecture.

These steps will be presented respectively in the following sub-sections.

5.1.1 Fitness Function

The fitness function should be defined in such a way that will separate better individuals from worse by assigning suitable fitness values. Thus defining an effective fitness function specifically designed for a domain is a crucial part in designing a GNP-based methodology.

Table 3 illustrates the fitness evaluation pseudo code for the pursuit problem as it is used in this thesis. After each cycle we measure the *Manhattan Distance (MD)* between each of the agents and the prey - the sum of which is added to the current fitness value. After each simulation we provide further rewards for each agent still occupying a capture position, and add yet a further bonus if the prey is captured. Therefore, this fitness function rewards movement close to the prey, with bonuses for occupying capture positions and a further bonus for capturing the prey. Manhattan distance (MD) between two agents A1 and A2 is defined as follows: if agent A1 is in the square ($x1$, $y1$) and

agent A2 is in square ($x2, y2$) then their MD is calculated by the following formula: $MD(A1, A2) = |x2 - x1| + |y2 - y1|$.

Table 3. Fitness Evaluation Pseudo code for the Pursuit Problem

```
initialise the positions of the agents according to the test case
fitness = 0
for cycle = 0 to MaxCycles
 move the predators according to the evolved code
 if the prey is not stationary move the prey (only 90\% of the time)
 check to see if the moves are valid
 undo invalid moves
 commit valid moves
 foreach predator p, fitness += gridWidth / mahattenDistance(p, prey)
endfor
fitness += MaxCycles * gridWidth * NoOfCapturePositionsOccupied
if the prey is captured
  fitness += MaxCycles * gridWidth * 4
```

5.1.2 Running Parameters

Running parameters are those which control the way the evolutionary process of the GNP system is tuned and should be initialized before the program starts running. They are one of the most important stages in designing a GNP-based system, and their value depends on the domain and the specific addressed problem. If these values are not well defined the evolutionary process will not converge to an optimal or sub-optimal solution. So the GNP evolutionary process should be calibrated with appropriate values for these parameters and the only way to reach reasonable values is by the trial and error method.

The values for the parameters used in this thesis are shown in Table 4. It should be mentioned that we were successful in obtaining reasonable results with these values yet are they optimal and could we get better results with other values? This still remains an unanswered question and further research is required.

In Table 4, the term *elite* signifies the fittest individual of a population. GP and GNP usually use *elite preservation strategy* to preserve a number of best individuals created in a population and directly transfer them to the next population. In this thesis we merely keep one elite individual and transfer it directly to the next generation without, any changes to its chromosome.

Table 4. Parameter specification

Parameter	***Value***
Population size	200
Tournament size	5
Elite size	1
Number of generations	50
Crossover probability *Pc*	0.9
Mutation probability *P*m	0.01
Number of nodes per kind	1
di* and *dij	0

5.1.3 Termination Condition

The evolutionary process of a GNP system should terminate somewhere. In order to have an efficient system, a good termination condition is needed to stop the algorithm. A Termination condition can be chosen by one the following criteria depending on the environment and design of the GNP system.

1. The algorithm stops when an individual is found to have a fitness value more than the predefined threshold.
2. The algorithm stops when the number of generations reaches a predefined number.

In this thesis we have chosen the second criterion based on our environment and other researchers' experiences. For Instance, our evolutionary process stops when the number of generations reaches 50 and it stops after 100 generations in experiments where there is no conflict resolution mechanism.

5.1.4 Agent Architecture

As mentioned earlier, the GNP-based agents have graph architecture. Other than the initial boot node (IBN) which is common in all GNP graphs, each GNP graph has its own unique set of judgment and processing nodes depending on the problem and the way the GNP system is designed to solve that problem. In this subsection we are going to explain the judgment and processing nodes and their characteristics used in our GNP graph for the pursuit domain experiments.

- **Processing nodes**

The processing nodes are *MN, MS, ME, MW* and *SH*. These nodes determine actions which predator agents can execute in the environment. When *MN* (Move North) is executed, the agent moves north, when *MS* (Move South) is executed, it goes south. *ME* (Move East) is for moving east and *MW* (Move West) is for going west. In addition when *SH* (Stay Here) is carried out, the agent stays still and does nothing. Each processing node has only one outgoing branch. Therefore, when the processing nodes are executed, the next node is uniquely determined in accordance with their connection.

- **Judgment nodes**

The judgment nodes are CNC, CSC, CEC, CWC and FP. These nodes act as decision functions dealing with the specific inputs from the environment. CNC (Check North Cell) checks the north cell of the predator agent and has three judgment results (Floor, Predator, and Prey). CSC, CEC, CWE are for checking south, east and west cell of the predator agent respectively and they have three judgment results as well. FP (Find Prey) finds the approximate direction of the prey with respect to the predator and has four judgment results (North, South, East, West). Note that in our design, the function label numbers for processing and judgment nodes is the order in which they appear in Table 5, e.g., the function label of CNC and MN is 1, CSC's and MS's is 2 and so forth.

Table 5. Proposed judgement and processing nodes for the GNP graph

Node Type	Node Function	Branch
Judgment Node	CNC, CSC, CEC CWC	3
	FP	4
Processing Node	MN, MS, ME MW, SH	1
Initial Boot Node	IBN	1

5.2 Summary

In this chapter, a methodology for evolving cooperation among predator agents in pursuit domain is presented. This methodology is based on genetic network programming and is actually the first time that GNP has been applied to evolve cooperation among agents in pursuit domain. Our methodology is concerned with analysis and design and of course the implementation of a GNP-based system in the pursuit domain which is capable of evolving teamwork and cooperation- technically called *emergent cooperation*.

There are four major phases required to design a GNP based methodology for a specific domain which are presented and described in this chapter. These involve defining the following:

1) Designing a suitable fitness function,

2) Assigning appropriate values to the running parameters,

3) Choosing a suitable termination condition

4) Designing appropriate judgment and processing nodes for the GNP agent architecture.

Chapter 6

Simulation Results and Discussion

In this chapter, we evaluate the proposed methodology and show that GNP can be used to evolve teamwork and cooperation strategy among multiple agents in a multi-agent system. Furthermore the results are compared with the GP approach, illustrating that the performance and learning speed of a GNP solution is more than the GP, while its computation cost is less. Also, we attempt to explain the superior outcome results for GNP. Finally, it is analytically shown why the GNP approach is significantly superior to GP in terms of computational cost and bloating.

6.1 Introduction

In this thesis several experiments are conducted on the pursuit domain and the main goals are as follows:

1. Gaining the knowledge of how to use GNP as a new methodology to evolve cooperation strategy among multiple agents.
2. Design and development of a GNP-based system for pursuit domain.
3. Showing how our proposed methodology is successful and effective.
4. Implementing the best GP tree evolved in Haynes experiments for the pursuit domain and comparing its performance with the best GNP graph, evolved in our experiments.
5. Examine the ability of proposed methodology in devising a conflict resolution mechanism for the predator agents.
6. Mathematically proving the superiority of GNP to GP in terms of computation costs.

6.2 Simulation Environment Settings

As mentioned in Table 1, the pursuit domain can be customized by a couple of parameters and the values for such parameters in our experiments are listed in Table 6.

Table 6. Environment parameters and their values

Parameters	Values
Grid size	finite
Grid shape	toroidal
Capture definition	prey surrounded
Legal moves	orthogonal only
Movements	agents move simultaneously
Sensors	the whole environment
Predator communication	disallowed
Prey movements	random
Predator/Prey number	1/4

6.2.1 Cycle and Episode

While working with pursuit domain, we encounter the concepts of cycle and episode. These concepts as used in Jelle Kok's pursuit package [49] are as follows: Time is divided into cycles. Each cycle consists of different stages in which either the prey or the predators are allowed to communicate with the server, and once a cycle is passed the predators will make just one move simultaneously after the prey moves. Note that a predefined number of consecutive cycles is called episode.

Table 7 displays all stages that the server uses. At the beginning of a cycle, visual messages are communicated to the prey after which the prey has *time_step ms* to send a communication message to the. Once this period has elapsed, the server communicates the *(send_action episode_nr cycle_nr)* message to the prey to indicate that the communication period is over and they now can send their movement command to the server. When *time_step ms* once again have elapsed the sending period is over and the prey on the field is updated according to the received movement commands. Now it is predators' turn, and the whole procedure repeats itself. The predators thus move simultaneously after the prey has moved.

Table 7. Description of the different stages of the server during one cycle

```
while not all preys are captured do
  {cycle}
  send visual information to all preys
  wait time_step ms to receive and process communication messages from prey
  send message (send_action episode_nr cycle_nr) to all prey
  wait time_step ms to receive movement commands prey
  update field
  check collisions
  send visual information to all predators
  wait time_step ms to receive communication messages from predators
  send message (send_action episode_nr cycle_nr) to all predators
  wait time_step ms to receive movement commands predators
  update field
  check collisions
  check whether a prey is captured
end while
```

6.2.2 The Pursuit Rules

Our pursuit games were designed to maximize the probability of conflicts, and reduce inadvertent resolution of conflicts (those that occur due to the movement of the prey). The rules of our pursuit games were as follows:

Goal: To capture the prey by occupying each of the four orthogonal positions around the prey and maintaining these positions throughout the rest of the simulation.

- A simulation lasts for 100 cycles
- A new simulation is run for each test case
- For each simulation cycle, each agent is allowed one move
- The moves can be any one of North, East, South, West or Here
- Here is used by an agent to forfeit its move and remain still
- All of the agents move concurrently so that there is no ordering of moves
- The randomly moving prey moves 90% of the time.
- No two agents can occupy the same cell
- In a case of conflict, agents involved will have their moves cancelled

- Collisions occur when:
  1. One or more agents try to move to a cell that is already occupied
  2. Two adjacent agents try to exchange cell positions
  3. Two or more agents try to move to the same cell

6.2.3 Architecture of GP Trees

In this section we introduce the function and terminal sets used in Haynes [50] GP system. This is illustrated in Table 8 and for further information we suggest to study their papers [50, 51].

Table 8. Function and terminal sets used by Haynes

Terminal	Type	Purpose	Function	Return	Arguments	Purpose/Return
B	Boolean	TRUE or FALSE.	CellOf	Cell	Agent A and Tack B	Get the cell coord of A in B.
Bi	Agent	The current predator.	IfThenElse	Type of B and C	Boolean A, Generic B and C	If A, then do B, else do C. (B and C must have the same type.)
Prey	Agent	The prey.	<	Boolean	Length A and Length B	If A < B, then TRUE else FALSE.
T	Tack	Random Tack in the range of Here to North to West.	MD	Length	Cell A and Cell B	Return the *Manhattan distance* between A and B.

6.3 Experiment 1: Implementing and Applying Haynes Best GP Tree

Haynes et al [50] designed and developed a GP based system to evolve cooperation strategy for homogenous predator agents in the pursuit domain. They were successful in defining a set of function and terminal nodes and evolving a GP tree for the predator agents to capture the prey. Their results were competitive with the hand-coded algorithms for the pursuit domain. After their research this GP system became one of the core approaches in the community which uses a machine learning technique to evolve a cooperation strategy in the pursuit domain. For this reason we used their best evolved tree as a benchmark for our GNP network. Figure 21 shows the best evolved tree in Haynes GP system.

```
IFTE( <( IFTE( T,
               MD( CellOf( Prey, H ),
                   CellOf( Bi,E )),
               MD( CellOf( Prey, N),
                   CellOf( Bi, H ))),
        MD( CellOf( Prey, N ), CellOf( Bi, W ))),
      IFTE( <( MD( CellOf( Bi, N),
                   CellOf( Prey, H )),
               MD( CellOf( Bi, N),
                   CellOf( Prey, N ))),
            N,
            E ),
      IFTE( <( MD( CellOf( Prey, N),
                   CellOf( Bi, N)),
               MD( CellOf( Bi, E),
                   CellOf( Prey, N))),
            W,
            S ))
```

Figure 21. The best GP program evolved by Haynes

The goals sought in this experiment were as follows:

1. To become familiar with the pursuit domain and get a hands-on experience with the simulation package developed by Jelle Kok [49].
2. To investigate how to design and implement homogenous agents.
3. To implement the conflict resolution mechanism as it was proposed by Haynes.
4. To see the practical results of a GP technique in coordinating agents in the pursuit domain.
5. To know why Haynes used the set of function and terminal nodes introduced in Table 8.
6. To use the results of this experiment as a benchmark to compare with the one in GNP technique.

This GP tree was tested in the pursuit domains with the 15*15 and 30*30 grid size. The results are shown in Table 9 and Table 10 respectively.

Table 9 . The fitness and number of captures for the best GP tree in 15*15 grid size

Run #	Fitness	Number of Captures (out of 100 test case)
1	4235	72
2	4427	92
3	4367	93
4	4409	90
5	4293	74
6	4376	81
7	4169	77
8	4333	85
9	4247	79
10	4366	84
Average	**4322,2**	**82,7**

Table 10. The fitness and number of captures for the best GP tree in 30*30 grid size

Run #	Fitness	Number of Captures (out of 100 test case)
1	8418	61
2	8187	49
3	8382	62
4	8445	69
5	8081	53
6	8657	71
7	8164	53
8	8559	74
9	8232	58
10	8251	59
Average	**8337,6**	**60,9**

As illustrated in the tables, the GP tree is executed 10 runs to increase the accuracy and in each run there are 1000 episodes (each episode includes 100 cycles). Next, the average value is calculated for the fitness and the number of captures.

When comparing the average results for both environments, we observed that the average number of captures in an episode decreased as the grid size increased.

6.4 Experiment 2: The Proposed GNP Methodology Applied on 15*15 Grid Size

In this experiment we used the GNP graph made from the judgment and processing nodes which were introduced and defined in section 5.3.1. The predator agents use the best evolved graph by our proposed GNP methodology to determine their moves. For training purposes, we used 30 training test cases. In all of cases, the prey is positioned in the middle of the environment. The size of the grid is 15*15. The value for the fitness of an individual is determined by getting the average of the fitness for those 30 test cases. The agents use the conflict resolution mechanism mentioned in chapter 4.3. All the predator agents execute the same code and are identical, hence homogeneous. Predator agents can sense the entire world.

6.4.1 The Evolved Graph

The genotype of the best evolved graph for the 15*15 environment is shown in Figure 22. In this graph all the delays are assumed zero. All of the four predator agents run this graph as their program. Evolution of this graph and predators ability to capture the prey by using this graph proves that our proposed methodology is successful and effective in this environment.

node 0	0 0	2
node 1	1 1	7 : 8 : 3
node 2	1 2	6 : 4 : 7
node 3	1 3	7 : 10 : 9
node 4	1 4	7 : 10 : 9
node 5	1 5	6 : 2 : 8 : 9
node 6	2 1	5
node 7	2 2	5
node 8	2 3	6
node 9	2 4	7
node 10	2 5	9

Figure 22. Genotype of the best evolved graph for the 15*15 environment

6.4.2 Results

Figures 23, 24 and 25 illustrate the fitness graph for the best, average and worst individuals during the 50 generations of the experiment.

As predicted the individuals in the first generation have low fitness since they are generated randomly. The desirable graphs start to appear from the 30th generation. Figure 23, has a step-shape graph since the elite preserve strategy is used. Figure 24, has a patterned curve illustrating the learning process. Figure 25, has a couple of spikes which show that genetic operators will not invariably improve performance and it is possible that they decrease the fitness.

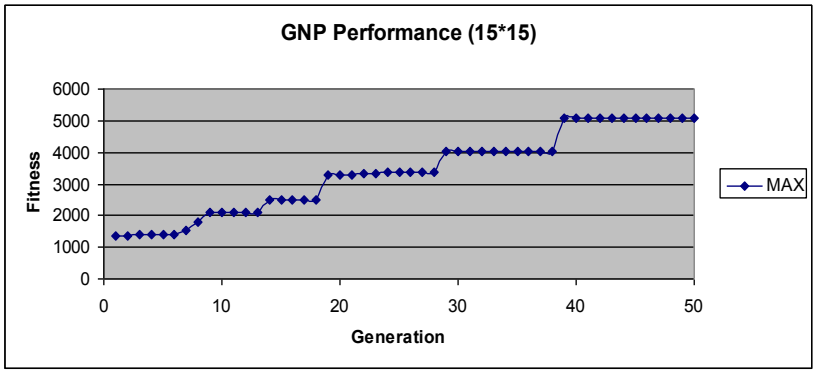

Figure 23. Fitness curve of the best individuals in the 15*15 environment

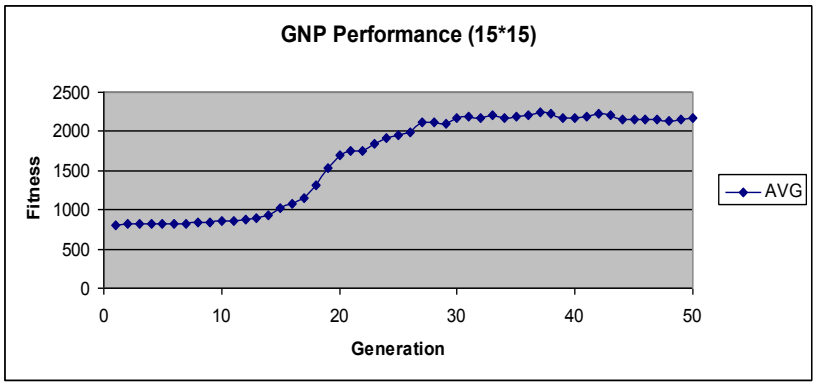

Figure 24. Average fitness curve in 15*15 environment

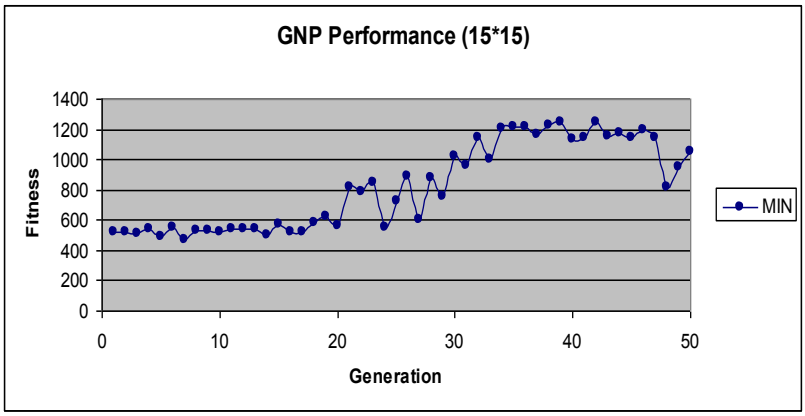

Figure 25. Fitness curve of the worst individual in 15*15 environment

6.4.3 Behavioural Analysis of the Result Graph

The execution of graph starts with the initial boot node (node 0) as the genotype of the evolved graph shows in Figure 22. It then goes to node 2, a judgment node, where it checks the south square of the predator. Destination node changes depend on the result. If there was a predator it goes to node 6, if it was a prey it goes to node 4 and if there was no agent there, then it goes to node 7. Afterwards, if we examine nodes 4, 6 and 7 and follow their connection gene, we reach node 5 which is a judgment node that judges the position of the prey. If the prey is to the north of the predator, the execution of the graph continues to node 6 which is correspondent to the MN processing node, thereby making predator move to the north in that cycle and if the prey is to the south of the predator, it continues to judgment node 2. Following the connection gene for node 2, we realize that this node eventually reaches node 7, which is a processing node corresponding to moving south. If the prey is to the east of the predator, it should go to processing node 8 which is correspondent to moving east and if prey is on the west side of the predator it continues to processing node 9 which is correspondent to moving west. If we follow the connection gene of nodes 6, 2, 8 and 9 we realize that they eventually end up to node 5 and the previous steps repeat.

Considering the above discussion, we can conclude that the strategy implicit in this graph for predators is “go towards the prey and never stop”. This result is notable since it is a somewhat greedy approach evolved by the GNP system.

6.4.4 Generality Test of the Results

As mentioned, in order to train the predator agents, 30 test cases were used in which the predators were randomly positioned in the domain at the beginning of an episode. Therefore this question remains, is the evolved graph dependent on the test cases or not? To answer this question we executed the best graph evolved in experiment 2 in 100 randomly generated environments and repeated this experiment 10 times. This test is called the generality test. The results are shown in Table 11.

Table 11. The results of the generality test for experiment 2

Run#	Fitness	Number of Captions (out of 100)
1	3844	75
2	3896	82
3	3922	80
4	3882	79
5	4116	91
6	3832	78
7	4009	87
8	3756	66
9	3854	68
10	3956	76
Average	**3906,7**	**78,2**

6.5 Experiment 3: The Proposed GNP Methodology Applied on 30*30 Grid Size

This experiment has the same settings as experiment 2, except the grid size is 30 by 30. All the predators are homogenous and execute the same code. The value for the fitness of an individual is determined by obtaining the average of the fitness over the 30 test cases. The agents use the conflict resolution mechanism mentioned in section 4.3.

6.5.1 The Evolved Graph

The genotype of the best evolved graph for the 30*30 environment is shown in Figure 26. In this graph all the delays are assumed to be zero. All of the four predator agents run this graph as their program. Evolution of this graph and the predators' ability to capture the prey by using this graph proves that our proposed methodology is successful and effective in the 30*30 environment.

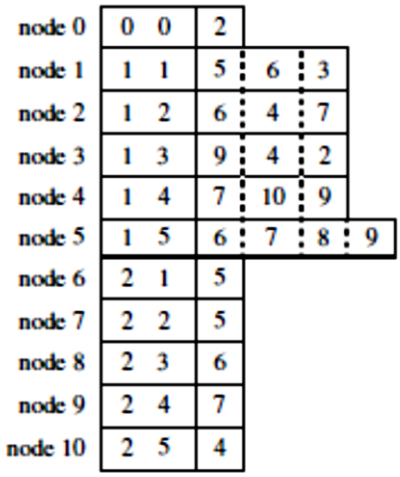

Figure 26. Genotype of the best evolved graph on experiment 3

6.5.2 Results

Figures 27, 28 and 29 illustrate the fitness graph for the best, average and worst individuals during the 50 generations of the experiment.

As expected the individuals in the first generation have low fitness since they are generated randomly. The desirable graphs appear from the 30th generation. Figure 27, has a step shape graph since the elite preserve strategy is used. Figure 28, has a patterned curve and it shows the learning process. Figure 29, has a couple of spikes which show that genetic operators will not invariably improve performance and it is possible that they decrease the fitness.

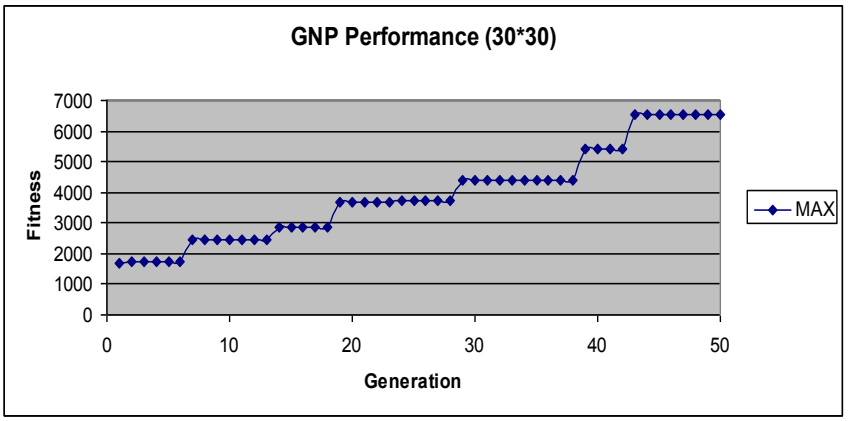

Figure 27. Fitness curve of the best individuals in the 30*30 environment

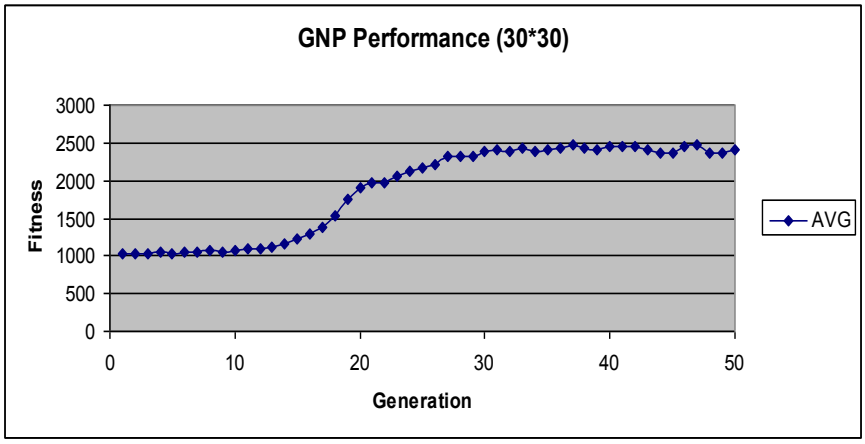

Figure 28. Average fitness curve in 30*30 environment

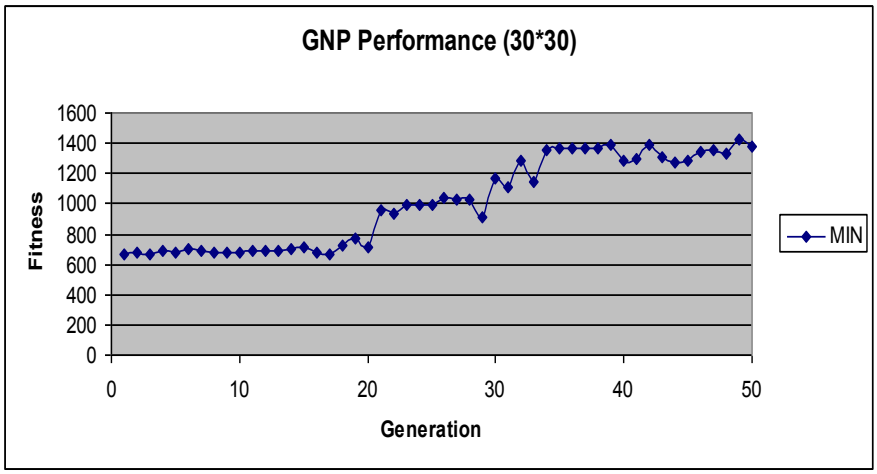

Figure 29. Fitness curve of the worst individual in 30*30 environment

6.5.3 Behavioural Analysis of the Result Graph

The execution of graph starts with the initial boot node (node 0) as the genotype of the evolved graph shows in Figure 26. It then goes to node 2 which is a judgment node and it checks the south square of the predator. Depending on the result, if there was a predator it goes to the node 6, if it was a prey it goes to the node 4 and if it there was no agent there it goes to the node 7.

Then if we examine nodes 4, 6 and 7 and follow their connection gene, we reach to the node 5 which is a judgment node that judges about the position of the prey. Now if the prey is in the north of the predator, the execution of the graph continues to node 6 which is correspondent to the MN processing node and makes the predator to move to the north in that cycle and if the prey is in the south of the predator, it continues to the judgment node 2 and with following the connection gene for node 2 we realize that this nodes eventually reaches node 7 which is a processing node corresponding moving to the south. If the prey is on the east side of the predator the graph goes to the processing node 8 which is correspondent to ME and if it is on the west side of the predator it goes to node 9 which is correspondent to MW.

Now if we examine the connection gene of the nodes 2, 6, 8 and 9 we realize that they end to the node 5 and the whole previous process repeats. Considering the above discussion it concludes that the strategy implicit in this graph for the predators is “go towards the prey and never stop”.

If we compare this cooperation strategy in the graph evolved for the 30*30 grid size with the results for the 15*15 environment we realize that, although the graphs are different, both represent the same strategy.

This is quite significant given that our proposed GNP methodology is independent of the grid size. In other words, we can use the evolved graph for any grid size to capture the prey in other environments.

6.5.4 Generality Test

As mentioned, in order to train the predator agents, 30 test cases were used in which the predators were positioned in the domain randomly at the beginning of an episode, so the question is that if the evolved graph is dependent to the test cases or not?. So we executed the best graph evolved in experiment 3 for 100 randomly generated environments and repeated this experiment 10 times. This test is called generality test. The results are shown in Table 12.

Table 12. The results of the generality test for experiment 3

Run#	Fitness	Number of captions (out of 100)
1	6356	42
2	6136	39
3	6235	36
4	6311	43
5	6553	49
6	6382	45
7	6105	38
8	6161	41
9	6357	41
10	6246	32
Average	6284,2	40,6

6.6 Experiment 4- Conflict Resolution Capability of the GNP Proposed Methodology

As mentioned earlier in our experiments, the conflict resolution mechanism is explicitly designed and hand-coded in the environment. In this section, the goal is to investigate if our proposed methodology has the ability to resolve conflicts implicitly.

In this experiment we used the same GNP methodology we used for experiment 2 and 3. The only difference is that we removed the hand-coded conflict resolution mechanism in the environment, whereby the agents may occupy the same square and they will not be

bumped back if they want to take the same square. This experiment is conducted on a 30*30 grid size.

As in the first trials of this experiment, the fitness for the graphs was low; we decided to continue the evolution process for 100 generations.

6.6.1 The Evolved Graph

The evolved graph resulted from experiment 4 is shown in Figure 30. This graph was used by all predators, yet it was inadequate in terms of illustrating a strategy. This will be described later in the analysis section.

Node 0	0 0	7			
Node 1	1 1	2	7	6	
Node 2	1 2	10	8	7	
Node 3	1 3	9	7	10	
Node 4	1 4	9	8	2	
Node 5	1 5	6	2	8	9
Node 6	2 1	5			
Node 7	2 2	1			
Node 8	2 3	7			
Node 9	2 4	10			
Node 10	2 5	5			

Figure 30. The genotype of the best evolved graph in experiment 4

6.6.2 The Results

Figures 31, 32 and 33 illustrate the curve for the best, average and worst individual in terms of fitness during the evolution process (100 generations) in a 30*30 grid size.

Similar expectations were observed in this experiment;-since the individuals of the first generation were generated randomly and had a very low fitness. However, as it can be noticed from the average fitness graph, they will improve generation by generation. Furthermore, indicating a learning process. Nonetheless after 100 generations, the fitness of the individuals was low and nearly all of them in the last generation had a similar genotype. This graph was not successful in capturing the prey.

We can conclude from these results that the GNP methodology proposed in this thesis, is not capable of devising a conflict resolution mechanism for homogenous agents. To solve this problem, introducing new judgment and processing nodes maybe favorable and will be the subject for our future research. But, to ensure the validity of our conclusion for all grid sizes we also conducted the generality test in section 5.7.3.

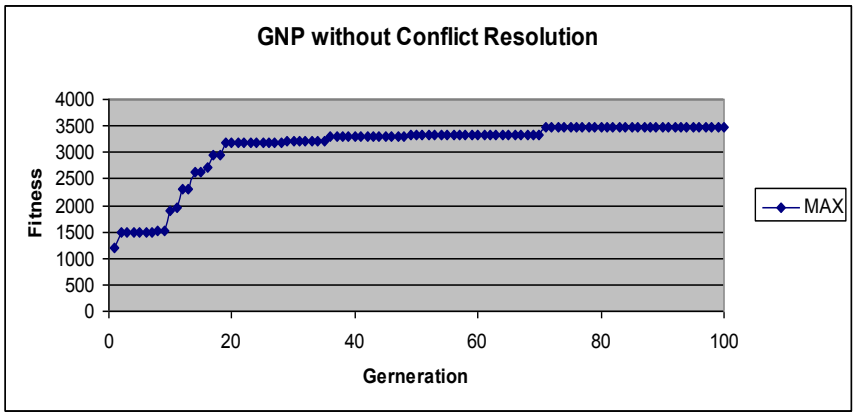

Figure 31. Fitness curve of the best individuals evolved in experiment 4

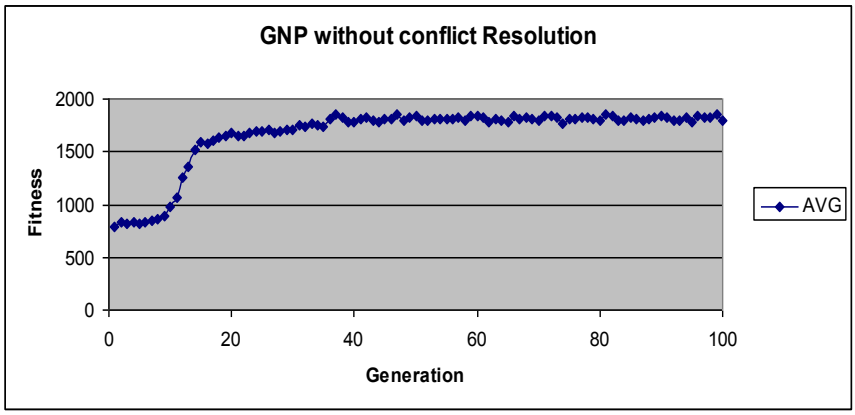

Figure 32. Average fitness curve of the individuals in experiment 4

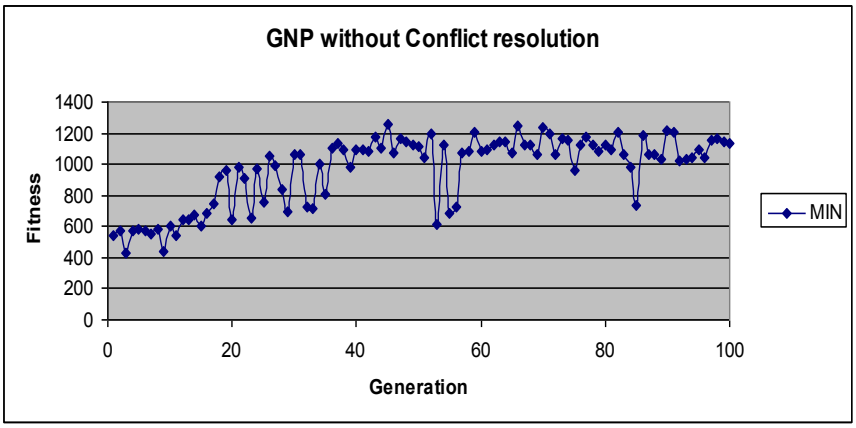

Figure 33. Fitness curve of the worst individuals in experiment 4

6.6.3 Generality Test

Table 13 shows the results for the generality test for experiment 4. As it can be seen, the graph was unsuccessful in capturing the prey except in run 3, in which there was only one capture which is also negligible. In short, we can conclude that the proposed methodology was unable to devise a conflict resolution mechanism.

Table 13. The results of generality test for experiment 4

Run	Fitness	Number of captures (out of 100)
1	2504	0
2	2455	0
3	2527	1
4	2480	0
5	2517	0
6	2446	0
7	2497	0
8	2437	0
9	2514	0
10	2452	0
Average	**2482,9**	**0.1**

6.7 Comparison between GNP and GP Systems

In this section we are going to compare the results of the two evolutionary approaches used in this thesis. The GP approach on 30*30 grid size which was presented in subsection 5.4 as the first experiment and the GNP approach on the same grid size as presented in subsection 5.6.

6.7.1 Capture Rate Comparison

Table 14, illustrates the results of the comparison based on the number of prey captures in 100 test cases. As it can be seen from the Table 14, the GNP technique is superior to the GP approach and it proves yet again that GNP is not only a more flexible way to evolve predator agents but it is also more effective.

Table 14. Comparison of capture rate between the GP and GNP

Algorithm	Average number of prey capture (out of 100)
GP	60,9
GNP	78,2

6.7.2 Computational Costs Comparison

For training purposes, we used 30 randomly generated test cases, with the prey in the center and the predators in random positions. For each test case, we deployed randomly moving prey. The computational costs of evaluation made it infeasible to run every test case for each individual of each generation. There were four agents whose codes needed to be evaluated 50 times for each test case for each individual in every single generation. For just one generation the computational costs are 4 * 100 * 30 * M, where M is the population size. And the number of code evaluation for the entire G generations is 4 * 100 * 30 * M * G.

In GNP experiments M equals 200 and G equals 50, whereas in GP experiments M equals 3000 and G equals 500. This means the computational cost of GNP solution is 150 times less than GP solution. Because in EC most of the CPU time is used for fitness evaluation, our method is also significantly superior in saving CPU time.

We could have reduced the computational cost by reducing the population size. However, this parameter is known to be one of the most important for the successful application of GP. We therefore used a sampling technique, whereby for each generation we selected 10 test cases at random (without reselection) from the 30 test cases and used them to evaluate each individual. This allows us to fairly compare the fitness of individuals of the same generation whilst minimizing the cost of evaluation.

The cost of evaluating a generation is now reduced to 1/3th. However, a drawback with this technique is that we need to run the evolutionary process for a longer number of generations to allow the opportunity for each test case to be evaluated many times. Hence we doubled the number of generations that we would normally use, thereby bringing about a reduction in computation.

6.7.3 Bloat Comparison

In this subsection we are going to mathematically prove the superiority of GNP results in our experiments over GP in terms of bloating. *Bloating* presents a serious problem in scaling GP to real lager and more difficult problems particularly in dynamic environments such as pursuit domain. There is no bloating in GNP, but in genetic programming, after some generations (typically below one hundred), the search for better programs halts as the programs (trees) become too large and searching space becomes enormous.

- Bloated trees occupy a large amount of memory.
- Bloated trees are computationally intensive and slow down the system.
- Bloated trees are often more difficult to modify in meaningful ways.

In GP, it has been justified [73]: "Sub-tree crossover would tend to cause programs to increase in size on average at cost less than $O(t^2)$, but it will approach a square power law, $O(t^2)$, as the programs get bigger", where t is the passed time. Now let us consider a generation to be equivalent to a time step. We can say $O(t^2)$ is equivalent to $O(g^2)$. (in terms of big-Oh). Bloat can *only* occur when variation operators exist in the system (crossover, mutation) and the more variations that are introduced into the system the more bloating there will be. We can say that bloat approaches $O(i^2)$. (i is the number of

new individuals created per generation). The number of individuals created is proportional to the number of individuals in the population (n). So we can say $O(i^2)$ is equivalent to $O(n^2)$. Thus in GP, for a population size of n we have the following equality:

$$Bloat\ (n) = O(t^2) = O(g^2) = O(i^2) = O(n^2)$$

On the other hand, in GNP, since the programs are graphs and the variation operators simply change the way nodes connect to each other, there will not be any bloat. Therefore, in GNP, for a population size of n we have no bloating.

6.8 Summary

In the previous chapter we proposed a methodology based on GNP to evolve teamwork and cooperation among predator agents in the pursuit domain. In this chapter, the results of running such a system in pursuit domain were presented and compared to those of GP.

The results show that our proposed approach is successful and its capture rate is more than GP while its computational cost is less. Furthermore it shows a faster learning speed than GP. We also provided some analytical results of the proposed approach in terms of bloating.

In fact, the results of four different experiments were presented in this chapter. In the first experiment we implemented the best GP tree reported by Haynes [50], applied it to our platform and reported its performance. In the second experiment, we ran the proposed GNP system on the pursuit domain with a 15*15 grid size and reported the results. For the third experiment, we ran it on a 30*30 grid size to investigate if the grid size is important in evolving different strategies. However, the results showed the grid size does not affect the evolved strategy. Finally, in the fourth experiment we tried to investigate if our proposed methodology is successful in evolving cooperation strategies that has the capability to implicitly resolve conflicts since in all the first three experiments the conflict resolution mechanism is *explicitly* designed in the environment, The results indicated that our system was not capable of evolving an implicit conflict resolution strategy automatically but some suggestions were made and verifying them remains to be seen in our future research topics.

Chapter 7
Summary and Conclusions

This chapter summarizes the main contributions of this work, highlights our conclusions, and proposes some future research directions stemming from our work.

7.1 Review of the work

The identification, design and implementation of strategies to devise teamwork and cooperation are central research issues in the field of Multi-agent Systems (MAS). It is nearly impossible to identify or even prove the existence of the best cooperation strategy. In most cases a cooperation strategy is chosen if it is reasonably good.

In this thesis, we designed and developed a methodology using a relatively new evolutionary computation technique called Genetic Network Programming (GNP) to automatically evolve cooperation, technically called emergent cooperation, in the pursuit domain and finally the results were compared to those of GP.

The simulation results showed that our proposed methodology is capable of evolving cooperation among agents in the pursuit domain. Its performance is significantly superior to the GP solution. Moreover, its computation cost is less and the learning speed is faster. Finally, it is mathematically proven that why GNP is significantly superior to GP in terms of computational cost and bloating.

In Chapter 2 cooperative multi-agent learning and reviews on previous studies of how machine learning techniques are used to devise cooperation in multi-agent systems were introduced. It specifically surveyed the literature related to applying evolutionary techniques in the pursuit domain. Chapter 3 provided a general overview of the evolutionary algorithms and described and compared genetic programming and genetic network programming. In Chapter 4 the pursuit problem and its environment parameters were described in detail as well as the pursuit package which is used for the experimental simulations in this thesis.

In Chapter 5, we implemented the best GP tree evolved by Haynes, and evaluated its performance in the pursuit domain running under our platform for gird sizes of 15*15 and 30*30. We then demonstrated the design of our proposed GNP-based methodology for pursuit domain and simulation results were shown and critically evaluated on 15*15 and 30*30 grid sizes. Moreover, we compared the performance of GNP with GP. Finally, the ability of the proposed GNP methodology in devising conflict resolution mechanism was tested and shown to be incapable of evolving a conflict resolution mechanism.

It is worth mentioning that the experiments in this thesis were conducted with the help of a software package for pursuit domain called *Pursuit_Package_0.9* [49] and the platform was *SUSE 10.0* as the operating system and a PC with *Intel Pentium M*, 1.73 GHz with 512 MB of RAM as the hardware. On average, it took approximately 100 hours to carry out each experiment.

7.2 Directions for Future Researches

The experiments in this thesis provide a foundation and context for more extensive research. The following lists further research directions stemming from our work:

- Evolving predator agents which have limited visibility range.
- Evolving heterogeneous predators using GNP.
- Modifying crossover and mutation rate and studying the effects.
- Changing the number of judgment and processing nodes to more than just one for each.
- Changing the algorithm controlling the prey agent movements and study its effect on the evolution process.
- Improving the methodology in order resolve conflicts among agents.
- Changing the fitness function and studying its effect on learning speed.
- Investigating evolving agents via GNP for real world/large scale MAS applications

References

[1] Russell, S. J., and Norvig,P., *Artificial Intelligence: a Modern Approach*. Prentice Hall, 2nd edition. 2003.

[2] Vlassis, N., *A Concise Introduction to Multiagent Systems and Distributed AI* Informatics Institute, University of Amesterdam, 2003.

[3] Aly Gomaa W., *A new learning technique for planning in cooperative multiagent systems*, Msc thesis, Alexandria University, 2002.

[4] Holland John H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992. First Published by University of Michigan Press 1975.

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

[6] Hirasawa K., Okubo M., Hu J., and Murata J., "Comparison between genetic network programming (GNP) and genetic programming (GP)", in *Proc. IEEE Congr. Evolutionary Computation*, pp. 1276–1282, 2001.

[7] Katagiri H., Hirasawa K., and Hu J., "Genetic network programming-application to intelligent agents-*"*, in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, pp. 3829–3834, 2000.

[8] Katagiri H., Hirasawa K., Hu J., and Murata J., "A New Model to Realize Variable Size Genetic Network Programming - A Case Study with the Tileworld Problem", GECCO Late Breaking Papers 2002.

[9] Murata T., Nakamura T., and Nagamine S., "Performance of genetic network programming for learning agents on perceptual aliasing problem", *IEEE International Conference on Systems, Man and Cybernetics* (IEEE SMC05), Hawaii (USA) , 2005.

[10] Murata T., and Nakamura T., "Genetic network programming with automatically defined groups for assigning proper roles to multiple agents", Proceedings of the 2005 conference on Genetic and evolutionary computation, volume 2, pages 1705-1712, Washington DC, USA, 2005.

[11] Murata T., and Nakamura T., "Developing Cooperation of Multiple Agents Using Genetic Network Programming with Automatically Defined Groups", *Proc. of Late Breaking Papers in GECCO*, 2004.

[12] Panait L., Luke S., "Cooperative Multi-Agent Learning: The State of the Art", *Autonomous Agents and Multi-Agent Systems*, Springer-Verlag, Volume 11, pp. 387-434, 2005.

[13] Stone P., and Veloso M., "Multiagent systems: A survey from a machine learning perspective", In *Autonomous Robotics,* volume 8, number 3, 2000.

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

[15] Bongard J. C., "The legion system: A novel approach to evolving heterogeneity for collective problem solving", *Genetic Programming: Proceedings of EuroGP-2000*, 2000.

[16] Potter M., Meeden L., and Schultz A., "Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists". In *Proceedings of The Seventeenth International Conference on Artificial Intelligence (IJCAI-2001)*, 2001.

[17] Iba H., "Emergent cooperation for multiple agents using genetic programming", In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature IV: Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 32–41, Berlin, Germany, 1996.

[18] Bull L., and Fogarty T. C., "Evolving cooperative communicating classifier systems". In A. V. Sebald and L. J. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming* pages 308–315, 1994.

[19] Miconi T., "When evolving populations is better than coevolving individuals: The blind mice problem". In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

[20] Jansen T., and Wiegand R. P., "Exploring the explorative advantage of the cooperative co-evolutionary (1+1) EA", In E. Cantu-Paz *et al*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. Springer-Verlag, 2003.

[21] Chang Y. H., Ho T., and Kaelbling L., "All learning is local: Multi-agent learning in global reward games", In *Proceedings of Neural Information Processing Systems (NIPS-03)*, 2003.

[22] Tumer K., Agogino A. K., and Wolpert D. H., "A bartering approach to improve multiagent learning", In *Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 386–393, 2002.

[23] Stone P., *Layered Learning in Multi-Agent Systems*. PhD thesis, Carnegie Mellon University, 1998.

[24] Durfee E., Lesser V. and Corkill D., "Coherent cooperation among communicating problem solvers", *IEEE Transactions on Computers*, C-36(11):1275–1291, 1987.

[25] Dah T., Mataric M., and Sukhatme G., "Adaptive spatio-temporal organization in groups of robots", In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02)*, 2002.

[26] Kitano H., Asada M., Kuniyoshi Y., Noda I., and Osawa E., "RoboCup: The robot world cup initiative", In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 5–8, 1997.

[27] Balch T., "Reward and diversity in multirobot foraging", In *IJCAI-99 Workshop on Agents Learning About, From and With other Agents*, 1999.

[28] Parker L., "Multi-robot learning in a cooperative observation task", In *Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000)*, 2000.

[29] Schultz A., Grefenstette J., and Adams W., "Robo-shepherd: Learning complex robotic behaviors", In *Robotics and Manufacturing: Recent Trends in Research and Applications, Volume 6*, pages 763–768. ASME Press, 1996.

[30] Werner G. M. and Dyer. M., "Evolution of herding behavior in artificial animals", In *From Animals to Animates 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, 1993.
[31] N. Glance, and Huberman B., The dynamics of social dilemmas. *Scientific American*, 270(3):76–81, 1994.
[32] Lichbach M. I., *The cooperator's dilemma*. University of Michigan Press, 1996.
[33] Lesser V., Corkill D. and Durfee E., *An update on the distributed vehicle monitoring testbed*, Technical Report UM-CS-1987-111, University of Massachessets Amherst, 1987.
[34] Steeb R., Cammarata S., Hayes-Roth F., Thorndyke P., and Wesson R., "Distributed intelligence for air fleet control", In A. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 90–101.Morgan Kaufmann Publishers, 1988.
[35] Chang Y., Ho T., and Kaelbling L., *Multi-agent learning in mobilized ad-hoc networks*. In Proceedings of Artificial Multiagent Learning. Papers from the 2004 AAAI Fall Symposium. Technical Report FS-04-02, 2004.
[36] Fogel D. B., *Evolutionary Computation,* The Fossil record: Selected Readings of the History of Evolutionary Algorithms, IEEE press, 1998.
[37] Available through www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php.
[38] Qureshi M. A., *The evolution of agents*, PhD thesis, University of London, 2001
[39] Koza John R., "Evolution of emergent cooperative behavior using genetic programming", In Ray Paton, editor, *Computing with Biological Metaphors*, pages 280–297. London: Chapman and Hall, 1994.
[40] Koza John R., Andre D., Bennett III Forrest H., and Keane M., *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman, 1999.
[41] Eguchi T., Hirasawa K., Hu J. and Ota N., "A study of evolutionary multi-agent models based on symbiosis", *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, 36(1):179-193, 2006.
[42] Benda M., Jagannathan V., and Dodhiawala R., *On optimal cooperation of knowledge sources - an empirical investigation*, Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, 1986.
[43] Korf Richard E., "A simple solution to pursuit games", In *Working Papers of the11th International Workshop on Distributed Artificial Intelligence*, pages 183–94, 1992.
[44] Available through http://staff.science.uva.nl/~jellekok/software/index_en.html.
[45] Hayens T., Wainwright R., Sen S., and Schoenefeld D., "Strongly typed genetic programming in evolving cooperation strategies" , *Proceedings of Sixth International Conference on Genetic Algorithms,* pages 271-278, San Francisco CA, 1995.
[46] Balch T., *Behavioral Diversity in Learning Robot Teams*. PhD thesis, College of Computing, Georgia Institute of Technology, 1998.
[47] Bongard J. C., The legion system: A novel approach to evolving heterogeneity for collective problem solving. Genetic Programming: Proceedings of EuroGP-2000, volume 1802, pages 16–28, Edinburgh, 15-16 2000.

[48] Potter M., Meeden L., and Schultz A., Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI), 2001.
[49] Haynes T., and Sen S., Evolving behavioral strategies in predators and prey. In G. Weiß and S. Sen, editors, Adaptation and Learning in Multiagent Systems, Lecture Notes in Artificial Intelligence. Springer Verlag, Germany, 1995.
[50] Haynes T., Sen S., Schoenefeld D., and Wainwright R., Evolving a team. Working Notes for the AAAI Symposium on Genetic Programming, pages 23–30, MIT, Cambridge, USA, 1995.
[51] Haynes T., Sen. S. D. Schoenefeld, and Wainwright R., Evolving multiagent coordination strategies with genetic programming. Technical Report UTULSA-MCS-95-04, The University of Tulsa, May 31, 1995.
[52] Quinn M., Smith L., Mayley G, and Husbands, P., Evolving formation movement for a homogeneous multi-robot system: Teamwork and role-allocation with real robots. University of Sussex, Brighton, 2002.
[53] Good B. G., Evolving multi-agent systems: Comparing existing approaches and suggesting new directions. Master's thesis, University of Sussex, 2000.
[54] Andre D., and Teller A., Evolving team Darwin United. In M. Asada and H. Kitano, editors, RoboCup-98: Robot Soccer World Cup II. Springer Verlag, 1999.
[55] Luke S., Genetic programming produced competitive soccer softbot teams for RoboCup97. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 214–222. Morgan Kaufmann, 1998.
[56] Luke S., Hohn C., Farris J., Jackson G., and Hendler J., Co-evolving soccer softbot team coordination with genetic programming. In Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence, Nagoya, Japan, 2007.
[57] Bull L., and Fogarty T. C., Evolving cooperative communicating classifier systems. Proceedings of the Fourth Annual Conference on Evolutionary Programming, pages 308–315, 2004.
[58] Miconi T., A collective genetic algorithm. In E. Cantu-Paz et al, editor, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pages 876–883, 2001.
[59] Jansen T., and Wiegand R. P., Exploring the explorative advantage of the cooperative co-evolutionary (1+1) EA. In E. Cantu-Paz et al, editor, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO). Springer-Verlag, 2003.
[60] Cao Y., Fukunaga A., and Kahng A. Cooperative Mobile Robotics: Antecedents and Directions. Autonomous Robots 4, pages 7-27, 2007.
[61] Ijspeert A., Martinoli A., Billard A., and Gambardella L., Cooperation through the Exploration of Local Interactions in Autonomous Collective Robotics: the Stick Pulling Experiment,: Swiss Federal Institute of Technology, 2000.
[62] Mataric M., Designing emergent behaviors: From local interactions to collective intelligence. From Animals to Animates 2: Second International Conference on the Simulation of Adaptive Behavior pages 432-441, 1993.

[63] Bongard J., and Pfeifer R., Evolving Complete Agents Using Artificial Ontogeny. In F. Hara, & R. Pfeifer, Morpho-functional Machines: The New Species Designing Embodied Intelligence. 2003.
[64] Axelrod R., Evolution Strategies in the iterated prisoner's dilemma. In L. Davis Genetic Algorithms and Simulated Annealing, Morgan Kaufmann, 2000.
[65] Dugatkin L., N-Person games and the evolution of cooperation: A model based on predator inspection in fish. Journal of Theoretical Biology, 1990.
[66] Miller G., and Cliff D., Co-Evolution of Pursuit and Evasion I: Biological and Game-Theoretic Foundations, England: School of Cognitive and Computing Sciences, University of Sussex, 1994.
[67] Yong C., and Miikkulaine R., Cooperative Co-evolution of Multi-Agent Systems Austin, Texas, University of Texas, 2002.
[68] Montana D., Strongly typed genetic programming. Evolutionary Computation, 1995.
[69] Moriarty D., and Miikkulainen R., Forming Neural Networks through Adaptive Co-evolution Evolutionary Computation, pages 373-399, 1998.
[70] Denzinger J., and Fuchs M., Experiments in Learning Prototypical Situations for Variants on the Pursuit Game. In Proceedings of the Second International Conference on Multi-Agent Systems pages.48-55, 2006.
[71] Nishimura S., and Takashi I., Emergence of Collective Strategies in a Predator Prey Game Model. Artificial Life, pages 243-260, 2007.
[72] Larry Stephens M., The effect of agent control strategy on the performance of a DAI pursuit problem. In Proceedings of Distributed AI Workshop, 1999.
[73] Langdon W., and Poli R., *Foundations of Genetic Programming,* Springer, 2002.