Multi-Agent Communication and Collaboration

Bradley Justin Van Aardt

A Research Report submitted to the Faculty of Engineering and Built Environment, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg, January 2005

D E C L A R A T I O N

I declare that this is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg, South Africa. It has not been submitted before for any degree or examination at any other university.

Signed this _____ day of _____ 2005

_____

Bradley Justin Van Aardt

A B S T R A C T

Multi-Agent Systems are becoming a popular paradigm for many engineering applications. However, there is still much research to be performed in this fast growing field. In this thesis, the effect of learning in multi-agent systems on communication and collaboration between agents is investigated. This research focuses on agents learning local cooperative behaviour from a centralised agent, as well as using learning to reduce the amount of communication between agents that use negotiation to achieve their goals. A simple test problem is formulated in MATLAB. The effect of learning is clearly seen to reduce the amount of communication between agents by up to 50%, while still maintaining cooperative behaviour. The agents are also demonstrated to learn to a large degree cooperative local behaviour from a central system.

ACKNOWLEDGMENTS

I wish to thank the following people who helped throughout this research project:

Professor Tshilidzi Marwala, who is an exceptionally inspiring and motivating supervisor. He embodies all the potential and talent of this country.

My family, for supporting me throughout yet more studies and always believing in me.

Lukasz Machowski, who is always ready to discuss anything interesting and invariably provides excellent suggestions.

The Council for Scientific and Industrial Research (CSIR) Defencetek, for supporting and funding this research.

All of the A-Lab at the School of Electrical and Information Engineering at the University of the Witwatersrand for a year well-spend in their company.

F O R E W O R D

This thesis is compiled as part of the requirements for the degree of Master of Science in Engineering, at the University of the Witwatersrand. Chapter 1 introduces the topic, and explains the research performed. Chapters 2-4 provide details on the study and the results found. Chapter 5 summarises the results of the preceding chapters and concludes the thesis.

Chapters 2-4 of this thesis are written as stand-alone papers, and can be treated individually. However, the order of the chapters follows the logical progression of the work, where Chapters 2 and 3 present the earlier part of the research, and chapter 4 presents the culmination of the research with some additions and modifications to the preceding 2 chapters. The research presented in Chapter 4 was thus performed with the benefit of hindsight and comments offered on the first two published papers.

There are 2 appendices to this thesis. Appendix A presents extra data gathered from the simulations. Appendix B contains details of the source code used to conduct the study.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The purpose of this research is to study the influence learning can have in multi-agent communities. Specifically, we examine the influence on communication and collaboration in a cooperative agent system. Constructing Multi-agent systems is not a trivial task, and enabling cooperation and collaboration among agents in the system is an ongoing research topic. Typically, the task is to construct agent behaviours such that each of the agents' local actions lead to some desired overall result of the system. Normally, agents are enabled to negotiate with each other, or a selected central agent is used to coordinate the actions of all the agents, to ensure that the agents do not compromise the global result of the system.

The problem in these agent systems is that negotiating between agents can consume significant bandwidth, as is the case for the centralised controller, where each agent must communicate its local data to the central agent as well as the central agent communicating the action plan to each of the agents.

However, often the agents will face the same, or very, similar problems over the course of operation. This implies that solutions are continually being reformulated, with the consequent cost in communication. The hypothesis in this thesis in that once a problem has been seen by the agents, and the solution found, machine learning can be used to associate the problem characteristics with the solution. In this research the effect of learning (a) to enable the agents to learn appropriate cooperative behaviour and (b) on the level of communication between agents is studied.

A simple board game is used as the study test problem. The players in the game are a community of agents which must cooperate to achieve the end goal of the game. Two variations in the structure of the system are used. In the first, a central agent is used to coordinate all the agents in the system. In the second, the agents negotiate with each other to achieve their goals. In each case the agents are equipped with Multi-layer Perceptron (MLP) neural networks as the machine learning realisation. Neural networks are used due to their pattern recognition abilities, as well as the ability to generalise responses from seen problems to unseen problems.

The second chapter of this thesis examines the case of a central controller in the system, where each agent learns from this controller. Chapter 3 details a system in which the agents negotiate for a solution. The fourth chapter is a report on both types of systems, with further results and analysis. It encompasses the principles of the research.

The following papers have been published\submitted for publication from this research:

1. B. Van Aardt, T. Marwala, "A study in a Hybrid Centralised-Swarm agent Community", Proceedings of the 3rd IEEE International Conference on Computational Cybernetics (ICCC), Mauritius, 2005, pp 169 - 174.
2. B. Van Aardt, T. Marwala, "Reducing Inter-Agent communication due to Negotiation through Learning", Proceedings of the Fifteenth Annual Symposium of the Pattern Recognition Association of South Africa (PRASA), 2004, pp 149 – 155.
3. B. Van Aardt, T. Marwala, "Using Learning in Multi-Agent Communities to Improve Communication Efficiency", submitted for publication in the Journal of Systemics, Cybernetics and Informatics.

C h a p t e r 2

A STUDY IN A HYBRID CENTRALISED-SWARM AGENT COMMUNITY

Abstract *– This paper describes a systems architecture for a hybrid Centralised/Swarm based multi-agent system. The issue of local goal assignment for agents is investigated through the use of a global agent which teaches the agents responses to given situations. We implement a test problem in the form of a Pursuit game, where the Multi-Agent system is a set of captor agents. The agents learn solutions to certain board positions from the global agent if they are unable to find a solution themselves. The captor agents learn through the use of MLP neural networks. The global agent is able to solve board positions through the use of a Genetic Algorithm. The cooperation between agents and the results of the simulation are discussed here.*

Introduction

The concept of Multi-agent systems is an engineering paradigm that has been gaining momentum over the past years [1]. A particular form of multi-agent system, swarm based systems, have been successfully applied to a number of problems [2].

A large part of the difficulty in designing such systems is assigning local strategies to individual agents in a community, to obtain some desired resultant overall or global behaviour [3]. In fact, the subject of assigning local strategies is of current research, and practical, interest. A methodology whereby agents are equipped with a host of strategies at design time, and then learn through interaction the appropriate local strategy to use in a given situation has been proposed in [4].

The ideal situation would be where agents are able to dynamically develop their own local behaviours, with the outcome of the given user desired global behaviour, in response to the environment in which they operate. A possibility would be for agents to identify certain situations using pattern recognition techniques, and then act upon these situations using learned responses.

We present a study in this paper whereby agents learn responses to situations, or local strategy, from an entity with a global view of the problem. In this way we theorise that the agents will learn local strategies that are likely to lead to the overall globally desired behaviour of the system.

Firstly, we give a short background to the domain of multi-agent systems, and of the machine learning and evolutionary technology we use in the study. We then present the details of the study, and a brief analysis of our findings.

Background

Multi Agent Systems

The multi-agent paradigm, in which many agents operate in an environment, has become a useful tool in solving large scale problems through a "divide and conquer" strategy [5]. The Multi-agent system is a distributed, decentralised system. The paradigm of individual entities collaborating to solve a particular problem that is beyond each entity's own capabilities is a natural concept, and one that is proving to be very powerful in practice [6]. However, while the concept is easily understandable, the implementation is not trivial. There are many complexities and subtleties in these such as [5]:

- Decomposing and allocating problems to the agents
- Describing the problem to the agents
- Enabling communication and interaction among the agents
- Decentralised systems, in the context of Multi-Agent systems, promise the following advantages [5][7] [8][9][10]
- No single failure point, therefore greater robustness. Multi-agent systems have the capacity to degrade gracefully.
- Possibility of faster response times and fewer delays as the logic/intelligence is situated nearer to the problem domain.
- Increased flexibility to take account of changes occurring in the problem domain.
- Modularity and Scalability. Multi-agent systems can be increased in size dynamically according to the demands of the problem.

The problems associated with Multi-agent systems are [1] [2][5] [10]:

- Difficulty in measuring and evaluating the stability and security of the system.
- Excessive communication between agents can slow down the system. This is often countered by heavily restricting the amount of communication between agents.
- Possibility of getting stuck in non-optimal solutions of the problem, often due to the lack of global knowledge of the problem from each agent's point of view.

- Most Multi-agent systems are built in an ad-hoc way since there is no absolute theory for these types of systems. There have been recent attempts however, to formalise the design of agent based systems, such as the Gaia Methodology [7]. However these are not yet in widespread use.

Swarm Based systems

Swarm intelligence is a particular paradigm for multi-agent systems which emphasizes distributedness and agent simplicity. It is based on the observations of social insects in nature, such as ants, termites and bees [2]. Such insect societies are extremely organized, even though there is no central control or planning. Each agent in the system is programmed only to achieve its own Local Goal. The agent's behaviours in Swarms are very simple: the intelligence of the system is 'emergent' from the overall behaviours of all the agents in the system. The communication between agents is usually performed indirectly [9], by agents making changes to the environment, which other agents act upon. This is analogous to insects laying pheromone trails to food source, etc. Emphasis is therefore placed on reactivity in these systems. Swarm systems have been successfully applied to many problems, notably routing in computer and telecoms networks [2] and recently to a manufacturing control system [9].

The disadvantage of swarm based systems is that no agents actually have a global view of the problem to be solved. All agents are entirely focused on achieving their own Local Goals, whether or not these goals are to the benefit or detriment to the overall community. This can exacerbate the problem of the system getting stuck in local optima, or worse, cause the system to fail.

The advantage of a centralised system, where there is effectively one agent or processing unit to control the whole system, is that such a system will make decisions which will benefit the global goal of the system.

Neural Networks

Artificial neural networks are inspired by the functioning of biological neurons in animal and human brains. Artificial neural networks differ from most other computing techniques in that they are able to learn arbitrary relations between sets of data presented to them, and generalise these relations to unseen data. That is, rules are not explicitly programmed or set, but are learned from experience by the network [10]. The basic architecture of a neural network is also based on that of their biological counterpart: both networks consist of many simple elements, known as neurons, operating in parallel [11] [12]. The most widely used neural network models are the Multi-layer Perceptron (MLP) and Radial Basis Function (RBF) networks.

We use neural networks in the captor agents because of their ability to learn from example. The examples in this case will be the solutions found by the global agent.

Genetic Algorithms

Genetic algorithms (GAs) are inspired by the process of natural selection and evolution in nature. GAs are a class of non-greedy algorithms – they perform an exploration of the search space. GAs were first proposed by John Holland in 1975 [13] and can be thought of as four modules: Encoding, Selection, Recombination and Evaluation. Of the modules, only the evaluation Function is problem specific. With a GA, possible solutions to a problem are encoded as a chromosome string. A population of chromosomes make up a generation of possible solutions to the problem. From each generation, the fitness of each chromosome is evaluated by the problem-specific evaluation module. Certain chromosomes are selected by the selection module, by some procedure, and those selected are then mated by the recombination module to form the next generation of chromosomes.

Hypothesis

Reference [14] states that the method of human learning when presented by a new task is to use rational reasoning to perform the decision making process behind solving the task. After using this deliberative approach, and we begin to "master" a task, we are able to perform it "naturally", without explicitly performing rational reasoning. At this point, people use their pattern recognition skills to perform the task.

It is hypothesised that the human model of discovery and learning can be applied to agent strategising. Evolutionary optimisation and genetic programming techniques can be used to discover strategies for the agent community. These methods are desirable due to the large number of possible solutions to the problem. This will be the deliberative stage, where a problem solution is formulated. Once a strategy has been discovered and used, a neural network can be trained with the results. This is the pattern recognition stage. Thus over time, the neural networks should learn a large number of strategies, while generalising these strategies to other scenarios. Thus, as agents perform more tasks, the problem solving should become "natural" to them, as the neural networks start taking over from the genetic optimisation module.

Method

Our proposal is to link the qualities of the swarm based (or highly decentralised system) with a centralised system. To do this, we propose a system that consists of both a centralised control, and a swarm of agents. Our method aims to use the centralised control

in the deliberative phase of problem solving, and then switch to a distributed system once the problem solutions, as discovered by the centralised system, have been learned by the "subconscious" of the agents, as represented by a neural network.

Test Problem

The proposed system is applied to the game of Pursuit. A Pursuit board is represented by a 2 dimensional grid pattern, as shown in Figure 1. The crosses represent the Captor agents, while the circle represents the Fugitive agent (top row). The aim of the game is for the Captors to surround or corner the Fugitive. Each agent may only move one block per turn. Legal moves on a Pursuit board are Up, Down, Left, Right and Stay. No diagonal moves are allowed. At least one captor agent must move during the captors' move. Furthermore, the board does not wrap around. A game is ended when the fugitive is captured, implying that the fugitive has no place to move to.

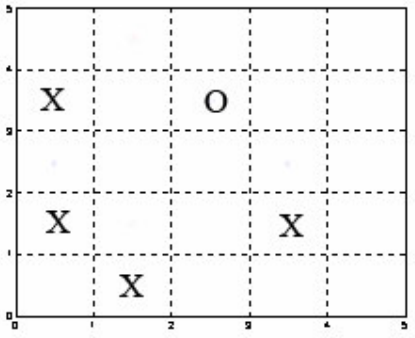


Figure 1 Pursuit Board representation

System Description

The architecture of the system is to have four separate agents representing the captors. Each of the agents has a neural network, which are trained independently of each other. Each agent is responsible for making a valid move.

The system also has a genetic algorithm based agent, which can control all of the Captor agents, called the Global Agent. It has a global view of the system. When the Global Agent is invoked, it acts as a centralised system, which proposes moves that best satisfy the global goal of the system.

The system operation is shown in flowchart form in Figure 2. At each turn for the captors to move, the Captor agents are invoked. The proposed moves from each of the captors are then combined, and the result is tested for legality. If the proposed moves for all the captors are legal, the moves are implemented synchronously. If not, the Global Agent is

invoked, and the moves proposed by the Global Agent are implemented. The new move data is then used to re-train the Captor Agents.

The fugitive agent has no real intelligence; it merely chooses a (legal) move at random.

The system is thus a hybrid Centralised/Swarm system. The centralized system is effectively used to train the multi-agent system. The advantage of this arrangement is that the centralized system has a global view of the Pursuit board. The centralised system, as represented by the Global Agent is programmed to favour a solution that minimises the sum of all the captor's distances from the fugitive. In other words, it looks for an arrangement of the captors that will maximally surround the fugitive. If a pure swarm system approach were to be followed, a likely implementation would be one where each agent attempts to minimize its own distance from the fugitive. This will cause agents to get in each others way, by blocking, or will result in an inferior strategy by the agents, since they are not prepared to make sacrifice to their positions, even if it would result in a globally better arrangement of the captors.

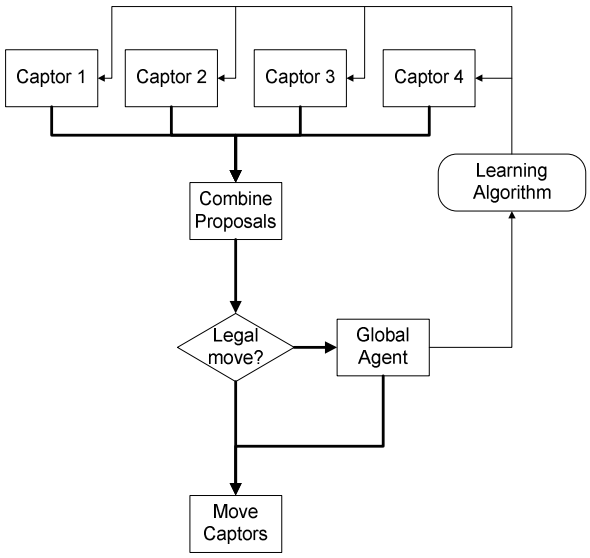


Figure 2 System Flow Diagram

Initially, as the system is run, it operates mainly as a centralised system, with the Global Agent proposing most of the moves. As time progresses, however, the Captor agents start performing more of the moves and the system edges towards a swarm system.

Captor Agents Description

The agents each contain a neural network. The network inputs are the current relative positions of all other agents (captors and fugitive) on the board to the agent. There are five

outputs for each network, each representing a particular direction in which the agent should move.

Board Representation and Training

The input representation to a neural network is extremely important to the success of practical network applications [13][16][17]. Smooth representation of the input data, as well as using an input representation that describes features of the data can help reduce the number of states the network has to learn in a lookup-table type fashion, and increase the ability of the network to generalise learned data to new situations [17].

For this study, we implement the board representation as a set of relative differences in position. Each Captor agent has the relative positions of the other captors inputted in order of Cartesian distance. The relative position of the fugitive is then given as the last two inputs. This is depicted in Figure 3.

This representation allows various formations of the agents to be captured, regardless of the exact location on the Pursuit board. Such formations are common when the captors are chasing the fugitive.

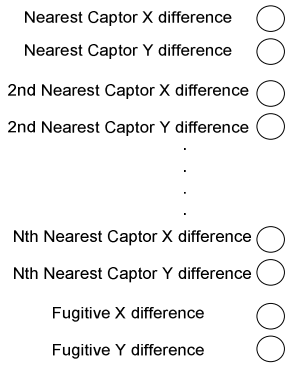


Figure 3 Network Input representation

The Captor Agents are trained on moves discovered by the Global Agent, using its genetic algorithm. Each Captor is trained with data specific to its own experience on the board. Training the agents from a centralised system means that the direct communication between agents can be drastically reduced. This is due to the fact that each agent will take into account other agents actions when moving. Therefore, in terms of communication, the system acts as a swarm system, but with one major difference: each agent is "aware" of the global goal through the training it has received from the Global Agent, and so inherently acts cooperatively with other agents.

Global Agent Description

The evaluation function for a chromosome of moves is calculated by summing the Euclidean distance of each of the Captor agents from the Fugitive agent. This is given by:

$$S = \sum_{N=1}^{M} \sqrt{(X_N - X_A)^2 + (Y_N - Y_A)^2} \quad (1)$$

Where: S is the sum of the distances to the fugitive; M is the number of Captor agents in the system; XA, YA are the Fugitive agent's coordinates and XN, YN are the current Captor's coordinates.

In order to ensure that illegal moves are discouraged, a penalty scheme is used where the sum is multiplied by a large number in order to create a peak in the evaluation function. Similarly, to ensure that a winning move (when possible) is chosen clearly above any other possible legal combination, such as move is heavily rewarded.

Implementation

The system is implemented in MATLAB. The Genetic Algorithm toolbox used is the open source GAOT [19]. NETLAB [18] is used to implement the neural networks for the Captor Agents. A two-layer MLP architecture is used for the networks. The networks use a Logistical function for the output units. A variable number of hidden units are used, depending on the amount of data the network has seen.

Since NETLAB does not directly support on-line training, an alternative scheme is used. Every time the Global Agent is invoked, the board position is recorded, as well as the moves proposed by the Global Agent. This data is added to the set of training data. After each game is complete, the agents are re-trained using the updated training data.

The results of training the network on a small subset of the Pursuit problem can be seen in Figure 4. Here, the agents repeatedly play the game, with the same starting position for each game. The agents thus learn all the permutations of the particular subset. In Figure 4, it can be seen that the agents make the majority of the moves after the 50th game, with between 80 -100% of the total moves made by the agents.

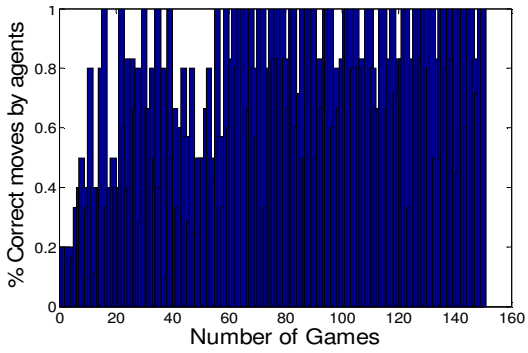


Figure 4 Number of correct game moves made by the agents for a limited subset of the Pursuit problem (normalised)

Figure 5 shows the percentage of game moves made by the agents during 350 games of pursuit. Although the result is not as successful as the limited sub problem as in Figure 4, it clearly demonstrates the ability of the agents to learn from the global agent.

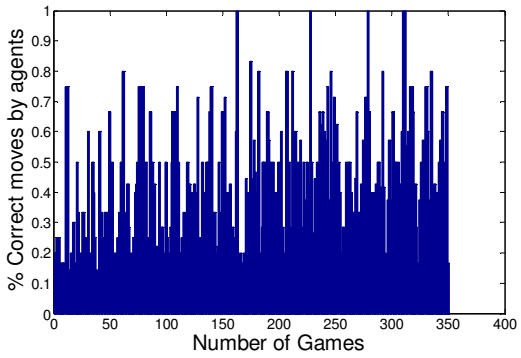


Figure 5 Number of correct game moves made by the agents for the game of Pursuit (normalised)

Discussion

Despite the fact that Pursuit is a very simple game, the successful suggestion of moves by the community of agents is quite significant. Firstly, to make a legal move, all the agents have to be aware of the rules of the game, which has not been explicitly programmed into them. Secondly, successful moves invariably require some degree of cooperation with neighbouring agents. Usually, this is achieved by explicitly using negotiation communication between agents. In this system, there is no such communication. Successful moves mean that the agents are able to anticipate each other's actions.

There are a very large number of possible board positions, given by the permutation of all the agents on the board over the blocks. This is given by:

$$ {}_{n}P_{k} = \frac{25!}{(25-5)!} = 6{,}375{,}600 \tag{2} $$

where we are using a 25 block board with 5 moveable agents (captors + fugitive). This is a large number of positions to learn from a finite set of data, and therefore methods of input representation compression should be investigated.

After a large number of games, each agent will have observed a large variety of board positions, and for this system, the neural networks will tend to be the same. This is only because the agents in this system all have the same "abilities" or allowed moves. In a scenario where there are agents with different abilities, as for the separate pieces in chess, the networks will each train on totally different output moves, and so will tend to be different even after observing an infinite number of moves. However, there is no reason why the agents should not still be able to anticipate each others moves, since they will be trained with this data.

Recommendations and Further work

On many of the occasions, the agents were not able to make a legal move due to the fact that some agents suggested more than one direction to move in at a time. While this intuitively suggests that the agents are not yet well trained in the particular situation, this is often not the case. The genetic algorithm does not always propose the same result for a particular board position, since there may not be a unique global optimum move for a certain board position. This means that for a given board position, an agent may be trained with many possible moves. What is needed is a method to resolve the proposed multiple directions by an agent in a particular situation with the other agents.

Further work involves a methodology allowing the agents themselves to formulate a global strategy, without relying on the external Global agent. This could be a simple principle such as distributing the evaluations of the Genetic Algorithm populations among the agents, or more elaborate methods.

Conclusion

The study of a hybrid Centralised-Swarm agent community has been discussed in this paper. The implementation of the proposed architecture, in the form of a genetic algorithm

as the Global agent and neural networks as swarm agents, is applied to the game of pursuit.

We found that although the swarm, or Captor, agents are able to learn a large number of moves, the fact that more than one move may be applicable in a situation results in agents proposing all these possibilities in a situation. This needs to be resolved within the swarm system to result in a valid move, and reduce the number of calls to the Global system.

The possibility of reduced communication between agents, while still cooperating with each other is demonstrated here.

References

[1] Maria Chli, Philippe De Wilde, Jan Goossenaerts, Vladimir Abramov, Nick Szirbik, Pedro Mariano, Rita Riberio, "Stability of Multi Agent Systems", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2003, Volume 1, 2003, Pages: 551-556

[2] E. Bonabeau, M. Dorigo, G Theraulaz, Swarm Intelligence – From Natural to Artificial Systems, Oxford University Press, 1999.

[3] J.A. Marshall, Z. Lin, M.E. Brouke, B.A Francis, "Pursuit Strategies for Autonomous Agents", Proceedings of the 2003 Block Island Workshop on Cooperative Control. Springer-Verlag Series: Lecture Notes in Control and Information Sciences, 2004.

[4] C. B. Excelente-Toledo and N. R. Jennings "The dynamic selection of Coordination Mechanisms" Autonomous Agents and Multi Agent Systems 9, 2004, pp 55-85

[5] K.P. Sycara. "Multiagent Systems". AI Magazine, American Association for Artificial Intelligence, Summer 1998. pp 79-92.

[6] J.J. Castro–Schez, N.R Jennings, X. Luo, NR Shadbolt. "Acquiring Domain Knowledge for negotiating agents: A case study", International Journal of Human-Computer Studies 61 (1), 2004, pp 3-31.

[7] N.R. Jennings, M. Wooldridge, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", Autonomous Agents and Multi-Agent Systems, 3, 2000. pg 285-12

[8] N.R Jennings, M. Woodridge,"Intelligent Agents: Theory and practice", The Knowledge Engineering Review, 10 (2), pg 115-152, 1995.

[9] Hadelia,, P. Valckenaersa, M. Kollingbaumb, H. Van Brussel, "Multi-agent coordination and control using stigmergy," Computers in Industry 53, 2004, pp 75–96

[10] Nils. J Nilsson, Artificial Intelligence: A new synthesis, Morgan Kaufmann Publishers, San Francisco, California, 1998.

[11] Alison Cawsey, The Essence of Artificial Intelligence, Prentice Hall Europe, 1998.

[12] M. Jordan, C. Bishop, Neural Networks, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Last accessed 12 August 2003, ftp.publications.ai.mit.edu.

[13] J. Holland. “Adaptation in Natural and Artificial Systems”, University of Michigan Press, 1975.

[14] R. Kurzweil, The age of intelligent Machines, Massachusetts Institute of Technology, 1990. pp 231-234

[15] Sebastian Thrun, “Learning to Play the Game of Chess”, Advances in Neural Processing Systems 7, 1995.

[16] G. Tesauro, “Programming backgammon using self-teaching neural net”, Artificial Intelligence 134(2002) pp 181-199.

[17] G.Tesauro, “Practical Issues in Temporal Difference Learning”, Machine Learning 8 (1992), pp. 257-277.

[18] I. Nabney, Netlab – Algorithms for Pattern Recognition, John Wiley and Sons, New York, 2001

[19] The Genetic Algorithm Optimization Toolbox (GAOT) for Matlab, http://www.ie.ncsu.edu/mirage/GAToolBox/gaot/, Last accessed 19 May 2004.

C h a p t e r 3

REDUCING INTER-AGENT COMMUNICATION DUE TO NEGOTIATION THROUGH LEARNING

Abstract – *This paper studies the effect that agent learning can have on inter-agent communication in a Multi-agent system. The agents are equipped with Multi Layer Perceptron neural networks to learn solutions to problems that have been solved through explicit negotiation and communication. We implement a test problem in the form of a Pursuit game, where the Multi-Agent system is a set of captor agents. The result is up to 44% fewer negotiation sessions with learning-enabled agents. The importance of learning, in terms of agent knowledge and overall system effectiveness is discussed.*

Introduction

The concept of Multi-agent systems is an engineering paradigm that has been gaining momentum over the past years [1]. A particular form of Multi-Agent systems, Swarm based systems, have been successfully applied to a number of problems [2]

Swarm systems are popular because of the lightweight processing power needed for them. A particular advantage in some instances is the low level, or often no direct, communication between agents, especially when communication bandwidth is limited or costly [2]. In [3], for example, the time taken for a system during negotiation to solve a task for a particular planning scheme, GraphPlan, is noted. The amount of time can be seen to increase significantly on some problems. This is potentially very inefficient, especially if the agents face the same problem frequently. However, it is often considered an absolute fundamental attribute of agents to be able to communicate with other agents in the system to improve the overall performance of the system [4].

In this paper, a study is performed on the effect that learning has on communication between agents. It is reasoned that once an agent community has solved a particular problem successfully, there should be no reason to repeat all the steps that led to the solution. In other words, the agents should be able to re-use solutions that they have discovered previously. In the experiment described here, the agents first negotiate a solution for each problem they are presented. Each agent then learns the problem situation from its point of view, and the resultant action that it took after successful negotiation with

other agents. Once they have learned the appropriate behaviour, they need not perform the task of negotiation for that situation, as long as the same conditions hold. It is reasoned that this can significantly cut down on inter-agent communication traffic.

Background

Multi Agent Systems

The multi-agent paradigm, in which many agents operate in an environment, has become a useful tool in solving large scale problems through a "divide and conquer" strategy [5][5]. The Multi-agent system is a distributed, decentralised system. The paradigm of individual entities collaborating to solve a particular problem that is beyond each entity's own capabilities is a natural concept, and one that is proving to be very powerful in practice [6]. However, while the concept is easily understandable, the implementation is not trivial. There are many complexities and subtleties in these such as [5]:

- Decomposing and allocating problems to the agents
- Describing the problem to the agents
- Enabling communication and interaction among the agents
- Decentralised systems, in the context of Multi-Agent systems, promise the following advantages [5][7][8][9][10]
- No single failure point, therefore greater robustness. Multi-agent systems have the capacity to degrade gracefully.
- Possibility of faster response times and fewer delays as the logic/intelligence is situated nearer to the problem domain.
- Increased flexibility to take account of changes occurring in the problem domain.
- Modularity and Scalability. Multi-agent systems can be increased in size dynamically according to the demands of the problem.

The problems associated with Multi-agent systems are [1][2][5][10]:

- Difficulty in measuring and evaluating the stability and security of the system.
- Excessive communication between agents can slow down the system. This is often countered by heavily restricting the amount of communication between agents.
- Possibility of getting stuck in non-optimal solutions of the problem, often due to the lack of global knowledge of the problem from each agent's point of view.

Most Multi-agent systems are built in an ad-hoc way since there is no absolute theory for these types of systems. There have been recent attempts however, to formalise the design of agent based systems, such as the Gaia Methodology [7]. However, these are not yet in widespread use.

Swarm Based systems

Swarm intelligence is a particular paradigm for multi-agent systems which emphasizes distributedness and agent simplicity. It is based on the observations of social insects in nature, such as ants, termites and bees [2]. Such insect societies are extremely organized, even though there is no central control or planning. Each agent in the system is programmed only to achieve its own Local Goal. The agent's behaviours in Swarms are very simple: the intelligence of the system is 'emergent' from the overall behaviours of all the agents in the system. The communication between agents is usually performed indirectly [9], by agents making changes to the environment, which other agents act upon. This is analogous to insects laying pheromone trails to food source, etc. Emphasis is therefore placed on reactivity in these systems. Swarm systems have been successfully applied to many problems, notably routing in computer and telecoms networks [2] and recently to a manufacturing control system [9].

The disadvantage of swarm based systems is that no agents actually have a global view of the problem to be solved. All agents are entirely focused on achieving their own Local Goals, whether or not these goals are to the benefit or detriment to the overall community. This can exacerbate the problem of the system getting stuck in local optima, or worse, cause the system to fail.

Neural Networks

Artificial neural networks are inspired by the functioning of biological neurons in animal and human brains. Artificial neural networks differ from most other computing techniques in that they are able to learn arbitrary relations between sets of data presented to them. That is, rules are not explicitly programmed or set, but are learned from experience by the network [10]. The basic architecture of a neural network is also based on that of their biological counterpart: both networks consist of many simple elements, known as neurons, operating in parallel [12][13]. The most widely used neural network models are the Multi-layer perceptron (MLP) and Radial Basis Function (RBF) networks.

Hypothesis and Method

Reference [11] states that the method of human learning when presented by a new task is to use rational reasoning to perform the decision making process behind solving the task.

After using this deliberative approach, and we begin to "master" a task, we are able to perform it "naturally", without explicitly performing rational reasoning. At this point, people use their pattern recognition skills to perform the task.

It is hypothesised that the human model of discovery and learning can be applied to agent strategising. An algorithmic discovery stage using negotiation between agents can be used on new problems. This will be the deliberative stage, where a problem solution is formulated. Once a strategy has been discovered and used, a neural network can be trained with the results. This is the Pattern Recognition stage. Thus over time, the neural networks should learn a large number of strategies, while generalising these strategies to other scenarios. Thus, as agents perform more tasks, the problem solving for known tasks should become "natural" to the entire community, as the neural networks start taking over from the negotiation modules.

Our proposal is to create a system that uses negotiation to enable the agents to find solutions to problems they encounter, and then learn this solution, in order to avoid the necessity of re-negotiating a solution, with the resultant communication and time costs, if the problem occurs again. Learning is considered to be an important feature of true autonomous agents [5]. One aspect of learning can be expressed as the ability to perform old tasks better as a result of learning and observing [4]. In this case it can be interpreted as the agents learning to perform old tasks more efficiently.

Test Problem

The proposed system is applied to the game of Pursuit. A Pursuit board is represented by a 2 dimensional grid pattern, as shown in Figure 6. The asterisks represent the Captor agents, while the circle represents the Fugitive agent. The aim of the game is for the Captors to surround or corner the Fugitive. The fugitive and captors take alternate turns to move. Each agent may only move one block per turn. Legal moves on a Pursuit board are Up, Down, Left, Right and Stay. No diagonal moves are allowed. Furthermore, the board does not wrap around. A game is ended when the fugitive is captured, implying that the fugitive has no place to move to.

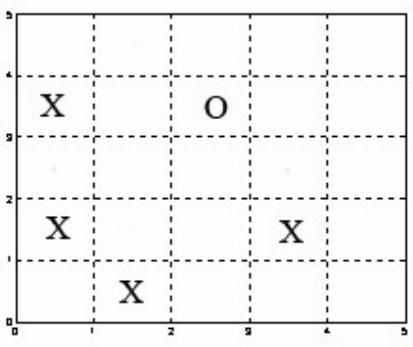


Figure 6 Pursuit Board representation

System Description

The system has four separate agents representing the captors. Each of the agents has a neural network that is trained independently of other agents, as well as a negotiating algorithm. Each agent is responsible for making a valid move. This is defined as a move that is within the bounds of the board, and does not infringe on any block that another agent is in.

The fugitive agent has no real intelligence; it merely chooses a (legal) move at random.

The agents interact by proposing the move that they would prefer to make. This is sent to all other agents. If there is a conflict among the proposals, the agents send out a signal, and the agents negotiate a new proposal. This occurs until there are no conflicts among the agents. The agents then implement their moves. The first move proposed for each agent is always drawn from the neural network. If this is not successful, the agents start the negotiations.

Captor Agent Description

As mentioned above, the Captor Agents consist of two parts: a pattern recognition unit, implemented as a neural network, and a negotiating algorithm that solves disputes with other agents.

As can be seen in Figure 7, the agents start off by proposing the solutions predicted by their neural networks. Only if this fails is the negotiation algorithm invoked. Whenever there is a conflict, all the agents are involved in the negotiation. The dotted feedback lines from the negotiation block to the neural network represents the network learning from the successful move made by the negotiation algorithm. In this way, the neural network learns cooperative behaviour.

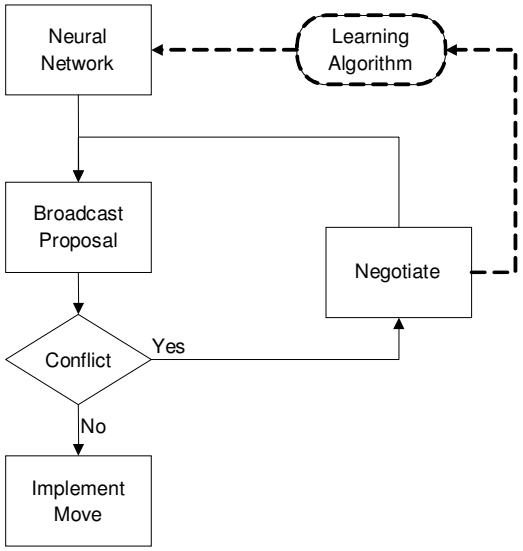


Figure 7 High Level Agent execution

The negotiation algorithm is shown in Figure 8. The agents each have a local goal based on the global goal of the system. The global goal of the system is to win the game, i.e. capture the agent. In this system the global goal is expressed as the minimisation of the distance of all the Captor agents from the Fugitive agents. This is given by:

$$S = \sum_{N=1}^{M} \sqrt{(X_N - X_A)^2 + (Y_N - Y_A)^2} \qquad (1)$$

Where: S is the sum of the distances to the fugitive; M is the number of Captor agents in the system; XA, YA are the Fugitive agent's coordinates and XN, YN are the current Captor's coordinates.

If the captors follow this rule, they will win on a finite sized board, although not always in the least number of moves.

It would follow that if each captor agent minimised its own distance from the fugitive, the global goal would be satisfied. However, since each block on the board can hold only one agent exclusively, not all agents will be able to reach their own minimum on every board state. Therefore, to minimise the function in reality, some agents may not be in their optimal positions. In a Centralised system, where one agent controls the movements of all the Captors, this would be straightforward to implement, as the controller would have a global view of the problem, as well as only having to act in its own best interest. However, in a decentralised or Multi-Agent system, the implementation is not as clear. Since each

agent will have its own local goal to pursue, it is not clear what the formulation of this local goal would be in order to reap the maximum benefit for the entire community [14].

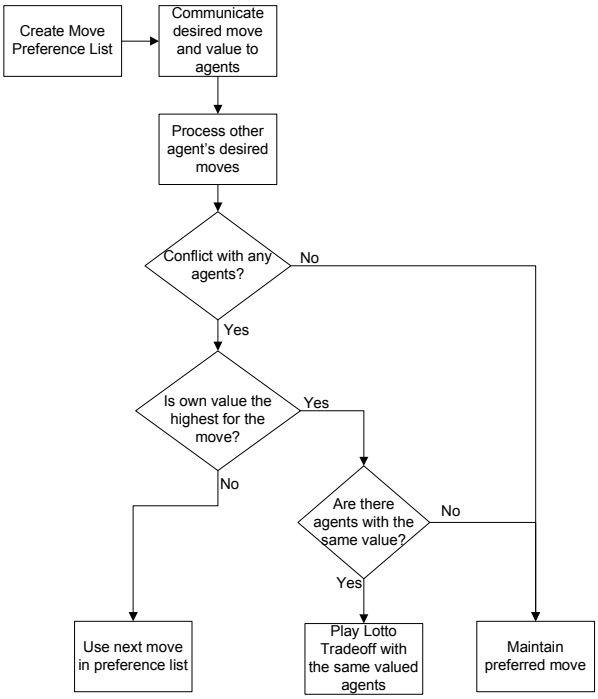


Figure 8 Agent Negotiating Algorithm

In this case, the local goal of each captor agent is to minimise its own distance from the fugitive through negotiation with the other Captors. The negotiation algorithm of each agent is depicted in Figure 8. The agents are aware of the possible moves they can make at each turn. By evaluating the payoff that each of the possible moves will bring, the agents construct a Preference List of the possible moves and payoffs ordered from highest payoff (least distance) to lowest (furthest distance). Once the agent has made its list, it transmits its preferred move (which is the block on the board it would like to occupy), and the payoff value of the next preferred move on the list. In this way, the agent expresses its "need" for the block. The agent then creates a list of all the moves and payoffs received from the other agents. If there is a conflict between itself and one or more other agents on the choice of blocks, the agent attempts to resolve this. An agent will give way to the other agent(s) if it does not value the position as highly as any other conflicting agent (i.e, if it does not need the block as badly as other agents). An agent gives way by using the next move on its preference list. Once it gets to the end of its list, it wraps around to the start. If there is a tie for the highest value on the block, the agents involved in the tie go into a "lotto" mode, where they each randomly pick a number, with the highest number winning

the right to keep the block. The losing agents then choose the next move on their Preference list. This procedure is iterated until no conflicts exist. At this point the agents implement their moves. In this manner, an approximation to the global optimum is performed. This sequence can be performed a number of times, which can be seen to create many messages between agents, and thus a high usage of the communication medium.

From this, it can be seen that the neural network of each agent learns solutions to board problems that are not only legal, but also cooperative with other agents on the board. Once these moves have been learned, the agents need only inform each other of the predicted move, and if there are no conflicts, proceed with the move. This then eliminates the need for the explicit negotiation transmissions.

Board Representation and Training

The input representation to a neural network is extremely important to the success of practical network applications [15][16][17]. Smooth representation of the input data, as well as using an input representation that describes features of the data can help reduce the number of states the network has to learn in a lookup-table type fashion, and increase the ability of the network to generalise learned data to new situations [17].

For this study, we implement the board representation as a set of relative differences in position. Each Captor agent has the relative positions of the other captors inputted in order of Cartesian distance. The relative position of the fugitive is then given as the last two inputs. This is depicted in Figure 9.

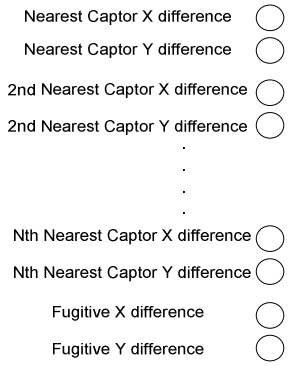


Figure 9 Network Input representation

This representation allows various formations of the agents to be captured, regardless of the exact location on the Pursuit board. Such formations are common when the captors are chasing the fugitive.

Each Captor is trained with data specific to its own experience on the board. Training the agents from their past experience means that the direct communication between agents can be drastically reduced. This is due to the fact that each agent will take into account other agents moves when moving itself. Therefore, in terms of communication, the system acts as a swarm system, but with one major difference: each agent is "aware" of the global goal through the experience it has gained through negotiation, and so inherently acts cooperatively with other agents.

Implementation

The system is implemented in MATLAB. NETLAB [19] is used to implement the neural networks for the Captor Agents. A two-layer MLP architecture is used for the networks. The networks use a Logistic function for the output units. A variable number of hidden units are used, depending on the amount of data the network has seen.

Since NETLAB does not directly support on-line training, an alternative scheme is used. Every time the agents negotiate a solution, the board position is recorded, as well as the moves agreed upon through negotiation. This data is added to the set of training data. After each game is complete, the agents are re-trained using the updated training data.

Results

The simulation was run for 300 games. The results are summarised in Table 1. The measure of effectiveness used is the average number of game moves per game successfully predicted by the neural network. This is shown as the Average Learned moves/Game. The average number of moves per game when not using learning, i.e. using only the negotiation modules, is 6.5 over 300 games. This figure is used as the control number, to measure the quality of the predicted moves. An average of the number of moves over a large number of games is used, since the exact number of moves can vary substantially, depending on the starting parameters of each game. If the average number of moves per game increases from this control, it implies that learning is actually having a detrimental effect on the agent's performance. We would expect that the average number of moves per game remain consistent between the learning enabled, and non- learning systems. This is due to the fact that the agents should be learning the game tactics demonstrated, and not creating new strategies. The bandwidth reduction is calculated by considering each negotiation session as using one unit of bandwidth. Therefore, if the agents make the moves without negotiation, then they will be saving one unit of bandwidth. We define this arbitrary measure, as the actual amount of data transferred will differ from one agent system to the next, and so a precise byte count will be meaningless.

The results of the learning simulation are broken into 3 stages of the game: 0 -100 games, in which it can be seen that the neural networks are still learning appropriate moves. The result is that the average number of moves per game in this phase actually increases from the control number, and it is for this reason that a percentage saving in bandwidth is not given, as it is meaningless in this case. A suggestion is this case would be to ignore the neural network's proposal until a critical number of training data has been obtained.

In the second phase, from game number 100 -200, it can be seen that the neural networks have learned sufficient appropriate moves to be effective in the system. In this phase, the average moves/game is at the level of the Control, but the amount of communication is decreased. This shows that the learning is useful in this stage, as the number of game moves remains at our expected control value, but the number of valid suggestion provided by the agents is increasing.

In the third stage, the neural networks account for nearly half of the game moves, while the overall moves per game is kept at the Control level. This represents a substantial reduction in negotiation communication.

Table 1 *Breakdown of Simulation Results after 400 games*

	Game numbers		
	0 -100	100- 200	200-300
Average moves/ game	7.6	6.3	6.23
Average learned moves/game	2.4	1.92	2.7
% Bandwidth Reduction	N/A	30%	44%

Analysis

The knowledge that the agents have obtained as a result of their learning can be thought of as obtaining a model of their environment, or more specifically, as a model of the behaviour of the other agents within the system. This is due to the fact that when the agents correctly predict moves using their neural networks, they are not only maximising their own goal, but taking into account the needs of the other agents in the system. This is despite the fact that they have no explicit knowledge of the behaviour or workings of the

other agents: the knowledge that they have obtained is purely through observation and interaction. In the negotiation, or deliberative algorithm, the agents take no account of the moves that the other agents might make: they wait for another agent to actively inform of a problem. To programme into such algorithms the ability to anticipate other agent's moves would effectively mean that the agents would each have a global view of the problem, as well as having intimate knowledge of every agent's abilities and desires in the system. This would defeat the object of a Multi-agent system, where each agent is expected to operate in a highly distributed environment, and having a limited viewpoint of the problem, as well as considerable duplication of abilities. Furthermore, it may not even be appropriate for each agent to have this knowledge, especially when the agents are divided into sub-teams [3].

After a large number of games, each agent will have observed a large variety of board positions, and for this system, the neural networks will tend to be the same. This is only because the agents in this system all have the same "abilities" or allowed moves. In a scenario where there are agents with different abilities, as for example the separate pieces in a chess game, the networks will each train on totally different output moves, and so will tend to be different even after observing an infinite number of moves. However, there is no reason why the agents should not still be able to anticipate each others moves, since they will be trained with this data.

The reduction in communication as shown here can have large benefits in certain situations, such as when the communication channel carries a significant monetary cost, or if the channel is not reliable or slow. All these factors contribute to the overall effectiveness of the agent community, and should be minimised.

Future research

There is already considerable research in the field of agent learning. We believe that learning is not only useful for added functionality in agents, but also to enhance existing agent operation. The main problem in any learning system is often the presentation of the information to the system [15]. With other means of representation, it may be possible to increase the benefits of learning, and create more efficient agents. In complicated environments, which are common of multi-agent domains, it would be desirable, and probably necessary, to have the agents automatically determine the relevant representation of information, as well as choosing the information that would help in the decision.

Furthermore, the area of agent coordination is a currently an active area of research. A methodology whereby agents are equipped with a host of coordination strategies at design

time, and then learn through interaction the appropriate local strategy to use in a given situation has been proposed by [18]. A mechanism to accommodate this type of agent behaviour will need to be investigated.

Conclusion

The possibility of reduced communication between agents, while still cooperating with each other is demonstrated in this paper. We show that learning can be successfully applied to this aim. This results in agents that effectively anticipate other agent's behaviours, despite having no explicit model of the agents programmed.

A reduction in communication of up to 44% is achieved in this study. This level of reduction can have extremely positive effects if communication in the system is in some way a significant cost factor in the overall effectiveness of the system.

References

[1] Maria Chli, Philippe De Wilde, Jan Goossenaerts, Vladimir Abramov, Nick Szirbik, Pedro Mariano, Rita Riberio, "Stability of Multi Agent Systems", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2003, Volume 1, 2003, Pages: 551-556

[2] E. Bonabeau, M. Dorigo, G Theraulaz, Swarm Intelligence – From Natural to Artificial Systems, Oxford University Press, 1999.

[3] D. Kalofonos, T.J. Norman, "An Investigation into Team-Based Planning," in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2004, pp 5590- 5595.

[4] Z. Ren, C.J. Anumba, "Multi-agent systems in construction – state of the art and prospects," Automation in Construction, no. 13, 2004, pp. 421-423.

[5] K.P. Sycara. "Multiagent Systems". AI Magazine, American Association for Artificial Intelligence, Summer 1998. pp 79-92.

[6] J.J. Castro–Schez, N.R Jennings, X. Luo, NR Shadbolt. "Acquiring Domain Knowledge for negotiating agents: A case study", International Journal of Human-Computer Studies 61 (1), 2004, pp 3-31.

[7] N.R. Jennings, M. Wooldridge, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", Autonomous Agents and Multi-Agent Systems, 3, 2000. pg 285-12

[8] N.R Jennings, M. Woodridge,"Intelligent Agents: Theory and practice", The Knowledge Engineering Review, 10 (2), pg 115-152, 1995.

[9] Hadelia,, P. Valckenaersa, M. Kollingbaumb, H. Van Brussel, "Multi-agent coordination and control using stigmergy," Computers in Industry 53, 2004, pp 75–96

[10] Nils. J Nilsson, Artificial Intelligence: A new synthesis, Morgan Kaufmann Publishers, San Francisco, California, 1998.

[11] R. Kurzweil, The age of intelligent Machines, Massachusetts Institute of Technology, 1990. pp 231-234

[12] Alison Cawsey, The Essence of Artificial Intelligence, Prentice Hall Europe, 1998.

[13] M. Jordan, C. Bishop, Neural Networks, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Last accessed 12 August 2003, ftp.publications.ai.mit.edu.

[14] J.A. Marshall, Z. Lin, M.E. Brouke, B.A Francis, "Pursuit Strategies for Autonomous Agents", Proceedings of the 2003 Block Island Workshop on Cooperative Control. Springer-Verlag Series: Lecture Notes in Control and Information Sciences, 2004.

[15] Sebastian Thrun, "Learning to Play the Game of Chess", Advances in Neural Processing Systems 7, 1995.

[16] G. Tesauro, "Programming backgammon using self-teaching neural net", Artificial Intelligence 134(2002) pp 181-199.

[17] G.Tesauro, "Practical Issues in Temporal Difference Learning", Machine Learning 8 (1992), pp. 257-277.

[18] C. B. Excelente-Toledo and N. R. Jennings "The dynamic selection of Coordination Mechanisms" Autonomous Agents and Multi Agent Systems 9, 2004, pp 55-85

[19] I. Nabney, Netlab – Algorithms for Pattern Recognition, John Wiley and Sons, New York, 2001

C h a p t e r 4

USING LEARNING IN MULIT-AGENT COMMUNITIES TO IMPROVE COMMUNICATION EFFICIENCY

Abstract *- This paper studies the effect that agent learning can have on communication efficiency in a cooperative Multi-agent system. The agents are equipped with Multi-layer Perceptron (MLP) neural networks to learn solutions to problems that have been solved through explicit negotiation and communication in one system, and delegated solutions in a slave/master type system. We implement two test problems in the form of a Pursuit game, where the Multi-Agent system is a set of captor agents. In the first system, the agents negotiate in a peer to peer fashion to solve problems. In the second system, the agents are instructed by a centralised global agent. In both systems the agents are able to learn the solutions. The results are: up to 54% of the solutions are learned and reused by the agents and up to 55% fewer negotiation sessions with learning-enabled agents. This can result in lower bandwidth costs, and improved efficiency of multi-agent system. The importance of learning, in terms of agent knowledge and overall system effectiveness is discussed.*

Introduction

Multi-agent systems are becoming a popular paradigm for many complex tasks [1]. The notion of enabling many agents to work together either for a common goal or in a common environment is intriguing, and challenging. This concept has been applied to many diverse applications from stock market simulation to manufacturing control and construction [2][3][4][5].

A common problem in multi-agent systems is that of decomposing the main task, also known as the global goal, into behaviours for each of the agents in the systems, also known as the local goals [6]. This problem is significant, when it is considered that in most systems, the agents have only local knowledge of the current problem at hand. Furthermore, agents often need to act cooperatively with one another to ensure that the tasks can be solved [7].

Commonly, two approaches are used to address the problem of translating local goals into benefit to the whole community. The first is where the agents are enabled to negotiate and

bargain in a peer-to-peer fashion, where if conflicts arise between agents, the agents come to a compromise to solve the conflict locally. In this way, the agents can cooperate in achieving their local goals [8]. The other method is to define hierarchies in agent communities, where one agent is a "leader" which solves the problem optimally for the community under its control using information gathered from other agents. The leader then delegates the resultant plan to each agent to implement. This is commonly known as a master/slave relationship [2].

Both approaches, however, have some drawbacks. In the negotiation case, the communication between the agents can become excessive, which is not only time consuming but also may carry a monetary cost in terms of the bandwidth used. The second approach has a similar drawback, since the communication between the "leader" agent and the other agents can be large, since each agent will have to send all of its information to the leader in order that a proper representation of the situation is gained. Furthermore, the leader agent will commonly need a large amount of processing power to solve the problem optimally for all agents.

In fact, these problems are so significant that a new type of Multi-agent system, known as Swarm systems, has become popular. These systems emphasise "emergent" behaviour; where agents do not directly communicate with each other, and agents have simple local goals [9]. The agents in these systems are generally reactive. This is attractive for two main reasons: the low level of communication required, and the lightweight processing power of the agents.

In this paper, a study is performed on the effect that learning has on communication between agents. It is reasoned that once an agent community has solved a particular problem successfully, there should be no reason to repeat all the steps that led to the solution. In other words, the agents should be able to re-use solutions that they have discovered previously. We propose that machine learning can be used to enable the agents to recall solutions to problems that they have encountered previously. We conduct 2 related experiments in this paper. In both experiments the agents are equipped with neural networks to learn solutions to problems.

The first experiment is an agent community with a centralised controller to find solutions to problems the agents are presented with, and delegate the implementation of the solution to each agent. We hypothesise that learning can reduce the number of times the central agent has to be invoked for repetitive or similar tasks.

The second experiment consists of a system of independent negotiating agents. In this experiment, the agents first negotiate a solution for each problem they are presented. Each

agent then learns the problem situation from its point of view, and the resultant action that it took after successful negotiation with other agents. Once they have learned the appropriate behaviour, they need not perform the task of negotiation for that situation, as long as the same conditions hold. It is reasoned that this can significantly cut down on inter-agent communication traffic.

In both experiments, we reason that over time the agents will learn cooperative behaviour, while minimising the amount of communication needed to achieve this, in effect becoming more of a Swarm type system, in terms of reactivity. Learning is considered to be an important feature of true autonomous agents [10]. One aspect of learning can be expressed as the ability to perform old tasks better as a result of learning and observing [2]. In this case it can be interpreted as the agents learning to perform old tasks more efficiently.

This paper is organised as follows. First we give a brief review of the Multi-agent System paradigm, and then describe the method we use for our experiments. We present the results and give a discussion of the simulations. We conclude the paper with proposals for further work, and a summary of our findings.

Multi Agent System Review

Multi Agent Systems

The multi-agent paradigm, in which many agents operate in a common environment, has become a useful tool in solving large scale problems through a "divide and conquer" strategy [10] The Multi-agent system is a distributed, decentralised system. The paradigm of individual entities collaborating to solve a particular problem that is beyond each entity's own capabilities is a natural concept, and one that is proving to be very powerful in practice [11]. However, while the concept is easily understandable, the implementation is not trivial. There are many complexities and subtleties in these such as [10]:

- Decomposing and allocating problems to the agents
- Describing the problem to the agents
- Enabling communication and interaction among the agents
- Decentralised systems, in the context of Multi-Agent systems, promise the following advantages [3][10][12][13][14]
- No single failure point, therefore greater robustness. Multi-agent systems have the capacity to degrade gracefully.
- Possibility of faster response times and fewer delays as the logic/intelligence is situated nearer to the problem domain.

- Increased flexibility to take account of changes occurring in the problem domain.
- Modularity and Scalability. Multi-agent systems can be increased in size dynamically according to the demands of the problem.

The problems associated with Multi-agent systems are [1][9][10][14]:

- Difficulty in measuring and evaluating the stability and security of the system.
- Excessive communication between agents can slow down the system. This is often countered by heavily restricting the amount of communication between agents.
- Possibility of getting stuck in non-optimal solutions of the problem, often due to the lack of global knowledge of the problem from each agent's point of view.
- Most Multi-agent systems are built in an ad-hoc way since there is no absolute theory for these types of systems. There have been recent attempts, however, to formalise the design of agent based systems, such as the Gaia Methodology [12]. Currently though, these are not yet in widespread use.

Swarm Based systems

Swarm intelligence is a particular paradigm for multi-agent systems which emphasizes distributedness and agent simplicity. It is based on the observations of social insects in nature, such as ants, termites and bees [9]. Such insect societies are extremely organized, even though there is no central control or planning. Each agent in the system is programmed only to achieve its own Local Goal. The agent's behaviours in Swarms are very simple: the intelligence of the system is 'emergent' from the overall behaviours of all the agents in the system. The communication between agents is usually performed indirectly [3], by agents making changes to the environment, which other agents act upon. This is analogous to insects laying pheromone trails to food source, etc. Emphasis is therefore placed on reactivity in these systems. Swarm systems have been successfully applied to many problems, notably routing in computer and telecoms networks [9] and recently to a manufacturing control system [3].

The disadvantage of swarm based systems is that no agents actually have a global view of the problem to be solved. All agents are entirely focused on achieving their own Local Goals, whether or not these goals are to the benefit or detriment to the overall community. This can exacerbate the problem of the system getting stuck in local optima, or worse, cause the system to fail. Furthermore, choosing local strategies to create emergent swarm behaviour is not trivial or even feasible.

Method

In this paper we conduct two experiments into the value of learning in Multi-agent communities.

In the first experiment, we use an agent community which is guided by a Central controller. In this experiment, we attempt to reduce the agents' reliance on the central controller through learning.

In the second experiment, the agents cooperate through negotiation. We attempt to reduce the amount of negotiation over time by enabling the agents to learn the outcomes of previous negotiation sessions.

We use the same test problem, described below, for both experiments. In both studies, the agents are equipped with neural networks as the learning mechanism.

We first give a brief overview of neural networks as our choice for a learning mechanism.

Neural Networks

Artificial neural networks are inspired by the functioning of biological neurons in animal and human brains. Artificial neural networks differ from most other computing techniques in that they are able to learn arbitrary relations between sets of data presented to them. That is, rules are not explicitly programmed or set, but are learned from experience by the network [14]. The basic architecture of a neural network is also based on that of their biological counterpart: both networks consist of many simple elements, known as neurons, operating in parallel [15][16]. The most widely used neural network models are the Multi-layer Perceptron (MLP) and Radial Basis Function (RBF) networks.

The basic element of a neural network is the neuron. A representation of a neuron is shown in

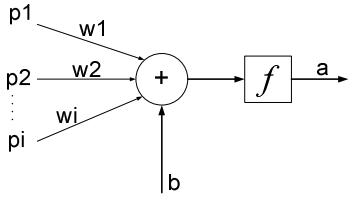


Figure 10figure 10.

A neuron may have an arbitrary number of inputs p_i. These inputs to the network are weighted, which is to say they are multiplied by a constant value w_i. The weighted inputs to the neuron are then summed, along with the bias b of the network. The bias is a constant value added to the summation of the neuron inputs. The resultant summation is then acted

on by an activation function f. This transfer function may be linear or nonlinear. In simple mathematical terms, the neuron can be described as:

$$a = f(w_1 p_1 + w_2 p_2 + w_3 p_3 + \ldots + w_i p_i + b) \quad (1)$$

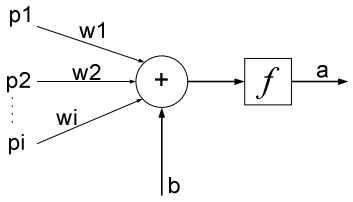


Figure 10 Neuron Representation

The output of the neuron may then be used as the input to further neurons, or as the output of the neural network. Generally many neurons are connected in layers to create a neural network, which is a number of neurons acting in parallel, receiving the same input data [17]. Each layer of a network feeds into a subsequent layer, or in the case of an output layer, to the outside world. Not all layers in a network need to have the same number of neurons. Furthermore, the activation functions of each neuron need not all be the same [17]. The last layer of the network is known as the Output layer. The point of input of the network is known as the Input layer, and the layers between the inputs and the output layer are known as the hidden layers of the network.

Training

An MLP network is trained by adjustment of the weights and the biases of each neuron in the network. For training, two basic methods are used: supervised training and unsupervised training [16].

With supervised training, the network is presented with inputs and targets. The targets are the desired outputs of the network. The aim is then to minimise the error between the actual network output and the target values, expressed by some cost function, by adjusting the weights and biases of the network.

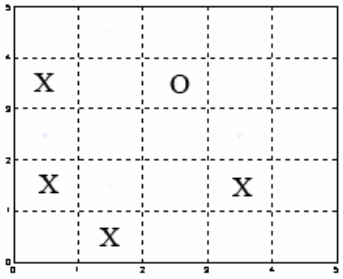
In the case of unsupervised training, the network is not given targets; instead the aim is for the network to categorise and classify inputs into a number of classes.

For supervised training, adjustment of the weights and biases is achieved by a training algorithm. A commonly used training algorithm is *backpropagation* [16]. Backpropagation, which is a form of optimisation based on steepest descent algorithms, updates the weights and biases from the output layer of the network towards the input

layer at each training cycle or epoch, hence the name backpropagation. The algorithm relies on the activation functions being differentiable, as the weights are updated in proportion to the derivatives calculated from the output of the network to the input. Improvements to the standard backpropagation involve using more efficient non-linear optimisation techniques for finding the weights. These include the Scaled Conjugate Gradient (SCG) method and Quasi-Newton methods [18].

Test Problem

The test problem is a multi-agent game of Pursuit. A Pursuit board is represented by a 2 dimensional grid pattern, as shown in Figure 11. The crosses represent the Captor agents, while the circle represents the Fugitive agent. The aim of the game is for the Captors to surround or corner the Fugitive. The fugitive and captors take alternate turns to move. The captors must move synchronously on their turn. Each agent may only move one block per turn. Legal moves on a Pursuit board are Up, Down, Left, Right and Stay. No diagonal moves are allowed. Furthermore, the board does not wrap around. A game is ended when the fugitive is captured, implying that the fugitive has no place to move to.



F*igure 11 Pursuit Board Representation*

Global Solution Strategy

The global goal of the system is to win the game, i.e. capture the agent. In this system the global goal is expressed as the minimisation of the sum of the distances of all the Captor agents from the Fugitive agents. This is given by:

$$S = \sum_{N=1}^{M} \sqrt{(X_N - X_A)^2 + (Y_N - Y_A)^2} \qquad (2)$$

Where: S is the sum of the distances to the fugitive; M is the number of Captor agents in the system; X_A, Y_A are the Fugitive agent's coordinates and X_N, Y_N are the current Captor's coordinates. We assume here that a signal describing the current state of the board has the Markov property, meaning all relevant information pertaining to the current state of game is encapsulated in this signal. Thus, if the captors positions are updated using

this rule of minimising the sum of their distances from the fugitive on each time step (turn), they will win on a finite sized board, although not always in the least number of moves.

It would follow that if each captor agent minimised its own distance from the fugitive at each turn, the global goal would be satisfied. However, since each block on the board can hold only one agent exclusively, and all the agents move synchronously, there may be conflicts for board space among the agents. Therefore, to minimise the function in reality and to maintain a legal board, some agents may not be in their optimal positions. In a Centralised system, where one agent controls the movements of all the Captors, this is straightforward to implement, as the controller would have a global view of the problem, as well as only having to act in its own best interest. However, in a decentralised or Multi-Agent system, the implementation is not as clear. Since each agent will have its own local goal to pursue, it is not clear what the formulation of this local goal would be in order to reap the maximum benefit for the entire community [19]. However, it is clear that the agents would have to act in a cooperative spirit, in order to minimise the objective function.

If a pure swarm system approach were to be followed, a likely implementation would be one where each agent attempts to minimize its own distance from the fugitive, and all agents move asynchronously. This will cause agents to get in each others way, by blocking, or will result in an inferior strategy by the agents, since they are not prepared to make sacrifice to their positions, even if it would result in a globally better arrangement of the captors.

System Descriptions

Centralised System

As shown in the system flowchart in Figure 12, this system consists both of separate captor agents, as well as a global agent. The global agent is effectively used to train the multi-agent system. The advantage of this arrangement is that the global agent has a global view of the Pursuit board. The global agent uses a brute-force search through all possible move combinations to find a solution that minimises the sum of all the captor's distances from the fugitive. In other words, it looks for an arrangement of the captors that will maximally surround the fugitive. The flow of the system is as follows: at each turn, the captors propose a move for themselves. The moves from all of the captors are combined and tested to see if the resulting board configuration is legal. If the proposals do result in a legal board, they are implemented by the agents. However, if an illegal board results from

the proposals, the global agent is invoked to solve the problem. The solution from the global agent is then used to train the captors.

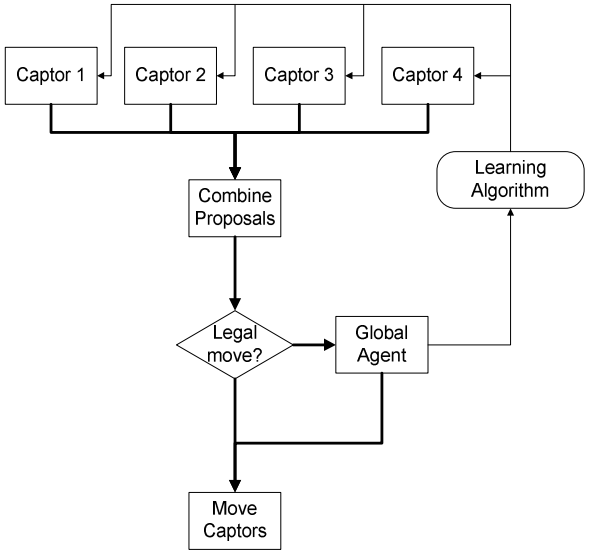


Figure 12 High level Agent execution in the Centralised system

Negotiating System

The agents interact by proposing the move that they would prefer to make, i.e., the move that would bring the agent closest to the fugitive. This is sent to all other agents. If there is a conflict among the proposals, the agents send out a signal, and the agents negotiate a new proposal. This occurs until there are no conflicts among the agents. The agents then implement their moves. The first move proposed for each agent is always drawn from the neural network. If this is not successful, the agents start the negotiations.

As can be seen in Figure 13, the agents start off by proposing the solutions predicted by their neural networks. Only if this fails is the negotiation algorithm invoked. Whenever there is a conflict, all the agents are involved in the negotiation. The dotted feedback lines from the negotiation block to the neural network represents the network learning from the successful move made by the negotiation algorithm. In this way, the neural network learns cooperative behaviour.

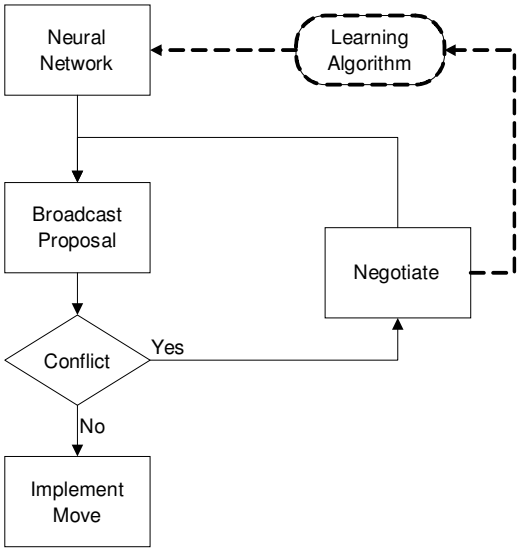


Figure 13 High level Agent execution in the Negotiating System

Captor Agent Description

In both experiments, the agents are equipped with neural networks to learn the solutions to the game in a pattern recognition fashion. The agents in the Negotiating system are also equipped with negotiation modules to bargain with the other agents.

Input representation

The input representation to a neural network is extremely important to the success of practical network applications [20][21][22]. Smooth representation of the input data, as well as using an input representation that describes features of the data can help reduce the number of states the network has to learn in a lookup-table type fashion, and increase the ability of the network to generalise learned data to new situations [22].

The agents each have 28 binary inputs to represent the current board state. The board is presented to the agents from their own point of view. Each agent can be thought of as having "sensors" to perceive their surroundings. The first 12 inputs to an agent represents if any other captor agents are in a 2-block radius around the agent. This is depicted in Figure 14, where the X represents a captor agent on the board. The light grey circle represents the 2 block radius. Only blocks that fall completely into this radius can be accurately detected by the agent, therefore only these are used as inputs. These input blocks are represented by the 12 darker grey shaded blocks. If a block is occupied by another captor agent, it is represented as a binary 1; else it is shown as a binary 0. The same scheme is used for the next 12 inputs, which represent the position of the fugitive

relative to the agent. The remaining 4 sensors are binary wall sensors, to inform the agent if it is at any of the four sides of the board. This type input structure is true to a multi-agent system, since each agent has only local knowledge of the game, and it also allows our agents to be trained with data specific to its own experience on the board. Furthermore, agents can obtain all the information they need from their sensors, and do not rely on external entities for information about the board state.

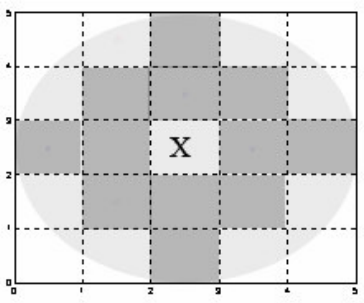


Figure 14 Captor agent input representation

This representation allows various formations of the agents to be captured, regardless of the exact location on the Pursuit board. Such formations are common when the captors are chasing the fugitive.

Negotiating Agents Description

In this case, the local goal of each captor agent is to minimise its own distance from the fugitive through negotiation with the other Captors. The negotiation algorithm of each agent is depicted in Figure 15. The agents are aware of the possible moves they can make at each turn. By evaluating the payoff that each of the possible moves will bring, the agents construct a Preference List of the possible moves and payoffs ordered from highest payoff (least distance) to lowest (furthest distance). Once the agent has made its list, it transmits its preferred move (which is the block on the board it would like to occupy), and the payoff value of the next preferred move on the list. In this way, the agent expresses its "need" for the block. The agent then creates a list of all the moves and payoffs received from the other agents. If there is a conflict between itself and one or more other agents on the choice of blocks, the agent attempts to resolve this. An agent will give way to the other agent(s) if it does not value the position as highly as any other conflicting agent (i.e., if it does not need the block as badly as other agents). An agent gives way by using the next move on its preference list. Once it gets to the end of its list, it wraps around to the start. If there is a tie for the highest value on a block, the agents involved in the tie go into a "lotto" mode, where they each randomly pick a number, with the highest number winning the right to keep the block. The losing agents then choose the next move on their Preference

list. This procedure is iterated until no conflicts exist. At this point the agents implement their moves. In a slight variance to Chapter 3's implementation, an additional stage was added at the end of the negotiating session where each agent observes if a block higher up on its preference list to the one they are to move to has become free. If it has, the agent changes its desired move, and communicates this to the other agents. If need be, negotiation is started again between the agents, if more than one wants to move to the unoccupied block. In this manner, an approximation to the global optimum is performed. This sequence can be iterated a number of times, which can be seen to create many messages between agents, and thus a high usage of the communication medium.

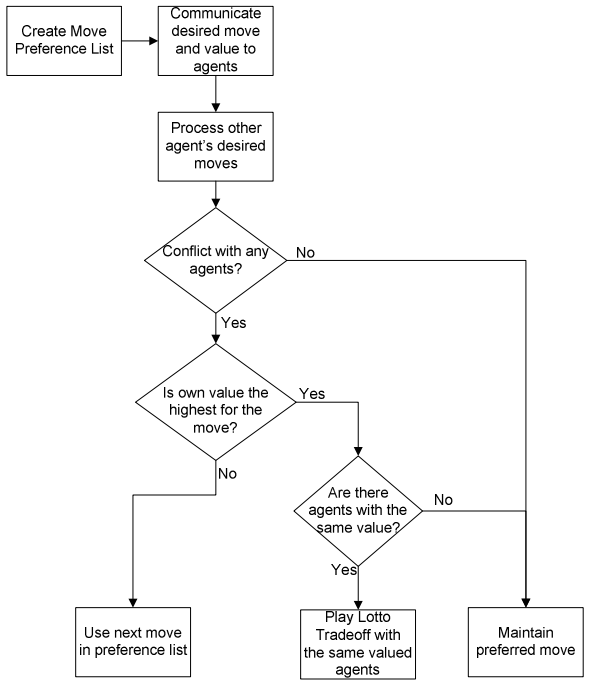


Figure 15 Agent Negotiating Algorithm

Implementation

The system is implemented in MATLAB. NETLAB [18] is used to implement the neural networks for the Captor Agents. A two-layer MLP architecture is used for the networks. The networks use a Logistic function for the output units. A variable number of hidden units, up to a maximum of 50, are used depending on the amount of data the network has seen.

Since NETLAB does not directly support on-line training, an alternative scheme is used. Every time the agents negotiate a solution or the global agent solves a problem, the board position is recorded, as well as the moves agreed upon through negotiation. This data is

added to the set of training data. After each game is complete, the agents are re-trained using the updated training data.

Results

The simulations were run for 400 games for the centralised system and 1000 games for the negotiation system experiment. The positions of the captors and the fugitive agents are randomised on the board at the beginning of each game. This is done in order to create a wide sampling of all the board combinations for the learning algorithms.

We consider the number of moves per game as the cost for that game, since the actual objective of the game is to capture the fugitive as quickly as possible. By this formulation, we can ensure that learning in the agents is not having a detrimental effect to the objective of the game. We do this by recording the average number of moves per game (cost) *without* learning enabled over large number of games. This is expressed as the *baseline cost.* The average number of moves per game is taken, as the precise number of moves per game can vary depending on game factors such as starting positions, previous moves, etc. After learning is enabled, we can compare the game cost with learning to the baseline cost. In these experiments, we expect the cost to remain approximately the same before and after learning. This is because we expect the agents to learn the moves they make from the fixed strategy presented in each case. The agents are not expected to create new strategies, but carry out the strategies which have been observed from the negotiation protocols or centralised agent. However if the cost, after learning is enabled in each experiment, increases significantly then learning can be said to have an overall negative effect on the community. For these cases, the neural networks in the agents would be suggesting legal moves, but the moves would be un-strategic, or possibly random (which would be expected while the neural networks are still untrained).

What is expected is that the communication levels, between the agents and the central controller and between agents in the negotiation system, shows an overall decrease.

For the negotiating system we also record the average number of conflicts that arise per game, before and after learning is introduced into the system. For each conflict that occurs, a negotiation session must be started. To normalise the measurement of bandwidth usage, we define one negotiation session as using one unit of bandwidth. Therefore, for each conflict that is avoided, one bandwidth unit can be saved. The measurements show the percentage reduction in conflicts in the learning system from the non-learning system, thus effectively showing the reduction in bandwidth usage.

Centralised System

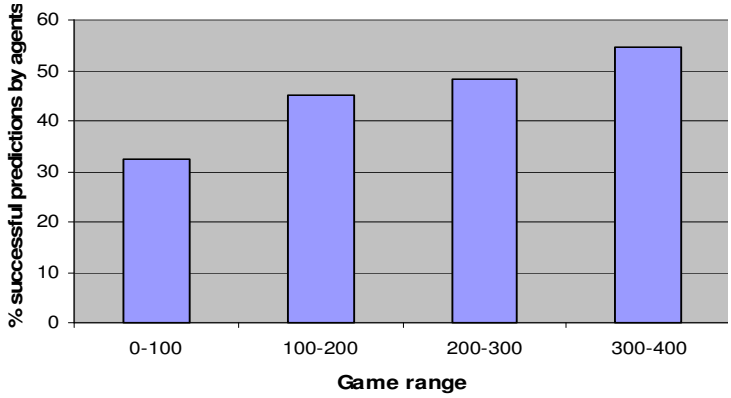


Figure 16 Results of the Centralised system simulation

The results of the simulation are summarised in Figure 16. The data is averaged over intervals of 100 games each, for clearer presentation. We show here the percentage correct moves made by the neural networks of the total moves per game over the simulation time. In other words, the graph shows the percentage of game moves made *without* invoking the central controller. This can be clearly seen to increase over the simulation time to a maximum of 54.75% of the total moves made. However, simulation beyond this point resulted in very little increased performance by the neural networks.

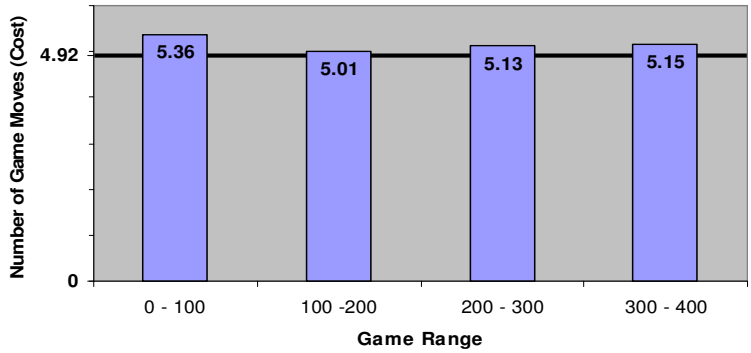


Figure 17 Average number of captor moves per game, averaged over 100 game intervals

The *baseline cost* was found to be 4.92 Moves/Game after simulation of 400 games without learning enabled. The standard deviation of this is σ =1.6649 Moves/Game. In Figure 17, the baseline cost is plotted as the bold horizontal line. The bars show the cost per game, averaged over 100 games per interval, with learning enabled. The cost with learning is slightly higher than the baseline cost, but is still within 1 standard deviation.

Negotiating System

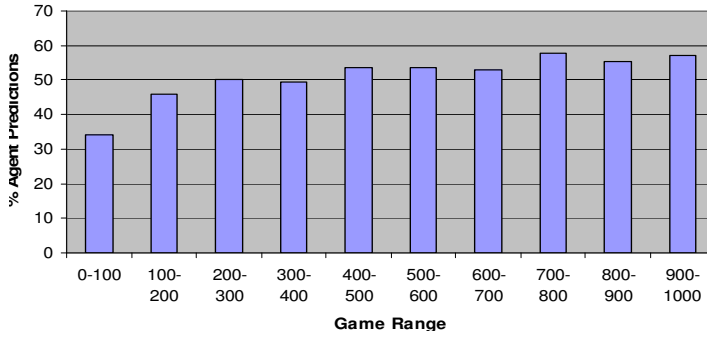


Figure 18 Results of the Negotiation System simulation

In order to have a benchmark figure to evaluate the contribution of learning to this system, we first ran a simulation of 500 games without using learning. This enabled us to obtain an average for the number of conflicts per game in normal circumstances and the baseline cost for this system. We found the average number of conflicts over the 500 games to be 7.02, and the baseline cost to be 4.874 Moves/Game with a standard deviation of σ = 1.7402. Learning was then introduced into the system. In Figure 18, we see that the percentage of moves learned by the agents is similar to that of the centralised system, with a peak of 57.6% successful predictions. The improvement in the moves learned by the agents' shows only moderate change after 700 games.

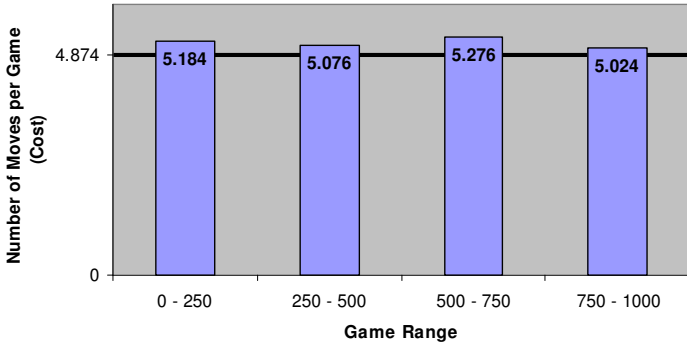


Figure 19 Cost of games in the negotiating system with learning enabled. The horizontal line shows the baseline cost without learning

Figure 19 shows the cost per game averaged over 250 game intervals with the baseline cost plotted as the bold horizontal line. Once again, the cost with learning is slightly higher than the baseline cost, but is still within 1 standard deviation.

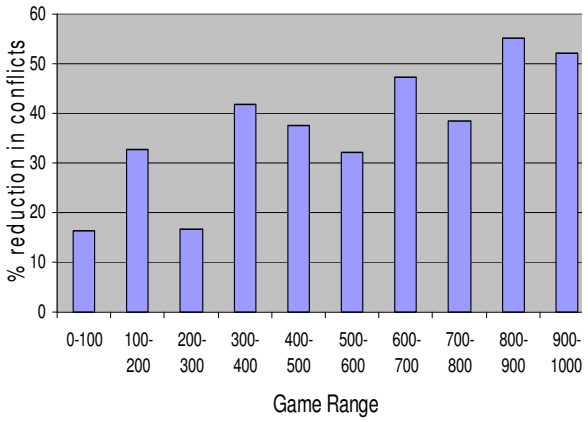


Figure 20 Results showing percentage reduction in agent conflicts compared to the measured average of 7.02 conflicts per game without learning

In Figure 20, we see the percentage reduction of conflicts in the learning agent system, compared with the number of conflicts in the non-learning system. Although the pattern seems erratic, it is clear that the overall trend is of increasing conflict reduction. It can be seen that the number of conflicts reduces by at least 30% after the 300 games mark, and reaches peaks of up to 55% reduction in conflicts. This is significant, since reducing conflicts means reducing negotiations, which can be bandwidth intensive. Using the concept of 1 bandwidth unit per negotiation, we can say that the bandwidth reduction is equal to the reduction in the number of conflicts.

Discussion

The knowledge that the agents have obtained as a result of their learning can be thought of as obtaining a model of their environment, or more specifically, as a model of the behaviour of the other agents within the system. This is due to the fact that when the agents correctly predict moves using their neural networks, they are not only maximising their own goal, but taking into account the needs of the other agents in the system. This is despite the fact that they have no explicit knowledge of the behaviour or workings of the other agents: the knowledge that they have obtained is purely through observation and interaction. In the negotiation system when the agents are actively negotiating, the agents take no account of the moves that the other agents might make: they wait for another agent to actively inform of a problem. To programme into such algorithms the ability to anticipate other agent's moves would effectively mean that the agents would each have a global view of the problem, as well as having intimate knowledge of every agent's abilities and desires in the system. This would defeat the object of a Multi-agent system, where

each agent is expected to operate in a highly distributed environment, and having a limited viewpoint of the problem, as well as considerable duplication of abilities. Furthermore, it may not even be appropriate for each agent to have this knowledge, especially when the agents are divided into sub-teams [8].

After a large number of games, each agent will have observed a large variety of board positions, and for this system, the neural networks will tend to be the same. This is only because the agents in this system all have the same "abilities" or allowed moves. In a scenario where there are agents with different abilities, as for example the separate pieces in a chess game, the networks will each train on totally different output moves, and so will tend to be different even after observing an infinite number of moves. However, there is no reason why the agents should not still be able to anticipate each others moves, since they will be trained with this data.

On many of the occasions, the agents were not able to make a legal move due to the fact that some agents suggested more than one direction to move in at a time. This limited the effectiveness of the learning function. While this intuitively suggests that the agents are not yet well trained in the particular situation, this is often not the case. Both the global agent in the centralised system and the negotiation modules in the negotiation system do not always propose the same solution for a particular board position. This means that for a given board position, an agent may be trained with many possible moves. What is needed is a method to resolve the proposed multiple directions by an agent in a particular situation with the other agents to improve the effectiveness of the learning theory.

The slight increase in the cost per game from the baseline costs measured for each experiment may be in part due to the neural networks over-generalising moves to new situations, as well as for the above mentioned reason of multiple possible moves for one position. In this case, the agents will act uncoordinated if they choose differing strategies. This may lead to longer games.

Further Research

There is already considerable research in the field of agent learning. We believe that learning is not only useful for added functionality in agents, but also to enhance existing agent operation. The main problem in any learning system is often the presentation of the information to the system [20]. With other means of representation, it may be possible to increase the benefits of learning, and create more efficient agents. In complicated environments, which are common of multi-agent domains, it would be desirable, and

probably necessary, to have the agents automatically determine the relevant representation of information, as well as choosing the information that would help in the decision.

The problem of agents learning different strategies in the same situation needs to be addressed to improve the overall effectiveness of learning. A methodology whereby agents are equipped with a host of coordination strategies at design time, and then learn through interaction the appropriate local strategy to use in a given situation has been proposed by [23]. A mechanism to accommodate this type of agent behaviour will need to be investigated

The problems that we have implemented here are very simple in order to demonstrate a theory. This theory should now be tested on more complex systems, where perhaps the agents in the environment are heterogeneous, and agents have many more options on a particular move.

Conclusion

We have demonstrated here the ability of learning to significantly reduce the amount of communication in a multi-agent system. To do this, the agents effectively learn a model of the other agents in the community, and start to behave reactively, while still being cooperative. The advantage of this method is that such learning techniques can be applied to existing agent communities to improve the efficiency of the community.

The reduction in communication as shown here can have large benefits in certain situations, such as when the communication channel carries a significant monetary cost, or if the channel is not reliable or slow. All these factors contribute to the overall effectiveness of the agent community, and should be minimised.

References

[1] Maria Chli, Philippe De Wilde, Jan Goossenaerts, Vladimir Abramov, Nick Szirbik, Pedro Mariano, Rita Riberio, "Stability of Multi Agent Systems", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2003, Volume 1, 2003, Pages: 551-556

[2] Z. Ren, C.J. Anumba, "Multi-agent systems in construction – state of the art and prospects," Automation in Construction, no. 13, 2004, pp. 421-423.

[3] Hadelia,, P. Valckenaersa, M. Kollingbaumb, H. Van Brussel, "Multi-agent coordination and control using stigmergy," Computers in Industry 53, 2004, pp 75–96

[4] P. Mariano, R, L. Correia, Ribeiro, V. Abramov, N. Szirbik, J. Goossenaerts, T. Marwala, P. de Wilde. "Simulation of a trading multi-agent system", Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Tucson, Arizona,USA 2001, 3378-3384.

[5] T. Marwala, P. de Wilde, L. Correia, P. Mariano, R. Ribeiro, V. Abramov, N. Szirbik, J. Goossenaerts, "Scalability and optimisation of a committee of agents using genetic algorithm" International Symposia on Soft Computing and Intelligent Systems for Industry, Scotland, 2001.

[6] D. Fitoussi, M. Tennenholtz, "Choosing social laws for multi-agent systems: Minimality and simplicity", Artificial Intelligence 119, 2000, pp 61-101

[7] S. Sen, "Believing others: Pros and cons", Artificial intelligence 142, 2002, pp 179 -203.

[8] D. Kalofonos, T.J. Norman, "An Investigation into Team-Based Planning," in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 2004, pp 5590- 5595.

[9] E. Bonabeau, M. Dorigo, G Theraulaz, Swarm Intelligence – From Natural to Artificial Systems, Oxford University Press, 1999.

[10] K.P. Sycara. "Multiagent Systems". AI Magazine, American Association for Artificial Intelligence, Summer 1998. pp 79-92.

[11] J.J. Castro–Schez, N.R Jennings, X. Luo, NR Shadbolt. "Acquiring Domain Knowledge for negotiating agents: A case study", International Journal of Human-Computer Studies 61 (1), 2004, pp 3-31.

[12] N.R. Jennings, M. Wooldridge, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", Autonomous Agents and Multi-Agent Systems, 3, 2000. pg 285-12

[13] N.R Jennings, M. Woodridge,"Intelligent Agents: Theory and practice", The Knowledge Engineering Review, 10 (2), pg 115-152, 1995.

[14] Nils. J Nilsson, Artificial Intelligence: A new synthesis, Morgan Kaufmann Publishers, San Francisco, California, 1998.

[15] Alison Cawsey, The Essence of Artificial Intelligence, Prentice Hall Europe, 1998.

[16] M. Jordan, C. Bishop, Neural Networks, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Last accessed 12 August 2003, ftp.publications.ai.mit.edu.

[17] M. Hagan, H. Demuth, O. De Jesus, "An Introduction to the use of Neural Networks in Control Systems", International Journal of Robust Control, Vol. 12, No. 11, September 2002, pp. 959-985

[18] I. Nabney, Netlab – Algorithms for Pattern Recognition, John Wiley and Sons, New York, 2001.

[19] J.A. Marshall, Z. Lin, M.E. Brouke, B.A Francis, "Pursuit Strategies for Autonomous Agents", Proceedings of the 2003 Block Island Workshop on Cooperative Control. Springer-Verlag Series: Lecture Notes in Control and Information Sciences, 2004.

[20] Sebastian Thrun, "Learning to Play the Game of Chess", Advances in Neural Processing Systems 7, 1995.

[21] G. Tesauro, "Programming backgammon using self-teaching neural net", Artificial Intelligence 134(2002) pp 181-199.

[22] G.Tesauro, "Practical Issues in Temporal Difference Learning", Machine Learning 8 (1992), pp. 257-277.

[23] C. B. Excelente-Toledo and N. R. Jennings "The dynamic selection of Coordination Mechanisms" Autonomous Agents and Multi Agent Systems 9, 2004, pp 55-85

Chapter 5

CONCLUSION

The study presented in this thesis shows that learning can have a significant impact on the communication levels in a multi-agent system, and possibly improve the efficiency of such systems.

Firstly, the ability of agents to acquire their behaviour purely through learning is demonstrated through the simulations of the centralised system structure. In these cases, the agents learn appropriate behaviour through example. The advantage of this model is that agents may learn desired local behaviour through observation of a global goal. Here the global goal is for captor agents to maximally surround a fugitive agent in the game of Pursuit. The local goals of each captor agent are subtly different: Each captor must move as close as possible to the fugitive, but must not block any other agents to the detriment of the global goal. The ability of the agents to learn this behaviour to a reasonable degree is shown here.

Secondly, the reduction of communication between agents in the test systems is clearly demonstrated in the study. In instances where the communication medium between agents is costly, for example wireless communication between robots, as well as instances where communication delay is undesirable, this can be significant. Communication reduction also allows agent systems to behave more like swarm systems, with the attendant benefits of quicker reaction and reduced communications latency.

This research has shown that, in the case of centralised system structure, the agents are able to learn the goals of the system without explicitly programming any rules into these agents. Furthermore, in both the negotiation formulation and the centralised system, the agents can be seen to learn cooperative behaviour whilst reducing inter-agent or agent - controller communication. This would traditionally mean programming the agents with models of all the other agents in the system. In this research, the agents learn the behaviour of other agents through interaction and observation. In building agent communities where the formulation of local goals and modelling of agents behaviours can be problematic, this research may be of use.

A p p e n d i x A

EXTRA DATA FROM AGENT SIMULATIONS

Negotiating Agents Simulation

This section shows the results of the negotiating agents simulation in more detail. Table 1 shows the results after 1000 games simulation. The results are averaged over 100 game intervals. "Total moves" is the average number of moves to the end of each game in the interval; "Moves by Agents" is the number of moves made by the agents without negotiation being required (i.e. the moves predicted by each agent's neural network); "Conflicts" is the average number of times the agents had to negotiated per game (Note that the agents may negotiate more than once per move); "% moves by agents" is the percentage moves made by the agents of the total moves during each game; "% conflict reduction" is the percentage the learning has reduced the number of conflicts from the mean measure of 7.02 conflicts per game without learning.

Table 1: Detailed Results for the Negotiating Agents Simulation

Game range	Total moves	Moves by Agents	Conflicts	% moves by agents	% Conflict reduction
0-100	5.17	1.77	5.88	34.23597679	16.23931624
100-200	4.93	2.27	4.73	46.04462475	32.62108262
200-300	5.49	2.76	5.84	50.27322404	16.80911681
300-400	4.86	2.41	4.08	49.58847737	41.88034188
400-500	5.2	2.78	4.38	53.46153846	37.60683761
500-600	5.53	2.97	4.77	53.70705244	32.05128205
600-700	5.09	2.7	3.71	53.04518664	47.15099715
700-800	5.15	2.97	4.32	57.66990291	38.46153846
800-900	4.89	2.71	3.15	55.4192229	55.12820513
900-1000	5.09	2.91	3.37	57.17092338	51.99430199

Centralised system simulations

Here, the results of the centralised system simulations are shown in more detail. Figure 1 shows the number of successful moves made by the agents' neural networks per game. The dark horizontal lines show the average number of agent moves per 100 game intervals. From this, the effect of the learning is clearly seen. Figure 2 shows the total number of moves made per game (moves made by the agents and by the centralised system). As can be seen on this graph, the number of moves per game increases slightly in

the first 100 games, compared to the other game intervals. This can be attributed to the neural networks still learning the pattern of legal moves, and thus making essentially random moves, which merely prolong the game.

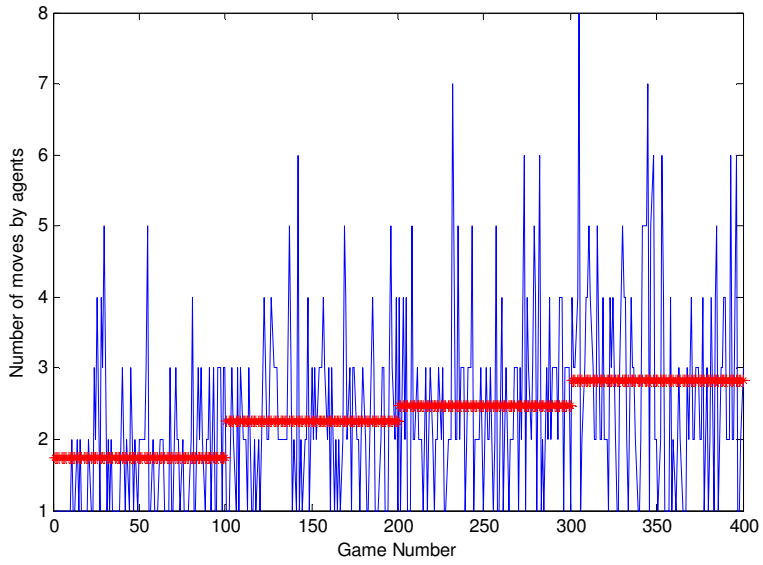


Figure 1 Results of the Centralised Agent simulations

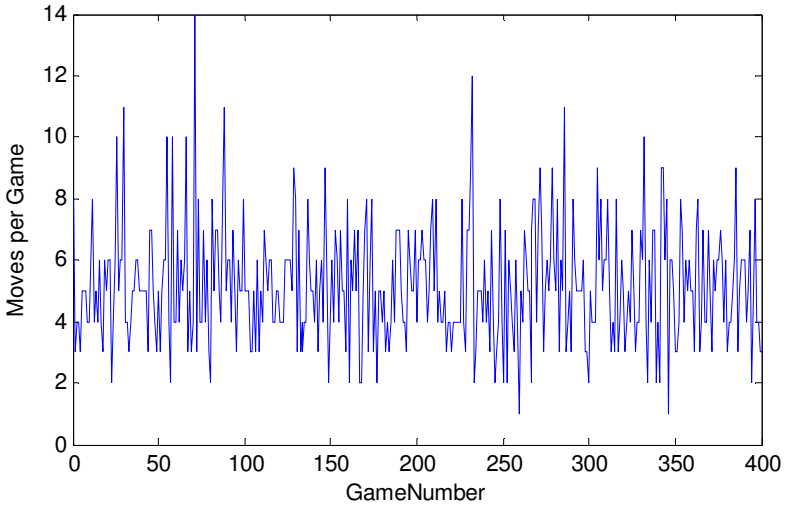


Figure 2 Total Moves per Game

Appendix B

SOURCE CODE

The source code for the research, and other material is given on a CD-ROM. The contents of the CD are as follows (Bold items are folders on the CD):

Source Code	-- The actual Code
Saves	-- MATLAB workspaces saved during simulation
Description.txt	-- Description of each saved item
Documents	
Thesis.doc	-- This document
PRASA.doc	-- Paper submitted to PRASA
ICCC.doc	-- Paper submitted to the ICCC Conference
JSCI.doc	-- Paper submitted to the JSCI journal